

Francesca Zerbato

Tackling Different Business Process Perspectives

Modeling data, time, and decisions in BPMN processes

Ph.D. Thesis

Submitted: December 17, 2018; Approved: March 26, 2019

Università degli Studi di Verona
Dipartimento di Informatica

Advisor
Prof. Carlo Combi

Co-Advisors
Prof. Barbara Oliboni
Prof. Mathias Weske

Università degli Studi di Verona
Dipartimento di Informatica
Strada le Grazie 15, 37134 Verona
Italy

Summary. Business Process Management (BPM) has emerged as a discipline to design, control, analyze, and optimize business operations. Conceptual models lie at the core of BPM. In particular, business process models have been taken up by organizations as a means to describe the main activities that are performed to achieve a specific business goal. Process models generally cover different perspectives that underlie separate yet interrelated representations for analyzing and presenting process information.

Being primarily driven by process improvement objectives, traditional business process modeling languages focus on capturing the control flow perspective of business processes, that is, the temporal and logical coordination of activities. Such approaches are usually characterized as “activity-centric”.

Nowadays, activity-centric process modeling languages, such as the Business Process Model and Notation (BPMN) standard, are still the most used in practice and benefit from industrial tool support. Nevertheless, evidence shows that such process modeling languages still lack of support for modeling non-control-flow perspectives, such as the temporal, informational, and decision perspectives, among others.

This thesis centres on the BPMN standard and addresses the modeling the temporal, informational, and decision perspectives of process models, with particular attention to processes enacted in healthcare domains. Despite being partially interrelated, the main contributions of this thesis may be partitioned according to the modeling perspective they concern.

The temporal perspective deals with the specification, management, and formal verification of temporal constraints. In this thesis, we address the specification and runtime management of temporal constraints in BPMN, by taking advantage of process modularity and of event handling mechanisms included in the standard. Then, we propose three different mappings from BPMN to formal models, to validate the behavior of the proposed process models and to check whether they are dynamically controllable.

The informational perspective represents the information entities consumed, produced or manipulated by a process. This thesis focuses on the conceptual connection between processes and data, borrowing concepts from the database domain to enable the representation of which part of a database schema is accessed by a certain process activity. This novel conceptual view is then employed to detect potential data inconsistencies arising when the same data are accessed erroneously by different process activities.

The decision perspective encompasses the modeling of the decision-making related to a process, considering where decisions are made in the process and how decision outcomes affect process execution. In this thesis, we investigate the use of the Decision Model and Notation (DMN) standard in conjunction with BPMN starting from a pattern-based approach to ease the derivation of DMN decision models from the data represented in BPMN processes. Besides, we propose a methodology that focuses on the integrated use of BPMN and DMN for modeling decision-intensive care pathways in a real-world application domain.

Contents

1	Introduction	9
1.1	Business Process Modeling	9
1.2	Research Contributions	12
1.3	Overview of this Thesis	15

Part I Background

2	Foundational Concepts	21
2.1	Introduction to Business Process Modeling	22
2.1.1	Business Process Model and Notation	25
2.1.2	BPMN Semantics and Petri Nets	29
2.1.3	Principles of Well-Structured Process Design	33
2.2	Decision Modeling	35
2.2.1	Decision Model and Notation	35
2.3	Process Modeling in Healthcare	38

Part II The Temporal Perspective

3	A Modular Approach for Defining Duration Constraints on Business Processes	47
3.1	Modeling Temporal Aspects in BPMN processes	50
3.2	Process Models for Specifying Activity Duration	53
3.3	Specifying Simple Duration Constraints	55
3.3.1	Specifying the Duration of an Activity with Boundary Events	59
3.3.2	Specifying Simple Duration Constraints of Process Regions	64
3.4	Specifying Deferred Activity Duration Constraints	74
3.5	Specifying Shifted Duration Constraints	80
3.6	Detecting and Managing Duration Violations	86
3.7	Process Verification with Time Petri Nets	91
3.8	Conclusion	98

4	Modeling and Enforcing Temporal Constraints in BPMN Processes	103
4.1	Foundations of Timed Automata	104
4.2	Catheter-Related Bloodstream Infections	105
4.3	Modeling Temporal Constraints in BPMN	110
4.4	Mapping BPMN onto Timed Automata	114
4.5	Concluding Remarks	118
5	Managing Decision Tasks in Time-aware Business Process Models	121
5.1	The Treatment of Knee Osteoarthritis as a Motivating Example	122
5.2	Characterization of Time-aware BPMN Processes	124
5.2.1	Specification of Temporal Properties and Constraints	125
5.2.2	Specification of Decisions: A Novel View on Decision Outcomes	126
5.2.3	Controllability of a Time-Aware BPMN Process	128
5.3	Dynamic Controllability Checking	128
5.3.1	A Short Introduction to CSTNUds	128
5.3.2	Mapping Time-Aware BPMN onto CSTNUd	131
5.4	Conclusion	136
6	Related Work	137
6.1	Temporal Constraints: Modeling and Management	137
6.1.1	Expressing Temporal Constraints in BPMN	139
6.2	Discussion	141
6.2.1	Modeling Duration constraints in BPMN	141
6.2.2	Time Petri Nets, Timed Automata, Timed Game Automata and Temporal-Constraints Networks	144

Part III The Informational Perspective

7	Conceptual Modeling of Processes and Data: Connecting Different Perspectives	151
7.1	Bridging the Gap between Processes and Data	152
7.1.1	Motivations and Open Research Questions	153
7.1.2	The Activity View to connect Processes and Data	155
7.1.3	Novel Conceptual Insights	159
7.2	A Relational Approach based on the Activity View	161
7.2.1	A Relational Approach for Modeling the Data Perspective	166
7.2.2	Querying Connected Process and Data Models	171
7.3	Conclusion	181
8	Conceptual Modeling of Inter-dependencies between Processes and Data	183
8.1	Introduction and Motivation	183
8.2	Foundational Concepts	185
8.3	Inconsistencies among Data Operations	189
8.3.1	Discovering Inconsistencies Considering Data Instances	191
8.4	Conclusion	197

9 Experimental Evaluation of the Activity View 199

9.1 Experiment Planning and Design 199

9.2 Results, Analysis, and Discussion 210

10 Related Work 217

10.1 Linking Data to Business Processes 218

10.2 Unravelling Data Flaws in Process Models 221

Part IV The Decision Perspective of Healthcare Processes

11 Deriving Decision Models from Process Models 229

11.1 Motivating Example and Foundations 231

11.2 A Stepwise Pattern-based Approach 235

11.3 Identification of Process-Related Data Used for Decision-Making 237

11.3.1 Analysis of data representation in BPMN 237

11.3.2 Other Kinds of Data Used for Decision-Making 242

11.4 Decision Patterns for the Data Perspective of BPMN 243

11.5 Mapping BPMN patterns to DMN DRDs 249

11.6 Applying the Pattern-based Approach to Real-World Process Models .. 252

11.6.1 The Reference Healthcare Domain 252

11.6.2 Identification of Decision Patterns in a Process Model 255

11.6.3 Mapping of BPMN patterns to DRD fragments 255

11.6.4 Post-Processing of Decision and Process Models 257

11.6.5 Empirical Findings 261

11.7 Concluding Remarks 264

12 A Methodological Framework for the Integrated Design of Decision-intensive Care Pathways 265

12.1 The Methodological Framework 268

12.1.1 Overview 268

12.1.2 Design: modeling processes and decisions 272

12.1.3 Dealing with process-related information and knowledge 274

12.1.4 Integrating simulation in the design of care pathways 276

12.1.5 Implementation and Enactment 278

12.1.6 Managing temporalities of care pathways 279

12.2 Design and simulation of care pathways for COPD 280

12.2.1 Chronic Obstructive Pulmonary Disease 281

12.2.2 BPMN-based Modeling of Care Pathways for COPD 282

12.2.3 COPD Diagnosis and Assessment 285

12.2.4 Ordinary Management of Stable COPD 289

12.2.5 Management of COPD Exacerbations 292

12.2.6 Modeling Clinical Decisions with DMN 296

12.2.7 Dealing with Data and Process Indicators 299

12.2.8 Simulation of COPD Processes 302

12.3 Conclusions 306

13 Related Work	309
13.1 Process and Decision Modeling	309
13.2 BPM and Healthcare Process Modeling	312
14 Concluding Remarks and Future Work	319
Publications	325
List of Acronyms	329
References	331

Introduction

Chapter 1 introduces the reader to this doctoral thesis.

The chapter begins with an overview of process-aware information systems aimed to frame the domain of business process modeling and the related research problems addressed in this thesis. Then, in Sect. 1.2 we outline the main research contributions, while in Sect. 1.3 we describe the structure of this thesis.

1.1 Business Process Modeling

Nowadays, processes are considered strategic assets in business and information technology (IT) related fields. Informally, a business process is defined as a set of linked activities that are performed in coordination in an organizational and technical environment to jointly realize a specific business goal [1].

Over the past two decades, business processes have internationally been recognized as key instruments for understanding and improving the way a company or organization operates and produces value.

As a result, business process management (BPM) has emerged as a novel disciplinary approach extending workflow management by combining insights from business administration, organization theory, and computer science [2, 3]. BPM is a systematic and structured approach including “concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of business processes” [1]. Being at a crossroads of disparate viewpoints, BPM combines the interests and vision of IT specialists, business managers, and industrial engineers.

Information technology and, especially, information systems cover an important role in BPM [1]. Indeed, the activities composing a business process must be supported by information systems, regardless of whether they are carried out manually or in a completely automated fashion.

Information systems that are “process-aware” and focus on the integration of IT and business processes go by the name of process-aware information systems

(PAIS). PAIS are software systems driven by explicit process models having the goal of coordinating and supporting agents in performing their activities [4].

Modeling has been central to the design and development of information systems long before the introduction of PAIS [5]. Accordingly, the notion of *process model* is foundational for BPM [6].

Broadly speaking, “a process model consists of a set of activity models and of execution constraints among them” [1]. Usually, process models serve a specific modeling purpose (e.g., process description, analysis, or enactment), which governs the way and the technique through which a process model is specified [3].

Process models are specified by means of a process modeling language, often providing a graphical notation for supporting their visualization.

At present, several process modeling languages exist, each one having different characteristics, strengths and limitations [7, 8]. Process modeling languages may be more or less formal, and the choice of which one suits best specific modeling goals and requirements depends on the audience to which the model is targeted: for example, if a model is addressed to humans, understandability is deemed important, whereas machines typically require more formal specifications [9].

Modeling purposes and languages are both closely related to the notion of process modeling perspective, that establishes which are the aspects or concepts of the business reality that should be captured, represented, and visualized in process model [10]. Process modeling languages differ in the extent to which their constructs support the representation of a certain perspective [9].

Traditional definitions of business process models, such as the one informally provided at the beginning of this chapter, bring the focus to the logical ordering of activities, that is, to the so-called control-flow perspective of business process models. Process modeling languages that focus on the representation of the process control-flow are called *activity-centric*.

An example of graphical activity-centric process modeling language is the Business Process Model and Notation (BPMN) [11], which has established itself as the standard notation for process modeling. Developed with the aim of bridging the gap between process stakeholders and developers, BPMN is currently maintained by the Object Management Group (OMG) and has gathered a lot of attention both in industry and academia due to its expressive power [8, 12].

However, processes belonging to complex real-world scenarios, such as the healthcare domain, comprise aspects that go beyond the process control-flow [13]. Thus, it is desirable that reference process modeling languages cover multiple intertwined perspectives [14, 15].

Taking inspiration from healthcare working environments, among the existing process modeling perspectives [14, 10] this thesis focuses on three important inter-related perspectives, i.e., the temporal, informational, and decision perspectives.

The temporal perspective deals with the (formal) specification and verification of time constraints in business process models [16, 17, 18]. The informational perspective, also called data perspective, focuses on the representation of information entities consumed, produced or manipulated by a process, and of

their interrelationships [9, 19]. Finally, the decision perspective describes how decision-making is coordinated within business processes, that is, it represents where decision-making takes place in the process and how decision outcomes affect process execution [20, 21].

Despite continuous research efforts addressing the modeling of temporal, data, and decision aspects in business processes, in activity-centric process models such modeling perspectives tend to be neglected and their representation is either not supported at all or it is “hidden” within the process control-flow.

This lack of support for modeling perspectives other than the process control-flow holds also for the BPMN standard [12, 22, 23], which nonetheless remains the most popular and influential process modeling language, widely used in industry, and supported by existing process modeling tools [24, 25].

This thesis is based on the assumption that increased process-awareness requires organizations to integrate process engineering methods with existing approaches for the management of temporal, data, and decision-making aspects among others. In this scenario, we advocate that process models should provide a way to represent different perspectives or to seamlessly integrate disparate viewpoints.

In this direction, a considerable number of studies have focused on proposing novel extensions, modeling languages, and paradigms to capture temporal, data, and decision aspects. However, we argue that the suitability of activity-centric process modeling approaches may still be improved to support the representation of different perspectives.

- *Temporal perspective.* The representation and handling of temporal constraints has long been recognized as a crucial factor for organizations and business enterprises [16, 26]. Several approaches in the fields of workflow management and BPM have focused on specific aspects encompassed by the time perspective. Among others, we recall the management of temporal exceptions [27], the definition of deadline-based escalations [28], and the formal verification of temporal constraints [29, 30, 31, 32]. Nevertheless, the support provided by existing process modeling and management approaches to the specification and run-time verification of temporal constraints is still lacking [17, 23, 33]. Unfortunately, the BPMN standard is no exception [34, 35, 36], despite including advanced modeling concepts, such as event and exception handling [11], which could be suitable to model foreseen events occurring during the process run-time.
- *Informational perspective.* In the field of information systems engineering it is common practice to combine data engineering methods with process engineering methods [37]. Data and processes are seen as two sides of the same coin when it comes to manage the core assets of an organization [38], and their integration brings several benefits in terms of improved process understanding, documentation, enactment, and consistency with IT systems.

Recent approaches dealing with the integration of data and processes follow the data-centric paradigm and are grounded on formal logic [39, 40, 41, 42], thus resulting very expressive, but harder to be understood by process designers and stakeholders compared to activity-centric ones [43, 44].

Hence, the connection between activity-centric processes and data, including the relationships of process data and the persistent data stored in enterprise databases, remains a “hot topic” in the BPM field [14]. In particular, at the conceptual level, process and data models are only rarely related [45] and languages such as BPMN provide only elementary ways to represent process access to persistent data sources [19, 22, 43].

- *Decision perspective.* Until a few years ago, decisions were modeled within business processes by encoding decision logic through control flow structures [20]. Then, the shift towards a separation of concerns [46] between process and decision models has led to the development of the Decision Model and Notation (DMN) [47] standard, aimed to complement BPMN for designing decisions made in process models.

Since the introduction of DMN, the common practice is to model decisions outside processes [20, 48, 49]. However, the consistent separation and integration of process and decision models is quite challenging [50], also because decisions involve both the control-flow and data aspects of a process.

In addition, given a process model including decision aspects, it is not easy for designers to determine which decision aspects should be extracted from the process model and put in a dedicated decision model. This is particularly true for decision-intensive domains such as healthcare [51], where processes have an intrinsic decisional character and decisions govern the process flow.

In addition, modeling complexity increases if we consider that the boundary between the temporal, informational, and decision perspectives is not always so crisp (e.g., temporal constraints may affect the way decisions are made, while the latter are based on data). Nevertheless, these challenges motivate the need for broader, multi-perspective research approaches, aimed to jointly improve the modeling of process-related aspects that span more than a single perspective.

1.2 Research Contributions

The contributions of this thesis lie in the proposal of novel modeling approaches based on reference and standard languages to foster the representation of the temporal [52, 53, 54, 55], informational [56, 57], and decision [58, 59, 60, 61] perspectives of activity-centric process models.

Starting from the limitations outlined in Sect. 1.1, this thesis takes BPMN [11] as reference language and investigates whether and how the rich expressive power of the notation may be exploited to capture temporal, data, or decision aspects of process models.

The choice of starting from BPMN as modeling language is mostly motivated by its expressiveness and extensibility, by its widespread use in real-world application domains, and by the existence of multiple editing tools supporting the standard notation. The plethora of existing process modeling languages discourages us from defining novel, ad-hoc approaches that would probably not be able to compete with established standards in terms of expressive power, ease of understanding, and tool support [24, 25]. In addition, BPMN base elements may be extended to support application-specific aspects and since the OMG maintains other modeling notations (such as DMN [47], the Case Management Model and Notation (CMMN) [62], and the Unified Modeling Language (UML) [63]), the connection of the latter ones with BPMN may be facilitated.

The contributions of this thesis may be partitioned according to modeling perspective they concern.

- *Temporal perspective.* This thesis tackles the temporal perspective of business processes by focusing on two main challenges: 1) the specification and run-time management of temporal constraints directly in BPMN and 2) the use of formal models either to validate the behavior of the designed BPMN processes or to verify temporal properties.

1) As for the specification of temporal constraints, this thesis introduces a modular approach to model different kinds of duration constraints for process activities and regions in BPMN process models. We design a set of well-structured duration patterns in BPMN that can be combined with the activities/process regions to constrain without altering the overall structure of the process.

Our proposal exploits the advanced event and exception handling mechanisms included in BPMN to capture the flowing of time, to realize synchronization between different branches of the same process, and to detect the violation of duration constraints at run-time. Being completely based on BPMN and its semantics, the proposed process patterns may be modeled, simulated, and executed by BPMN-compliant tools.

Instead, for representing more complex time constraints, such as time lags between events, and for dealing with contingent activity duration (i.e., whose actual value is observed only once the activity has completed), we add a temporal dimension to selected BPMN elements [55].

2) Mappings towards formal models are provided to define and validate behaviour of the proposed process models and to verify the dynamic controllability property [29, 64] of time-aware process models.

In detail, we define three different mappings of BPMN process models. We propose a mapping from BPMN onto time Petri nets (TPN) [65] to check the soundness of the obtained process models. Then, we consider mapping BPMN onto timed automata (TA) [66], to formalize the semantics of the proposed process models in an unambiguous way and to set the bases for their verification with existing model-checking tools. Finally, we consider mapping BPMN processes extended with symbols that represent temporal constraints

onto the Conditional Simple Temporal Network with Uncertainty and Decisions (CSTNUD) model [67] with the aim to assess whether a process is dynamically controllable [29].

- *Informational perspective.* In this thesis, we focus on the modeling of the connection between processes and persistent data, at a conceptual level. Our main contribution is the definition and experimental evaluation of the Activity View [57], a novel conceptual view aimed to capture the connection between a BPMN process model and the schema of a database, represented as a UML Class Diagram [63]. The goals of the Activity View are (i) to support process designers in modeling the operations performed by process activities on data stored in a database and (ii) to enable basic analysis of the interplay between connected models. As for (ii), this thesis shows how the Activity View may be used to analyze connected processes and data from at least two viewpoints. On the one side, we define a relational framework storing information about processes, data, and Activity Views, that can be queried to understand how data are used within one or more processes. Information about the structure of a process is retrieved with the help of a labeling algorithm and some sample queries are formalized in relational calculus and SQL. On the other side, we rely on the data operations captured by the Activity View to detect potential inconsistencies that may arise when data operations performed by different process activities and involving the same data instances follow an erratic order [56].
- *Decision perspective.* Following the recent introduction of the DMN standard, in this thesis we deal with the modeling of decisions made within BPMN processes, specifically focusing on the needs of real-world healthcare domains. Starting from the assumption that decisions have long been encoded into business processes [20], we take a closer look to the data used for making decisions typically represented in BPMN process models as data objects, data stores, or events, and define a pattern-based approach to ease the derivation of DMN decision models from the data perspective of process models. Then, we explore the benefits brought by the joint use of BPMN and DMN for modeling and standardizing decision-intensive care pathways, i.e., structured care plans that detail essential tasks that need to be carried out in the care of patients with a specific clinical problem [68]. To this end, we propose a methodological framework, inspired by the traditional BPM life-cycle [1], aimed to define reference steps for designing and simulating healthcare processes in a standard way that can easily be understood and shared by stakeholders. This case study allows us to also define contact points between different process perspectives and to elaborate on the overall contributions of this thesis with respect to a real, complex application scenario.

As previously mentioned, the boundary between the three outlined perspectives is not so sharp. Decision modeling embraces both temporal and data aspects, since decisions are based on data and they may affect or being affected by temporal constraints.

In this thesis, beside deriving decisions from the data perspective of process models, we consider the relation between temporal constraints and decisions both during constraint specification and dynamic controllability checking. In particular, we deal with a time constraint that affects the choice of alternative process paths based on the time distance between two event occurrences. Moreover, when dealing with dynamic controllability, we consider the possibility of reducing the outcomes of certain decisions in the process in order to satisfy temporal constraints.

1.3 Overview of this Thesis

This thesis is structured into four main parts, framed by an introductory and a concluding chapter. A graphical outline is provided in Fig. 1.1.

Except for *Part I*, which consists of a single chapter introducing foundational concepts, all remaining parts include a chapter dedicated to related work. At the beginning of each chapter, we report the references to the research works published during the doctoral program.

The contents of each part are detailed below.

- *Part I* consists of a single chapter, Chapter 2, which provides an overview of process modeling, decision modeling, and describes the application of BPM approaches to healthcare domains. In *Part I* we formally define the foundational concepts used in this thesis.
- *Part II* addresses the temporal perspective of process models and presents three different approaches for modeling and verifying temporal constraints. Chapter 3 describes a modular approach to model the temporal duration of activities and to manage their violation. In particular, we focus on the modeling of three different variants of duration constraints in BPMN and on the verification of the obtained models through time Petri nets [65]. Chapter 4 focuses on the modeling of more general temporal constraints, such as relative constraints and time variations that determine which among different alternative process paths should be chosen. We rely on BPMN signal-based communication to realize process models that capture such constraints, and provide a mapping onto timed automata [66] to validate their semantics and to set the bases for verification. Chapter 5 introduces the notion of time-aware BPMN process model, by also characterizing decisions based on the way their outcomes are managed at run-time. Then, we provide a formal mapping onto CSTNUds [67, 69] to verify that the process model is dynamically controllable.

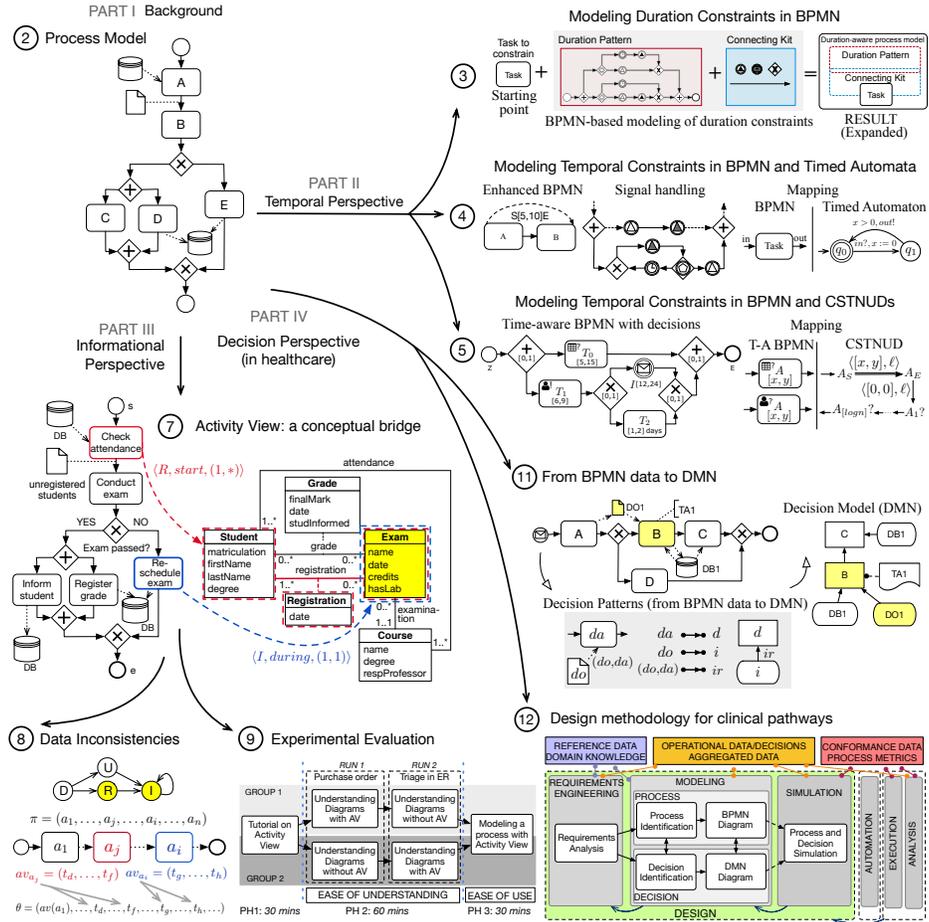


Fig. 1.1: Graphical structure of this thesis, showing the content of *Part I–Part IV*. Chapters are encircled. Chapters discussing literature related to each part are not included in this figure.

Finally, Chapter 6 presents related work and compares our contribution with relevant state-of-the-art approaches in the field.

- *Part III* deals with the informational perspective of process models, and describes our proposal to conceptually link a BPMN process model to the schema of a database, represented as a UML class diagram. In detail, Chapter 7 introduces our approach to realize such conceptual connection, called Activity View, and shows how it can be used to understand and analyze the way data are accessed and manipulated by a process.

Chapter 8 describes how the Activity View can serve as a starting point to detect inconsistencies among data operations performed by process activities, at different levels of abstraction.

Chapter 9 outlines the experimental evaluation that we carried out to have an estimate of how selected users perceive the Activity View. In particular, we focus on evaluating how easy our proposal is to understood and use, and what is its potential in improving the design of linked processes and data.

Chapter 10 provides an overview of recent work related to the integrated modeling of processes and data.

- *Part IV* tackles the decision perspective of business processes, particularly focusing on the complementary use of BPMN process models and DMN decision models.

Chapter 11 deals with the derivation of DMN decision models from the data perspective of BPMN process models. Our proposal is based on a set of process patterns that allow one to identify data connected to decision activities that may be used for decision-making. Then, we provide a mapping from such patterns to fragments of DMN decision models and discuss how to combine them to obtain a complete decision model that complements the starting BPMN process model.

Chapter 12 discusses the importance of using BPM in structured healthcare environments, where standardization is the main driver for process and decision design. In detail, we discuss the design of care pathways in BPMN and of the related decisions in DMN, and outline line the main steps of a methodological approach that exploits BPM techniques to deal with the complexity of healthcare processes. This chapter is related to Chapter 11 for the addressed topics (i.e., BPMN and DMN integration) and for the chosen healthcare domain on which the proposed methodologies are applied.

Chapter 13 discusses related work concerning the joint use of BPMN and DMN for process and decision modeling, and regarding the modeling of clinical guidelines and care pathways in decision-intensive healthcare domains.

Finally, in Chapter 14 we conclude by summarizing the research work presented in this thesis, discussing limitations, and providing an overview of future work.

Part I

Background

Foundational Concepts

Business process models should have a formal foundation. [...] It is desirable that a business process model can be understood by the various stakeholders involved in an as straightforward manner as possible [6].

Business process modeling lies at the core of BPM and consists of the explicit representation of business processes through models that follow the specific purpose for which they are created [1].

Process models are influenced by the chosen modeling technique. A modeling technique consists of a modeling language and a modeling method [3]. On the one hand, a modeling language is made up of a syntax, a semantics and, possibly, a notation providing a set of graphical symbols used to visualize the model. On the other hand, a modeling method defines the methodological procedure in which a certain modeling language can be used.

Nowadays there exist a plethora of languages for business process modeling. Among them, the Business Process Model and Notation (BPMN) [11] has established itself as the leading standard for process design and succeeds in supporting the modeling of the logical and temporal ordering of activities, i.e., the so called control-flow perspective [70]. However, since the execution semantics of BPMN is provided in an informal way, formal approaches are needed to disambiguate the behavior of BPMN modeling elements. Good candidates in this direction are Petri nets [71, 72], as BPMN semantics is based on token flows [11] and efficient static analysis techniques are available for Petri nets [73].

Beside the control flow of the process, further perspectives represented in process models have gained attention in the past decade. Most notably, the data perspective and the decision perspective have respectively attracted the interest of researchers dealing with scenarios where processes are mostly data-aware [74] and decision- or knowledge-intensive [75].

Indeed, in complex contexts such as healthcare process and decision aspects are being increasingly integrated for improving organizational efficiency and ef-

fectiveness [13, 26, 76]. In this scenario, the DMN standard [47] was proposed to possibly complement BPMN for designing decisions related to process models.

In this chapter, we recall some basic principles of process modeling and introduce the languages for business process and decision modeling used in this thesis. In detail, Sect. 2.1 reviews common process modeling paradigms and perspectives, focusing on the BPMN standard [11] and on the theory of Petri nets [71, 72, 77, 78]. Sect. 2.2 introduces decision modeling in DMN. Finally, Sect. 2.3 describes which are the main challenges to overcome when modeling processes in sophisticated clinical and healthcare scenarios.

2.1 Introduction to Business Process Modeling

A model is a simplification of an original, developed with the aim to achieve a specific goal for a targeted audience and expressed in a particular language [79].

Business process modeling is the human activity of creating a business process model [3]. Informally, a business process model consists of a set of activity models and execution constraints between them [1].

Being created and used to accommodate various goals, business process models represent only the aspects of reality that are considered relevant to the modeling purpose [3].

Most often, process models represent the business in a way that facilitates human understanding and communication among stakeholders, at different levels of abstraction [1]. Besides, they support both process improvement through reusability and comparability, and process management by enabling reasoning and forecasting. Finally, process models act as blue print for process execution and have a high potential for business analysis.

Beside being influenced by their final goals and objectives, business process models vary also upon the modeling paradigm and language used to specify them. The recent focus on PAIS has led to the development of a wide range of business process modeling paradigms and languages, each one having advantages and shortcomings with respect to specific application contexts [70].

In the first place, process modeling approaches may be distinguished based on whether they follow the *activity-centric* paradigm (sometimes also referred to as process-centric paradigm) or the *data-centric* one.

According to the activity-centric paradigm, a process is composed of activities representing units of work and control flow elements determine the order of activity execution. In activity-centric process models, modularity is achieved through activity decomposition into sub-processes.

Activity-centric process modeling languages may be further distinguished into imperative and declarative ones.

Imperative process models strictly prescribe how work is carried out, thus requiring all execution alternatives to be explicitly specified in the model before the execution of the process [80]. Accordingly, imperative process modeling languages focus on defining sequences of activities representing a business goal.

Underpinning imperative notations is a “closed world” assumption, meaning that the process model captures all the admissible activity flows and disallows all the unspecified ones [81].

Examples of well-established graph-based imperative process modeling languages, employed in both industry and academia, are approaches based on (high-level) Petri nets [65, 72, 78, 82, 83] and other flow-based modeling languages, such as Yet Another Workflow Language (YAWL) [84], event-driven process chains (EPC) [85, 86], UML activity diagrams [63], and the Business Process Model and Notation (BPMN) [11].

Alternatively, the declarative paradigm calls for the definition of constraints or rules that determine how the process shall be executed. Declarative process modeling languages focus on the logic that governs the interplay between the actions and objects of a process, by describing the activities that can be performed and the constraints that prohibit undesired behavior [87].

Examples of declarative process modeling languages are Declare [88], BPMN-D [81], Dynamic Condition Response Graphs [89], and the Guard-Stage-Milestone (GSM) approach [41].

In general, the boundary between imperative and declarative process modeling is not so sharp, as all existing process modeling languages lie somewhere on the spectrum from a less to a more imperative (declarative) nature and mixtures of declarative and imperative process modeling styles have been proposed [81].

Whereas the imperative process modeling paradigm has proven suitable in the context of regular and predictable processes, where variations are limited and well-scoped, declarative approaches seem to be more suitable to represent processes that deviate frequently from the prescribed execution path [90]. Nevertheless, imperative process models seem to be more comprehensible than their declarative counterparts, probably because stakeholders are more familiar with this paradigm [80].

Alternatively to the activity-centric paradigm, processes may be specified using the data-centric paradigm, which focuses on defining the data required by process activities. The latter ones are defined in terms of the changes they make to the data and the process progresses based on data availability.

Among them, artifact-centric models have emerged as a means to combine data and process aspects in order to represent the informational entities that are valuable for the organization, together their evolution within the process [91].

Pioneered by IBM Research, the artifact-centric approach proposes business artifacts to model key entities considered relevant for the overall business goals [91]. An artifact consists of an information model that holds relevant data, and of a life-cycle that describes both the possible changes to the information model and the interactions with other artifacts, i.e., artifacts are inter-linked [92]. The life-cycle of an artifact consists of different states. Transitions among states are triggered by events coming from human resources acting on them or from the interaction with enterprise systems [37]. The life-cycle of an artifact may be described imperatively by means of a finite state machine or declaratively with the help of the GSM meta-model [81].

Another approach for the formal specification and verification of data-centric processes is based on relational data-centric dynamic systems (DCDSs), i.e., “systems where both the process controlling the dynamics and the manipulation of data are equally central” [93]. A DCDS includes a data layer, holding the relevant information to be manipulated by the system, and a process layer formed by invocable actions and by a process based on them. Such a process characterizes the dynamic behavior of the system.

At present, support towards data-centric process modeling is still quite limited, since existing process engines [24, 94, 95] and commercial products from major software vendors like SAP, Oracle, and IBM are activity-driven. Besides, in many application domains where the process flow stems from ordered activities, data-centric approaches lacks major evidence of applicability [96].

In general, process models encompass different perspectives, each one offering an alternative way to describe the same concept [10, 14]. Thus, beside being characterized by modeling paradigms, business process modeling languages can also be classified based on the extent to which they support the modeling of different business and system perspectives [5, 97].

According to the conceptual framework presented in [9], process modeling languages should be able to support one or more of the following perspectives:

- the *functional* (or *process*) perspective represents what process elements are being performed (i.e., the process control-flow) and what are the related flows of informational entities. Typical notations used in the functional perspective include flow diagrams [7];
- the *behavioral* perspective represents when and how process elements are performed through feedback loops, iteration, decision-making and timing conditions, and other criteria;
- the *organizational* perspective represents where and by whom process elements are performed, the physical communication mechanisms used to transfer entities, and the physical media and locations used to store entities;
- the *informational* perspective represents the informational entities produced or manipulated by a process and the relationships among them. These entities include pure data, artifacts, and products [7].

In this thesis, we focus on the informational perspective of process models and distinguish two further dimensions of the behavioral perspective that can be seen as independent perspectives: the *temporal perspective*, addressing the modeling and checking of temporal properties and constraints [17, 18], and the *decision perspective*, dealing with the modeling of business decision-making [20, 49, 50].

Some existing process modeling languages, such as YAWL [84], EPCs [85, 86], UML activity diagrams [63], and BPMN [11] combine the functional and behavioral perspectives [10].

Among them, the Business Process Model and Notation has emerged as the standard language for graphically capturing business processes, especially in terms of domain analysis and high-level system design [12]. In addition, it has been going through a practice-driven maturing process and it is supported

by a variety of tools. Nevertheless, BPMN support towards the informational, temporal, and decision perspectives remains limited [23, 35, 43, 45].

Because of its widespread application and practical interest, in this thesis we chose to use BPMN [11] in its current version 2.0 for representing process models.

In particular, we consider dealing with organizations that already rely on a functioning information system, and operate through a set of defined *activity-centric process models*, represented in BPMN. Our choice was motivated by two main reasons: (i) on the one hand, we were discouraged by the lack of major empirical evidence regarding the applicability and usability of data-centric approaches, which are still far from being used within real organizational contexts; (ii) on the other hand we were engaged into investigating whether the support of BPMN for the perspectives mentioned above could be improved.

2.1.1 Business Process Model and Notation

Developed under the coordination of the OMG, BPMN is a graphical notation that can be suitably applied for modeling processes in various application domains in a standard manner. Indeed, BPMN supports a complete range of abstraction levels, spanning from less detailed business levels to more technical software technology ones [1].

BPMN allows designers representing three basic types of sub-models: processes, choreographies, and collaborations [11]. Among these, the business processes internal to a specific organization constitute the main focus of this thesis.

In BPMN a business process is defined as sequence of *activities* or *events*, connected by a *sequence flow* (also called *control-flow*), that denotes their ordering relations. Flow routing is realized by *gateways*, which allow to split the sequence flow into multiple paths and merge them.

Processes are visualized by means of graphical diagrams, which can be defined at different levels of abstraction, depending on the audience they are targeted to. Operational semantics is defined through the concept of tokens that traverse the process flow and enable the encountered flow elements. Depending on the semantics of traversed flow element, the number of tokens in a process can vary, as tokens are continuously generated and consumed by process elements. In BPMN, a token is a theoretical concept that is used to help defining the behavior of a process that is being performed [11].

BPMN diagrams provide an exhaustive set of modeling elements, which is partitioned into five main categories: (1) Flow objects, (2) Data, (3) Swimlanes, (4) Artifacts, and (5) Connecting Objects.

(1) The basic modeling elements (or flow objects) of a BPMN process are activities, events, and gateways.

Activities identify units of work that is performed within the process. In BPMN there are two kinds of activities: *tasks* that are atomic units of work that cannot be broken down to a finer level of abstraction, and *subprocesses* that are compound activities whose internal details are modeled using other elements.

Graphically, tasks are depicted as rectangles with rounded corners and labeled by their name. Subprocesses can either be represented as collapsed, i.e., as tasks decorated by a “+” sign, or they can be expanded to show internal details.

Events represent facts that occur instantaneously during process execution and that affect the sequence or timing of process activities. They are visualized as circles, which may contain a marker to diversify the kind of event trigger. Events can be distinguished based on their triggering behavior or their position in the process. Depending on their triggering behavior, events can either throw or catch a result. The throwing and catching of an event are comprehensively referred to as *event handling* [11]. Usually the (sub)processes that focus on handling events are called “event handlers”. Depending on their position in the process events are classified into *start*, *intermediate*, or *end* events. By convention, start events initiate a process instance, end events conclude it, while intermediate events indicate where something happens somewhere between the start and end of a process. When attached to an activity boundary, *interrupting* intermediate events interrupt the task they are attached to, whereas *non-interrupting* ones initiate a new process path (*exception flow*), which runs in parallel to activity execution.

Gateways are elements in the process used to control the divergence and convergence of the sequence flow, either according to data-based conditions or event occurrence. Graphically, they are shown as diamonds with an internal marker that differentiates their routing behavior. Symbol \oplus denotes parallel gateways, i.e., AND-split and AND-merge nodes, whereas symbol \times identifies a data-based exclusive gateway (XOR-split and XOR-merge nodes), i.e., a point in the process where a condition must be evaluated in order to choose one process path out of more alternative ones. Finally, an encircled pentagon denotes an event-based gateway, i.e., a routing point in the process flow where event occurrence determines which is the path to follow. In this case, when a process path is chosen, all the others are discarded.

(2) Data are graphically represented as *data objects*, having the shape of a document with a bended angle, and *data stores*, depicted as a cylinder. Data objects describe volatile information that is needed as an input for activities to be performed, or is produced during activity execution. Data stores represent access to a persistent storage that persists beyond the scope of the process.

(3) Swimlanes provide a graphical account for participants in a process and are used to group other flow objects under a resource perspective. *Pools* represent the participants of a process. The assignment of a resource to an activity is done by placing the activity within the selected pool. *Lanes* are used to partition pools in order to organize and categorize activities.

(4) Artifacts are used to provide additional information about the process. In this thesis, we will make use of *text annotations* to provide additional information in natural language to help the reader of a process model.

Finally, (5) connecting objects represent different ways of linking flow objects to each other or to data elements. *Sequence flow* edges connect any two flow objects (activities, events, gateways). *Data flow* edges connect flow nodes,

especially activities, to data elements (data object, data store). Message flow edges connect send events/tasks contained in one pool with receive events/tasks in another pool and represent message exchange.

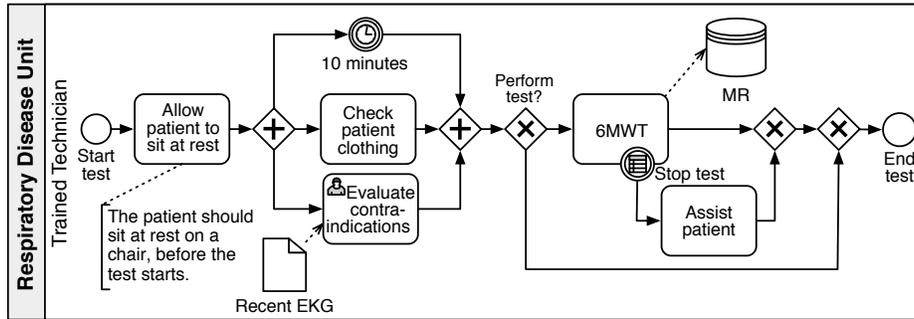


Fig. 2.1: BPMN process diagram summarizing some preliminary stages for the 6-Minute Walk Test, simplified and adapted from [98].

To visualize the graphical notation of a BPMN process model, let us consider the 6-Minute Walk Test [98] as an example, depicted by the process diagram of Fig. 2.1.

Example 2.1 (6-Minute Walk Test). The 6-Minute Walk Test (6MWT) is a clinical test used to objectively evaluate the functional exercise capacity in patients affected by chronic obstructive pulmonary disease (COPD), a progressive respiratory illness characterized by irreversible airflow limitation [98, 99]. The test is used to “measure the distance that a patient can quickly walk on a flat, hard surface in a period of 6 minutes” [98]. The patient is instructed to walk as far as possible in the given time frame, assisted by a trained technician. This test is typically used to evaluate exercise tolerance in chronic respiratory diseases and heart failure, and it allows physicians to monitor pulmonary and cardiovascular functions.

The 6MWT is typically performed in a dedicated hospital hallway, as timely and appropriate emergency measures must be enacted in case of complications. The test must be immediately stopped if the patient shows signs of chest pain, intolerable dyspnea, leg cramps, staggering, diaphoresis, and pale or ashen appearance [98]. In this scenario, the trained technician must take blood pressure, pulse rate, and oxygen saturation.

As shown in Fig. 2.1, the test is conducted in a Respiratory Diseases Unit, represented as a BPMN *pool* and it is carried out by a Trained Technician, depicted as a *lane*. First of all, the technician should Allow the patient to sit at rest for at least 10 minutes before the beginning of the test. This temporal span is represented by an *intermediate timer event*. A couple of *parallel gateways* are

employed to denote that, during this time frame, the technician must **Check patient clothing** and **Evaluate contraindications**. The evaluation of contraindications is a decision about the conduct of the test, based on the current values of the patient vital parameters.

The output of the decision is used by the following *exclusive gateway Perform test?*. If the test is believed to put the patient at risk, then the process is ended without further action, as represented by the simple *end event End test*. Otherwise, the 6MWT is performed. The latter is represented by means of a *collapsed subprocess*, as individual steps are not detailed. However, if the patient experiences dyspnea, chest pain, leg cramps, staggering, diaphoresis or pale appearance, the 6MWT must be immediately interrupted [98]. In Fig. 2.1, this is set of conditions prompting interruption is captured by *boundary interrupting conditional event Stop test*, which encodes the aforementioned conditions and generates an exception flow that leads to task **Assist patient**.

All the information collected during the execution of the 6MWT is recorded in the patient's medical record MR, represented as a *data store*.

Following the notions provided in the BPMN standard [11], in this thesis we refer to the definition of process model provided below.

Definition 2.1 (Process Model). A process model $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$ is a tuple consisting of a finite non-empty set of control flow nodes N , a finite set of data nodes DN , a finite non-empty set C of control flow edges, a finite set of text annotations TA , a finite set of data associations F , a finite set of undirected associations T , and a finite set of resources R . The set $N = \{A \cup G \cup E\}$ of control flow nodes consists of the disjoint sets A of activities, G of gateways and $E = \{E_{start} \cup E_{int} \cup E_{end}\}$ of events. $E_{start} \subseteq E$ is the set of start events. $E_{end} \subseteq E$ is the set of end events. $E_{int} \subseteq E$ is the set of intermediate events, which includes also the set of boundary events E_B , i.e., $E_{int} \supseteq E_B$. Control flow $C \subseteq N \times N$ defines a partial ordering between the elements of N . $DN = DO \cup DS$ is the set of data nodes, consisting of the disjoint sets DO of data objects and DS of data stores. $F \subseteq (DN \times A) \cup (A \times DN)$ is the set of data associations that connect data nodes with activities. $T \subseteq (TA \times A)$ is the set of associations that connect text annotations to activities. α_k , and α_t are functions that associate a type to the elements of A . $\alpha_k : A \rightarrow \{task, subprocess\}$ distinguishes activities into tasks and subprocesses. Let us call $A' = \{a \mid a \in A \text{ and } \alpha_k(a) \mapsto task\}$ the set of tasks in m . Function $\alpha_t : A' \rightarrow \{abstract, user, business\ rule, service, script\}$ associates to each task a specific type. Functions γ_r and γ_{ty} associate a type to the elements of G . $\gamma_r : G \rightarrow \{split, merge\}$ assigns a gating mechanism to each gateway of G . $\gamma_{ty} : G \rightarrow \{parallel, exclusive, event-based\}$ assigns a routing type to each gateway of G . Functions ϵ_{tr} , ϵ_{ty} , and ϵ_k associate a type to the elements of E . $\epsilon_{tr} : E \rightarrow \{throwing, catching\}$ distinguishes events in those that throw a trigger and those that catch a result. It always holds that for each $s \in E_{start}$ $\epsilon_{tr}(s) = catching$ and for each $e \in E_{end}$ $\epsilon_{tr}(e) = throwing$. Function

$\epsilon_{ty}: E \rightarrow \{none, message, signal, error, timer, conditional, escalation, cancel, multiple, parallel\ multiple, terminate\}$ associates a type to each event of E . $\epsilon_k: E_B \rightarrow \{interrupting, non-interrupting\}$ associates to each boundary event its interrupting behavior. $\beta: A \rightarrow 2^{E_B}$ is a function that associates to each activity $a \in A$ a set of boundary events attached to its border. Function $\rho: A \rightarrow R$ assigns to each activity the resource responsible for its execution. Finally, $\mathcal{L}: N \rightarrow \sigma$ is a function that assigns a label σ , represented as a string, to each flow node in N .

Given a process node $n \in N$, in this thesis we denote as $\bullet n \subseteq N$ and as $n \bullet \subseteq N$ the sets of predecessor and successor nodes of n , respectively.

Further, we consider dealing with process models satisfying the following structural criteria. Given a process model m :

- (i) each activity in m has exactly one preceding and exactly one succeeding control flow node, i. e., $\forall a \in A, |\bullet a| = 1$ and $|a \bullet| = 1$;
- (ii) each split gateway in m has exactly one preceding flow node and at least two succeeding flow nodes, i.e., $\forall g \in G$ such that $\gamma_r(g) = split, |\bullet g| = 1$ and $|g \bullet| \geq 2$;
- (iii) each merge gateway in m has two or more preceding flow nodes and exactly one succeeding flow node, i.e., $\forall g \in G$ such that $\gamma_r(g) = merge, |\bullet g| \geq 2$ and $|g \bullet| = 1$;
- (iv) it has a unique start event $e \in E_{start} \subseteq E$, i.e., $|E_{start}| = 1$;
- (v) it has multiple end events, i.e., $E_{end} = e_1 \dots e_n \in E_{end} \subseteq E$ if and only if the set E_B of boundary events is not empty (i.e., $E_B \neq \emptyset \iff |E_{end}| \geq 1$). In this case, $e_1 \in E_{end}$ is the end event of the main process flow, while every other $e_i \in E_{end}$ ends the exception flow outgoing from $eb_i \in E_B$. Otherwise m has a unique end event $e \in E_{end} \subseteq E$, i.e., $|E_{end}| = 1$.

According to Def. 2.1 process models are static directed graphs with typed nodes. Def. 2.1 includes several modeling elements covering the whole descriptive conformance of BPMN [11], thus spanning different process perspectives.

Without loss of generality, in the following chapters we may refine this definition to focus on specific aspects (e.g., data modeling) and assume certain sets of modeling elements to be empty.

2.1.2 BPMN Semantics and Petri Nets

Graphical notations like BPMN are intuitive enough to be well understood almost at first sight.

However, in BPMN the execution semantics of process elements is described informally, as it is common to base on further formalization efforts, such as those presented in [73, 100, 101]. Indeed, the standard lacks of the precise mathematical basis that is required to render the semantics of certain modeling elements unambiguous [101] and a subset of constructs, called “non-operational”, are meant to be used only for conceptual modeling [11].

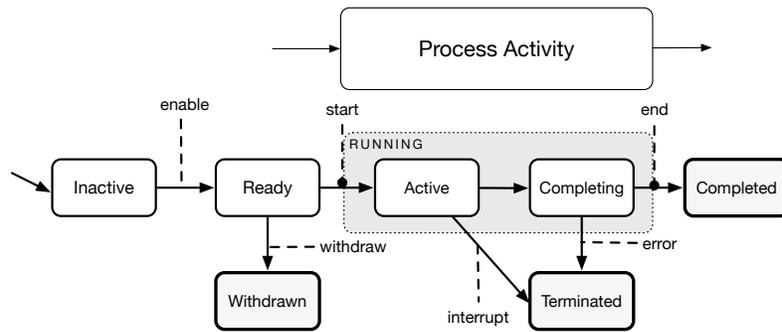


Fig. 2.2: State diagram describing the life-cycle of a BPMN activity, adapted from [1, 11, 33].

The goal of this section is to introduce the main aspects of the operational semantics of BPMN needed to understand the work presented in this thesis. Elucidations about the semantics of specific modeling elements, such as different types of intermediate events, will be recalled later in each chapter, when needed.

In BPMN, a process is instantiated when one of its start events occurs. Upon occurrence, each start event creates a token that begins flowing through the following process elements, activating them. As mentioned previously, in this thesis we consider processes having a single start event. For a process instance to be completed, all tokens in that instance must reach an end event.

Activities are associated to a life-cycle that determines their execution semantics. In this thesis, we refer to the life-cycle depicted in Figure 2.2, adapted from [1, 11, 33]. When a process is instantiated, all the activities are in state *Inactive*. An activity is *enabled* and becomes *Ready* for execution when the number of tokens is available to activate it. When data and allocated resources are available, the activity changes from *Ready* to *Active*, through transition *start*. The start of the activity actually corresponds to an event that determines its beginning [11, 33]. When the activity ends without anomalies, it enters state *Completing*, which captures processing steps prior to activity completion, such as the end of process flows originated from non-interrupting events attached to its border. Then, the activity switches to state *Completed* through transition *end*. An activity moves from state *Ready* to *Withdrawn*, whenever it is placed on a process branch in the configuration of an event-based gateway that is not chosen during execution. Finally, any active or completing activity can switch to state *Terminated* in case of an execution error.

Gateways, are used to split or merge the sequence flow. Parallel gateways are used to spawn concurrent threads on parallel process branches and to synchronize multiple concurrent incoming branches. A parallel gateway is activated if there is at least one token on each incoming sequence flow. The parallel gateway consumes exactly one token from each incoming sequence flow and produces exactly one token at each outgoing sequence flow. Exclusive gateways are acti-

vated any time a token arrives. The token is routed to exactly one of the outgoing sequence flows, i.e., the one associated with the data-based condition that evaluates to true. Henceforth, no more conditions are evaluated. Finally, event-based gateways are activated depending on which is the first event in its configuration to be triggered. The choice of which process branch to take is deferred until one of the triggered completes. The first one to complete causes all other branches to be withdrawn.

The semantics of other modeling elements or complex event handling mechanisms, will be recalled later, when needed.

Petri nets [71] have been introduced as a means to formally model concurrent systems in the same way that finite automata are used as a mathematical model for sequential systems [102]. In BPM, Petri nets are often used as a formal back-end for different notations, as they can be suitably applied to support process semantics specification, structural and behavioral property verification, and static analysis [78].

Petri nets token-based execution semantics makes them suitable for reproducing BPMN process execution, as its expressiveness is sufficient to reproduce the behavior of all the basic routing constructs of a BPMN process [103]. Besides, Petri nets are useful to provide an unambiguous graphical representation of business processes, suitable to prevent uncertainties and contradictions, which can easily arise when using informal diagramming techniques such as BPMN [104].

We base on the mappings of the core modeling elements of BPMN onto Petri nets proposed in [73, 79], which are considered in Chapter 3.

A Petri net is a particular kind of directed, bipartite graph, formally defined as follows [103]:

Definition 2.2 (Petri Net). A Petri net is a tuple $N = (P, T, A, M_0)$ where:

- P is the finite set of places, $P \neq \emptyset$;
- T is the finite set of transitions, $T \neq \emptyset$;
- $A \subseteq (P \times T) \cup (T \times P)$ is the set of directed arcs from places to transitions and from transitions to places;
- M_0 is the initial state (or marking) of the net, defined as a mapping $M_0 : P \rightarrow \mathbb{N}^+$ that assigns each place a nonnegative integer.

Compared to the definition of Petri net provided in [103], we omitted the weight function on the arcs, as we consider arcs to have a weight equal to 1.

Graphically, places are depicted as circles, whereas transitions are represented as rectangles. Moreover, it holds that $P \cap T = \emptyset$.

A place $p \in P$ is called an *input place* for a transition $t \in T$ if and only if there exists a directed arc from p to t . Similarly, a place $p \in P$ is called an *output place* for a transition $t \in T$ if and only if there exists a directed arc from t to p .

Petri nets are traversed by tokens, drawn as black dots, that are collected by places and enable transitions. Transitions are the active components of a Petri net and they fire according to a defined *firing rule*. Specifically, a transition can fire when at least one token is available in each of its input places (i.e., the

transition is *enabled*). When a transition fires, it removes one token from each of its input places and it adds one token to each of its output places.

The state M of a Petri net, also called *marking*, is the distribution of tokens over places (i.e., $M \in P \rightarrow \mathbb{N}$). In this chapter, we use the multiset notation to represent states, that is, $M = p_1(n) \dots p_i(n)$ denotes that place p_i contains n tokens. At any time, each place can have zero or more tokens and the number of tokens may change during the execution of the net. Given a Petri net N , a state M_n is said to be *reachable* from a state M_0 if there exists a firing sequence σ of transitions $t_0 \dots t_n$, such that $M_0 \xrightarrow{\sigma} M_n$.

Petri nets have certain interesting properties that allow one to verify that the designed net behaves as expected. Below, we introduce the concepts of *liveness* and *boundedness* [77], that correspond to the dynamic behavior of a Petri net in a give state M_i .

Definition 2.3 (Liveness). A Petri net $N = (P, T, A, M_0)$ is live if and only if, for every reachable state M_1 and every transition $t \in T$ there is a state M_2 reachable from M_1 which enables t .

Definition 2.4 (Boundedness). A Petri net $N = (P, T, A, M_0)$ is bounded if and only if for every state reachable from M_0 the number of tokens in each place p does not exceed a finite number k .

Whereas liveness ensures the complete absence of deadlocks, regardless of the chosen firing sequence, boundedness guarantees that there are no places in the net where tokens accumulate.

In general, structural constraints can be applied to Petri nets in order to better suit domain-specific goals and to promote the application of existing verification techniques. In order to model business processes or workflow procedures, Petri nets must have some peculiar properties [77]. First of all, a Petri net must have a distinct source place (i.e., a single place that is not the target of any arc) and a distinct sink place (i.e., a single place that is not the source of any arc). Besides, all of its nodes must lie on some path from the source place to the sink place [100].

These structural restrictions identify an interesting sub-class of Petri nets, called workflow nets.

Definition 2.5 (Workflow Net). A workflow net is a tuple $WN = (P, T, A, M_0, e, c)$ where:

- (P, T, A, M_0) is a Petri net;
- $e \in P$ is a distinguished place (called source or initial place) that has no incoming edges, i.e., $\bullet i = \emptyset$;
- $c \in P$ is a distinguished place (called sink or final place) that has no incoming edges, i.e., $e \bullet = \emptyset$;
- Every place and every transition is located on a path from e to c .

Intuitively, a workflow net captures the execution of one instance of a business process, from its creation up to its completion. Process models conforming to the

definition of workflow net are called *structurally sound*, as all their activities are embedded in the context of the process. Structural soundness is the simplest criterion used to assess process model correctness.

However, to verify properties related to the dynamic behavior of process models, such as the absence of deadlocks or livelocks, the property of *soundness* has been defined on top of workflow nets as follows.

Definition 2.6 (Soundness). A workflow net $WN = (P, T, A, M_0, e, c)$ is sound if and only if:

- (Safeness) each place cannot hold multiple tokens at the same time. Formally, $\forall p \in P, M(p) \leq 1$.
- (Option to complete) it is always possible to reach the state that marks the sink place c starting from the source place e and state M_0 . Formally, $\forall M (e \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} c)$;
- (Proper completion) state c is the only state reachable from state e with at least one token in place c . All other places must be empty. Formally, $\forall M (e \xrightarrow{*} M \wedge M \geq c) \Rightarrow (M = c)$;
- (Absence of dead parts) for every transition t in the net, there is a sequence enabling it. Formally, $\forall t \in T \exists M, M' e \xrightarrow{*} M \xrightarrow{t} M'$.

When adapted to process models, soundness guarantees that tasks can participate in a process instance; each process instance eventually terminates, and when it terminates there is exactly one token in the final place [1]. The soundness of a workflow net can and should be checked at design time, but it is not easily seen at the model level [105]. Moreover, to decide whether a given workflow net is sound, reachability graphs should be created and checked, facing exponential run time behavior.

In this regard, in the following section we introduce an important structural property of business process models that may assist non-expert modelers in guaranteeing process models soundness.

2.1.3 Principles of Well-Structured Process Design

In general, being a graph-oriented process definition language, BPMN allows combining flow objects almost arbitrarily. However, this often leads to the definition of hardly-readable and complex process models, often containing semantic errors difficult to be detected during early process development phases [106].

In this setting, structural restrictions are desirable for increasing process model readability and for preventing the onset of undesired deadlocks at runtime [73].

Processes may be “well-structured” to enhance modularity and improve local reasoning on large and complex process models. Below, we report the informal definition of well-structuredness for a BPMN process model, adapted from [107].

Definition 2.7 (Well-structured BPMN process model). A BPMN process is said to be *well-structured* if for every node with multiple outgoing edges,

i.e., a split gateway, it has a corresponding node with multiple incoming edges, i.e., a join gateway of the same kind, such that the set of nodes delimited by the split and the join nodes form a Single-Entry-Single-Exit (SESE) region, and these regions within the process are properly nested.

The advantage of this property is that it can be checked easily on the structural level of the model.

Moreover, a well-structured process is guaranteed to be sound if it is live [105]. Liveness can be checked in polynomial time in the size of the net for a special class of workflow nets, called *free choice* nets [102]. In free choice nets the result of the choice between two transitions can never be influenced by the rest of the system – in other words, choices are free.

Definition 2.8 (Free-choice workflow net). A workflow net $WN = (P, T, A, M_0, e, c)$ is free-choice if $(p, t) \in A$ implies $\bullet t \times p \bullet \in A$ for every place p and every transition t .

Free choice nets have the property that the sets of input places of two transitions are either disjoint or identical [1]. That is, if there is an arc from a place p to a transition t , then either t is the only output transition of p or p is the only input place of t . In this way, whenever an output transition of p is enabled, all its output transitions are enabled, and therefore the choices in which t takes place are free [102]. For a free-choice workflow -net it is possible to decide soundness in polynomial time [77].

Free choice nets may be bidirectionally mapped to BPMN process models that are solely composed of tasks and exclusive or parallel gateways.

Another way to check soundness (cf. Def. 2.6) of well-structured processes is based on SESE fragments decomposition [108]. Indeed, soundness is compositional with respect to SESE fragments, i.e., each fragment can be checked in isolation.

For the reasons discussed, in this thesis we prefer to define process models that are both well-structured and structurally sound, when possible. In this regard, we acknowledge that some business processes can hardly be matched by a well-structured process description, especially since modeling in a well-structured manner requires having an overview of the whole process.

As an example, the presence of exception flows, especially those originating from non-interrupting events, compromises both well-structuredness and structural soundness, since BPMN recommends ending each exception flow originating from a boundary event with its own end event [11]. Thus, the exception flows in the process models presented in this thesis do not satisfy these properties.

However, we exploit BPMN process structural properties as a mean for guaranteeing that the designed process models are flexible and modular and that they can be easily extended by adding new structured process branches, regions or subprocesses.

2.2 Decision Modeling

The interplay between process and decision models plays a crucial role in BPM.

Process gateways and conditional events often encode data-based operational decisions, whereas decision activities encompass more complex non-operative decision-making. Decisions affect the process flow and, consequently, they impact both the performance of the process and the quality of its (business) outcomes.

However, important decisions often remain hidden within process diagrams and BPMN is misused for decision modeling [20, 109]. For these reasons, business process modeling must be properly complemented with decision design.

2.2.1 Decision Model and Notation

The OMG recently developed the Decision Model and Notation (DMN) standard, currently at version 1.1 [47]. DMN is a standard notation that may be used to bridge the gap between the high level business logic, represented in processes, and decision implementation, typically specified in terms of decision logic.

The notation supports the design of both human and automated decision-making, at different levels of abstraction.

Decision models consist of two logical layers, one dealing with decision requirements, the other one with decision logic. Together, these two layers provide a complete decision model that may be used to complement a process model by detailing the decision-making carried out in process tasks [47] or subprocesses [110].

The decision requirements level is modeled by means of a Decision Requirement Graph (**DRG**), which may be linked to one or more tasks belonging to one or multiple business processes. A DRG represents the main elements involved in a decision-making domain and the dependencies between them. As DRGs may be large and complex, they can split into one or more Decision Requirement Diagrams (**DRDs**). DRDs may be used to show partial views of the whole decision domain, focusing on areas of interest for a specific user or goal. These diagrams define which decisions are made in process tasks, which are their interrelationships, and their requirements for decision logic.

The main elements forming a DRD are introduced in the following list.

- *Decisions* portray the act of determining an output from a number of inputs, using logic defining how the output is retrieved from the inputs. Decisions may reference one or more knowledge models. Graphically, a decision is depicted as a rectangle, labeled with the decision name.
- *Business knowledge models* denote functions that encapsulate business knowledge, such as business rules, decision tables, or analytic models. They are illustrated as rectangles with clipped corners.
- *Input data* represent information used as an input by one or more decisions. When enclosed within a knowledge model, input data represent the parameters to the knowledge model. Graphically, they are depicted as a shape with two parallel straight sides and two semi-circular ends.

- *Knowledge sources* identify an authority for a business knowledge model or a decision, such as a person responsible for managing the represented knowledge, or a published document enclosing it. They are depicted as a shape having three straight sides and a wavy one.

Three kinds of dependencies are defined between the presented elements, in the form of requirements. An *information requirement*, depicted as a simple arrow, represents any input data (or decision output) that is used as input to a decision. Graphically, an information requirement is depicted as a solid-line arrow with a filled arrowhead. A *knowledge requirement* denotes the invocation of a business knowledge model by a decision or by another knowledge model. Graphically, it is a dashed-line arrow, having an open arrowhead. Finally, an *authority requirement* depicts the dependence of a DRD element on another DRD element that acts as a source of guidance or knowledge. This may denote the fact that a business knowledge model is consistent with a published document, or that a specific person is responsible for a certain decision. Graphically, it is represented by a dashed line with a filled circular head.

The formalization of decision requirement diagram used in this thesis is presented below.

Definition 2.9 (Decision Requirement Diagram). A *Decision Requirement Diagram (DRD)* is a tuple (D, B, I, K, IR, KR, AR) consisting of:

- a finite non-empty set of *Decision* nodes D ;
- a finite set of *Business knowledge* nodes B ;
- a finite non-empty set of *Input data* nodes I ;
- a finite set of *Knowledge source* nodes K ;
- a finite non-empty set of directed edges IR representing *Information requirements* such that $IR \subseteq (I \cup D) \times D$;
- a finite set of directed edges KR representing *Knowledge requirements* such that $KR \subseteq B \times (D \cup B)$;
- a finite set of directed edges AR representing *Authority requirements* such that $AR \subseteq (D \cup I \cup K) \times (D \cup B \cup K)$.

Herewith, $(D \cup B \cup I \cup K, IR \cup KR \cup AR)$ is a directed acyclic graph.

Decision logic completes the (executable) description of decisions, by allowing one to define them in sufficient detail to support validation and/or automation. In general, decision logic can be modeled in several ways, which translate business expertise into business rules, analytic models, or other formalisms.

DMN defines different kinds of value expressions to associate decision logic with elements of a DRG. A *literal expression* represents decision logic as text that describes how an output value is derived from a set of inputs. Literal expressions may be represented with a formal, and possibly executable language, or they can be written in plain English. Formal expressions can be specified with the Friendly Enough Expression Language (**FEEL**) [47], which translates if/then/else data structures and calculations into executable expressions.

A *decision table* is a tabular representation of a set of input and output values, organized into rules that describe how a given input relates to one or

more corresponding outputs [47]. The Simplified FEEL (S-FEEL) is a subset of FEEL used for defining expressions in decision tables.

An *invocation* is a tabular representation of how decision logic, represented by a business knowledge model is invoked by a decision, or by another business knowledge model. Tabular representations of decision logic are called *boxed expressions*.

As an example, let us recall the 6-Minute Walk Test, introduced in Sect. 2.1.1. In some clinical situations, this test may be contraindicated for patients considered at risk. Contraindications are evaluated by a trained technician right before test execution. These are a resting heart rate of more than 120 bpm, a systolic blood pressure of more than 180 mm Hg, and a diastolic blood pressure of more than 100 mm Hg [98]. The technician performing the test must measure the patient vital parameters and decide whether the test can be conducted safely.

A simple DRD representing the decision about the conduct of the 6MWT is shown in Fig. 2.3. The main decision regarding the **Conduct of 6MWT** is based on the assessment of **Contraindications**, which are encoded as rules within a *business knowledge model*. This model is invoked by the main decision, which passes *input data* **Arterial Blood Pressure** and **Heart rate** as parameters for the model. Finally, the **Trained technician** is indicated as a *knowledge source*, to represent the authority responsible for deciding about the conduct of 6MWT.

The described DRD of Fig. 2.3 may be linked to task **Evaluate contraindications** of the BPMN process of Fig. 2.1. In general, there may be a one-to-one correspondence between a BPMN process task and a DMN decision. Yet, this is not mandatory, as the same decision can refer to multiple tasks and can be used across processes.

The DMN standard recommends that decisions are made either manually as in *user* tasks, or in a semi-automated manner, as in *business rule* tasks [47]. In BPMN, a *business rule* task provides a mechanism for the process to provide input to a business rule engine and to get the output of calculations that the business rule engine could provide. A *user* task is executed by a human performer. However, it is worth noticing that not all user and business rule tasks

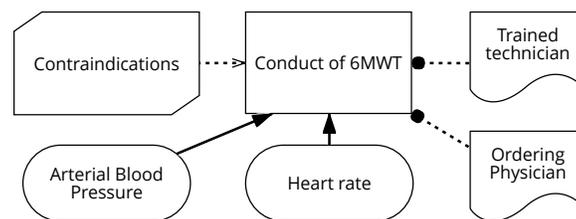


Fig. 2.3: Decision Requirement Diagram representing the decision for the conduct of the 6-Minute-Walking-Test, with related input data and invoked business knowledge model **Contraindications**.

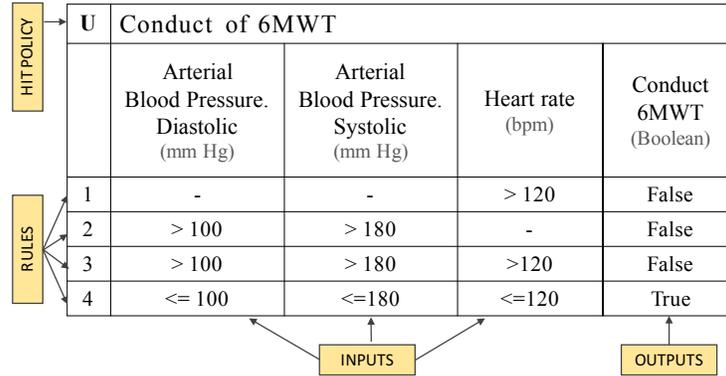


Fig. 2.4: Labelled decision table portraying the decision logic underlying decision Conduct of 6MWT of Fig. 2.3.

represent a decision activity. Indeed, the task type may help decision analysts in the identification of decision-making tasks, but usually approval of stakeholders is needed to properly identify decisions [20].

Fig. 2.4 depicts the decision logic corresponding to business knowledge model *Contraindications*, by using a decision table. Input and output values are arranged into columns, while rules correspond to rows. At the top left corner of a DMN decision table, a couple of attributes can be present, one to denote the kind of hit policy for the table, the other one to indicate its completeness. A hit policy specifies how overlapping rules are handled, that is, how output values must be returned in case more than one rule match input data. In the presented case, character **U** is used to denote a *unique* hit policy for the table, meaning that no overlapping rule is allowed, that is, all rules are disjoint and only a single rule can be matched [47]. A completeness indicator can be optionally used to specify whether at least one rule in the table matches a possible input configuration. Symbol **C** denotes a complete table (which is the default for DMN), whereas **I** is used for incomplete ones. Finally, symbol “-” is used to denote that the value of a certain input is *irrelevant* for the final decision, that is, the decision can be made even if that input value is unknown.

In this thesis, specifically in Chapters 11 and 12, we will mostly focus on Decision Requirements Diagrams, as they are meant to form a bridge between business process models and decision logic [47] and, thus, they are suitable to meet same the level of abstraction of (conceptual) BPMN process models.

2.3 Process Modeling in Healthcare

Healthcare organizations rely on a wide range of processes with different characteristics and requirements that are managed and executed on a daily basis [76].

Patient management combines organizational and administrative tasks with clinical procedures that coexist and are both interlinked and interleaved [51]. Healthcare is thus widely recognized as one of the most promising, yet challenging, domains for the adoption of process-oriented models and solutions that are able to support both organizational and clinical processes [13, 26, 51, 76].

In general, process models can be successfully employed to model processes belonging to disparate organizational domains. Among the various settings suitable to be represented through business process formalisms, the clinical domain represents a challenging and important application scenario, mostly due to its intrinsic organizational complexity [26]. In this thesis, we take the view that being able to propose general solutions that work with complex clinical process, is promising for successfully applying them to other sophisticated scenarios [13].

A critical aspect of healthcare domains is represented by the fact that medical knowledge is fragmented, hard to maintain, and often based on patient-specific information and on clinicians' expertise and experience [75]. Different actors are loaded with multiple and heterogeneous tasks and responsibilities, and often have complementary expertise. Furthermore, when considering the application of multidisciplinary care plans and clinical guidelines, there is a need for constant coordination in terms of temporal and resource scheduling.

Thus, clinical and healthcare working environments pose several challenges to traditional process modeling approaches [26]. Among them, we recall:

- (i) the need of integrating different process modeling perspectives, most importantly the control-flow and the data perspectives must be blended with decision-making aspects;
- (ii) the need of ensuring that the processes are reliable and consistent, and that can be efficiently verified to avoid unexpected run-time behavior;
- (iii) the need of comply with temporal constraints;
- (iv) the need of supporting deviations from standard/structured procedures and of handling exceptions that may arise during execution.

In general, healthcare processes can be classified based on the degree of structuring and predictability they exhibit [76]. *Structured processes* are suitable to represent highly repeatable and predictable routine work. Despite including predefined exception flows, they tend to be quite inflexible, but can be easily standardized and automated. On the contrary, *unstructured processes* are characterized by a low level of predictability, but meet high flexibility requirements. These dynamic processes, are often referred to as knowledge-intensive, i.e., "processes whose conduct and execution are heavily dependent on knowledge workers performing various interconnected knowledge intensive decision-making tasks" [111].

Knowledge-intensive processes are especially useful to model diagnostic and therapeutic steps driven by clinical decision-making. Instead, for administrative and organizational tasks (e.g., as patient discharge, transfer, or routine testing) or when there is the need to standardize clinical procedures, activity-centric traditional process modeling approaches may be used.

An important step towards the standardization of clinical and healthcare procedures is constituted by the development of evidence-based clinical practice

guidelines and care pathways, which introduce a process-oriented perspective in the management of patients [76].

A clinical practice guideline consists of a set of systematically developed statements to assist practitioners decisions, in order to provide optimal care with respect to a specific clinical condition [112]. The main goal of clinical guidelines is to improve the quality of care by limiting unjustified practice variations and by reducing costs associated to patient management.

Application scenarios of clinical practice guidelines include facilitation of the communication between various medical personnel and patients, standardization of medical treatment processes, training and education of young medical professionals, design and implementation of health information systems, automated analysis for the purpose of process optimization [113].

In order to be effective, clinical practice guidelines must be integrated with the healthcare process technology and decision support systems [114]. To this end, clinical practice guidelines need to be formalized as computer-interpretable guidelines (CIGs), that is, the clinical knowledge contained in clinical guidelines must be represented in a computer-interpretable way.

Several representation languages for CIGs have been developed over the years. Among them, we find document-centric models, decision trees and probabilistic models, and Task-Network Models, i.e., hierarchical models of the guideline control flow represented as a network of specific tasks that unfold over time. Being strongly activity-centric and relying on flow-chart-like structures, the latter ones are particularly interesting from a BPM standpoint. In Chapter 13, we recall relevant approaches concerning the modeling and formalization of CIGs.

Care pathways are adaptations of clinical guidelines tailored to respond to the healthcare and economic requirements established for a local clinical context.

Care pathways have been defined in slightly different ways. In this thesis, we refer to the following two widely acknowledged definitions.

According to [115] a care pathway is “a complex intervention for the mutual decision-making and organization of care processes for a well-defined group of patients during a well-defined period”. Instead, in [68] integrated care pathways are defined as “structured multidisciplinary care plans which detail essential steps in the care of patients with a specific clinical problem.”

Both definitions embrace the notion of care processes/plans and emphasize salient features of care pathways, such as the multidisciplinary, the inherent decision-making aspects, and the focus on specific groups of patients.

Care pathways have an intrinsic dual essence that reflects the above-mentioned duality of structured and unstructured process models [76]. Whereas, the sequence of actions to be executed by care-providers and its temporal evolution are inherently process-oriented, clinical evaluation and medical reasoning tasks, such as diagnosis and treatment, constitute reliable examples of decision-making [58].

One of the main challenges in creating clinical practice guidelines and care pathways is the selection of the modeling method [75, 76, 113]. However, clinical domains are not novel application scenarios for BPM-based approaches and could help in this direction [13, 26, 51, 116, 117].

From a BPM standpoint, a care pathway encapsulates the workflow that delineates how to conduct a specific healthcare procedure for addressing specific medical complication (see for example the BPMN process of Fig. 2.1).

Such a heterogeneous scenario requires the standardization of patient management and care delivery procedures, and of their decisional counterpart, to guarantee proper information and knowledge sharing [118]. Reaching a standard care plan definition is the first step towards ensuring that all the patients presenting the same clinical conditions benefit from the same care. Thus, care plans must respond to disparate and individual needs, and must comply with organizational and social objectives.

In the described setting, business process modeling and re-engineering techniques may be used to support the iterative and incremental design of care pathways and of the related decision-making activities [59].

When considering different process modeling perspectives (cf. Sect. 2.1), clinical and healthcare processes require standard yet flexible solutions to deal with informational, temporal, and decision aspects.

Under an informational perspective, it is crucial to provide clinicians with a data-aware overview of the care process, in order to improve understanding and to support clinical decision-making. To this end, process models must be integrated with existing health databases, starting from the conceptual level.

Under a temporal perspective, constant coordination in terms of temporal and resource scheduling is vital to achieve acceptable levels of care provision quality. Moreover, clinical decisions are affected by temporal constraints, and so is their effectiveness [112]. Physicians have to decide which are the most appropriate interventions to schedule, which is their preferred order, and which are the activity to carry out in case of unexpected changes in treatment outcomes [26]. Such delicate, but crucial decisions, typically based on evidence-based knowledge, clinical best-practice or examination results, affect the outcomes of the overall care process and their reliability often depends on temporal properties.

For the reasons discussed in this section, the concepts and solutions devised in this thesis are often applied to real-world clinical and healthcare processes, without loss of generality.

The Temporal Perspective

Formal specification and operational support of time constraints constitute fundamental challenges for any process-aware information system [23].

One key perspective when dealing with business process management is time. Nonetheless, temporal constraints are often neglected during process modeling and time support is limited in existing process management systems [17].

Different methods for the specification of temporal constraints and time-aware verification techniques and tools have been developed in the fields of workflow, BPM, web service composition, and inter-organizational research [18]. Most state-of-the-art approaches in the field of activity-centric process modeling [32, 35, 34, 119], propose to extend existing process modeling languages (most notably BPMN) to support the representation of temporal constraints. Then, the enriched process modeling language is typically mapped to a formal model, such as timed automata [66], for model checking purposes.

Part II introduces three distinct approaches to model and verify the temporal perspective of business processes. In line with existing literature, we consider the BPMN standard for the specification of temporal constraints and rely on formal models for verification purposes. However, we show how to design sound BPMN models able to capture certain temporal constraints, and focus on verifying either the soundness of the obtained process models or specific temporal properties.

In Chapter 3, we address the modeling and specification of constraints related to the duration of process activities in a way that is conceptually and semantically BPMN-compliant. Starting from the BPMN standard and considering the advanced event-handling mechanisms it includes, we propose an approach to define reusable duration-aware process models for representing different nuances of activity duration, at design time. Violations of the specified constraints are prompted at run-time through signal events. For ensuring that the proposed process models behave as expected, we map them to time Petri nets (TPN) [120] and use existing TPN (simulation) tools for soundness verification [121, 122]. However, in Chapter 3, we abstain from checking temporal constraints at design-time.

In Chapter 4, we deal with the modeling of selected temporal constraints that determine which process execution paths are taken based on temporal conditions (e.g., time-difference between certain events, temporal mutual exclusion, time lags). In particular, we focus on enforcing them in BPMN process models and formalize their semantics through timed automata [66], opening the path open for their formal verification through established model checking tools [123].

Temporal aspects also constrain how decisions are made in processes: while some constraints hold only along certain paths, decision outcomes may be restricted to satisfy temporal constraints. In Chapter 5, we introduce a new kind

of time-aware BPMN processes able to discern decision activities based on how their outcomes are managed at run-time. Then, we propose a mapping from time-aware BPMN processes onto a specific kind of temporal-constraint networks, called Conditional Simple Temporal Network with Uncertainty and Decision (CSTNUD) [67], to provide the semantics of the BPMN elements extended with the temporal dimension and to verify their dynamic controllability [29].

Although focusing on the importance of modeling time in BPMN processes, the approaches presented in Chapters 3–5 differ from each other based on their final goal, influencing which methods are used.

In detail, Chapter 3 focuses on representing duration constraints in BPMN, aiming to embed the temporal dimension in modular BPMN processes that can be run directly on BPMN engines. Time Petri nets are used for verifying the soundness of the obtained process models: we do not consider checking duration constraints at design time as (i) actual durations are known during execution; (ii) violations of duration constraints during execution are captured by event and exception handling mechanisms.

Chapter 4 begins with an approach similar to the one presented in Chapter 3 for the modeling of temporal constraints in BPMN. However, the proposed BPMN solutions *enforce* the temporal constraints and, thus, we do not consider constraint violations management. In addition, Chapter 4 proposes a mapping from BPMN towards timed automata that can be used for checking the temporal constraints.

Chapter 5 considers the tight interaction between temporal constraints and the decision-making enacted during process execution to ensure that such constraints are observed. The main focus of this chapter is not only the specification of temporal constraints, but also the checking of the process dynamic controllability, that is, the capability of executing the process for any possible duration of tasks. In this chapter, BPMN is extended to capture temporal constraints: the semantics of process elements extended with a temporal dimension and with decision-making aspects is provided by means of CSTNUDs. The latter are used in favour of TPN and TA to exploit existing results about dynamic controllability checking in workflows [29, 69, 67].

Finally, in Chapter 6 we discuss related work and compare the introduced contributions with state-of-the-art approaches.

A Modular Approach for Defining Duration Constraints on Business Processes

This chapter is based on results published in [52, 53].

Business process management focuses on the modeling and management of business processes by using suitable techniques that allow organizations to be more efficient and flexible in achieving their goals. In this context, a deep understanding of organizational processes supported by an intuitive design approach can improve the quality of the business from different viewpoints, such as costs reduction, resource planning, and increase in competitiveness.

A very important aspect to consider when dealing with business processes is time [16, 17, 27, 119, 124]. Temporal coordination is naturally represented and managed during process design. Indeed, a business process is as a collection of activities related across time and space, that realizes a specific service or business goal [1]. Temporal constraints play a crucial role in process execution, as most real-world processes run under time constraints [27]. Activity performance takes time, the scheduling of resources and workforce requires temporal coordination, and process compliance with deadlines is fundamental in most application environments [23, 125, 126].

In this chapter, we discuss issues related to the modeling and management of temporal duration of activities and specific process regions.

Activities are used to represent any amount of work carried out within a process, and thus, they implicitly span over a certain, finite amount of time. Explicitly representing activity duration in process models is useful whenever temporal information is crucial for understanding and re-engineering the designed procedures. Furthermore, in many real-world applications, constraining activity duration becomes essential in order to guarantee the completion of the overall process within desired time limits, which are driven by resource availability and scheduling requirements [127]. Similarly, under a process compliance perspective, activities need to adhere to regulations and policies that set deadlines or limit duration based on best-practices. Actually, it is quite common that both the execution and results of an activity are affected by its temporal duration: some

tasks and procedures become irrelevant with respect to process goals if they are not performed within predefined time limits [52].

As an example, let us consider dialysis, whose efficacy is determined by the overall duration of the treatment.

Example 3.1 (Dialysis). Dialysis is used to purify blood from waste and extra water and a single treatment session usually lasts 3-5 hours. Despite short and rapid dialysis is less painful for the patient, rapid fluid-removal attempts can result in depletion of the circulating volume and in hypotension that, together with incomplete removal of salt and water, are associated with increased risk of death [128]. Therefore, treatments briefer than 3 hours are considered ineffective and dangerous for the patient.

Example 3.1 shows that, in order to properly model activities “dialysis”, the representation of defined duration constraints is an important issue to consider. Indeed, in some cases executing an activity without observing its duration constraints undermines the validity of its outcomes.

The contribution of this chapter lies on the specification and management of duration constraints using the well-known Business Process Model and Notation [11]. In general, due to the lack of direct support for time aspects, modeling and managing temporal constraints in BPMN is quite demanding for process designers [23]. To overcome this limitation and foster the verification of time-aware processes, a considerable number of research proposals have focused on extending the BPMN standard [32, 34, 35, 36, 119]. Compared to these approaches, we focus on solely using existing BPMN constructs to represent different nuances of duration constraints, even if this requires combining multiple process elements to express a single temporal aspect. Our approach differs from previous research proposals in the way BPMN is used: instead of extending or enriching the notation, we represent different kinds of duration constraints directly with BPMN, by designing re-usable process models that fully exploit the advanced event handling mechanisms present in the standard. Despite requiring us some initial modeling effort, the obtained processes provide a clear conceptualization of duration-aware process activities and regions, entirely based on the standard BPMN semantics [11, 101]. This latter aspect is very relevant when considering existing standard-compliant tools and engines. Moreover, the continuous addition of extension elements to the notation requires the definition of a novel operational semantics, potentially increasing the complexity of the standard and leading to ad-hoc solutions scarcely used in practice [129].

The process models designed in this chapter are well-structured and intended to be used as base building blocks for modeling duration-aware processes. That is, we do not expect process designers to come up themselves with the proposed models, but rather provide them with ready-to-use solutions to model duration constraints in a standard-compliant way.

Our proposal focuses on fulfilling the following research requirements, which have been selected from a broad set of requirements that are frequently advocated in literature in the fields of BPM [16, 18, 23, 124, 27, 29, 125, 130], (temporal) information systems [112, 131], and temporal constraint networks [132, 133, 134].

- *Temporal Management*. In business process management, temporal aspects are a very important issue to consider [16, 18]. Time plays a major role in activity coordination and temporal constraints satisfaction affects process results. Thus, representing and managing temporal constraints is needed during process design and execution.
- *Standard-based modeling*. In business process representation, the use of a standard graphical notation facilitates the understanding of business procedures, internal collaborations, coordination between activities, and tool support [7].
- *Designer Support*. In business process modeling, designers and analysts need to be supported in order to be able to easily model business activities and the related temporal aspects [23, 29].
- *Modularity/Re-usability*. In business process modeling, modularity and re-usability reduce design complexity and improve both readability and management of complex process models. Having ready-to-use (sub)process models facilitates the modeling of complex processes [132].

This chapter deals with the specification of different kinds of duration constraints in BPMN and tackles the detection and management of constraint violations during process run-time [27, 124].

We begin with proposing a basic process model for specifying the duration of a given activity. In this regard, we discuss the possibility of having boundary events influencing the execution of a constrained activity and propose suitable models for specifying such constraints and for discerning different causes of activity interruption.

Then, moving from simple activities to more complex process parts, we deal with the specification of duration constraints of SESE regions, and of arbitrarily selected Non-Single-Entry-Single-Exit (non-SESE) regions [107].

Beside simple duration, we address two more variants of activity duration: (i) one deals with the possibility of dynamically specifying activity duration after its initiation (*deferred duration*), while (ii) the other one regards *shifted duration*, i.e., a duration that is measured starting only from a relevant moment, after activity initiation [53]. To complete the picture, we introduce suitable techniques that can be included in the proposed process models in order to detect and handle duration violations.

Among various domains suitable for the application of BPM techniques, clinical working environments cover a role of primary importance with respect to the temporal perspective [27, 51]. Indeed, time is extremely relevant in healthcare processes, as patients lives are involved, resources are limited, and clinical procedural aspects must be integrated with organizational and administrative prac-

tice [26]. For this reason, we often refer to real-world clinical examples throughout the chapter, without compromising the generality of the proposed approach.

The remainder of the chapter is structured as follows. Sect. 3.1 provides the reader with basic background concepts related to BPMN and time, while Sect. 3.2 summarizes the considered duration constraints. Sect. 3.3, Sect. 3.4, and Sect. 3.5 describe the structure and behavior of the introduced process models. Sect. 3.6 presents possible strategies for detecting and handling duration violations. Sect. 3.7 discusses the verification of the proposed process patterns based on time Petri nets. Finally, Sect. 3.8 draws some conclusions.

3.1 Modeling Temporal Aspects in BPMN processes

To begin with, let us consider the process for the diagnosis and management of patients diagnosed with appendicitis, exemplified in Fig. 3.1.

Example 3.2 (Diagnosis and Management of Appendicitis). Appendicitis is a common gastrointestinal infection necessitating prompt surgical intervention [135]. Choices of operative strategy and timing of intervention have been reported to influence outcomes and hospital length of stay.

For simplicity, we consider dealing with patients having perforated appendicitis and requiring emergency appendectomy. The process of Fig. 3.1 begins with start event S, which leads the sequence flow towards task Patient Evaluation during which the patient is interviewed and individual clinical variables are assessed. The following collapsed subprocess Diagnosis includes multiple inter-related diagnostic steps, such as biochemical testing, ultrasound, and additional imaging tests, that aim to exclude complications other than appendicitis.

Then, exclusive gateway associated to question Acute appendicitis diagnosed? split the process flow in two paths, one towards task Additional Clinical Assessment

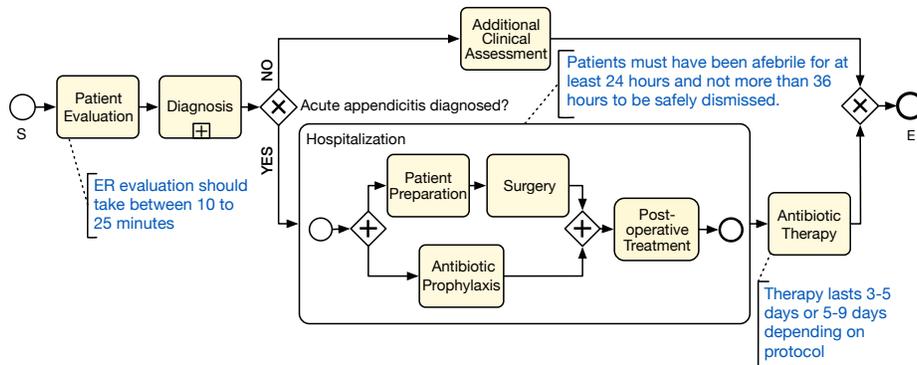


Fig. 3.1: Sample BPMN process representing selected steps of the diagnosis and management of patients requiring emergency appendectomy.

if appendicitis is excluded, the other leading to expanded subprocess **Hospitalization** if a diagnosis of appendicitis is confirmed. Hospitalization is functional to appendectomy, the surgical removal of the infected appendix. **Surgery** may either follow the open approach or the laparoscopic one, depending on individual patient needs and resource availability. Usually, preoperative **Antibiotic Prophylaxis** is commenced once the diagnosis of appendicitis is established and may continue during surgery.

Postoperative antibiotic treatment is considered standard of care in cases of perforated appendicitis. After surgery, **Postoperative treatment** is administered in hospital until discharge, while oral **Antibiotic therapy** may continue for some days at home. The process ends with end event **E**.

From the context described by Example 7.2 it is clear how timing is crucial for patient safety.

In general, clinical diagnosis and assessment activities must have reasonable durations to avoid mistakes due to hasty decision-making or dangerously long waiting times that may lead to infections of the surgical site. Similarly, both hospital length of stay and the duration of antibiotic regimens are of crucial importance for optimal process outcome, being influenced by clinical events and influencing overall process duration. However, their duration cannot be expressed by a simple duration constraint.

For example, let us consider discharge planning of an hospitalized patient.

Example 3.3 (Discharge from hospital). Let us consider **Hospitalization** as a general medical task. In order to safely discharge a patient, physicians must ensure that he or she has had no fever (**NF**) for the 24 hours preceding discharge. Otherwise, the patient is kept in hospital until fever resolves and the patient has remained afebrile for 24 hours.

Fig. 3.2(a) shows task **Hospitalization**, where a duration of 24 hours is measured after the occurrence of event **NF**, which denotes the beginning of apyrexia.

However, fever resolution may be followed by future episodes of fever (**F**). In this scenario, physicians must wait until fever resolves again before starting to measure the 24-hour interval required for safe discharge. Fig. 3.2(b) shows the explained setting: if fever **F** occurs within the 24-hour interval following **NF**, the duration count is reset and physicians must wait until fever resolves again (**NF**) before restarting duration measurement.

Example 3.3 illustrates that the specification of minimum and maximum turnaround times of subprocess **Hospitalization** is not sufficient to capture the duration of the subprocess. In the remainder of this chapter, we focus on specifying the duration constraints exemplified in the text annotations of Fig. 3.1.

Although business processes describe sequences of actions that follow an ordering that is homogeneous to the flowing of time, support of temporal perspective of business processes remains limited [16, 23, 32]. Among several design

levels, conceptual modeling is probably the most affected by this lack of expressiveness [119], as there exist control and scheduling tools that manage certain temporal constraints during process run-time.

When modeling business processes from a temporal standpoint the temporal dimension of the process elements, whenever defined, depends on the modeling language used and on its semantics. Since many process modeling languages focus on control flow aspects, designing and visualizing temporal properties in process models becomes quite challenging, especially because of the lack of a common, clear temporal semantics.

Usually, it is assumed that the process sequence flow, gateways, and event triggering are instantaneous, that is, their execution does not consume any time or it consumes a fixed amount of time, which does not change during process execution [33]. This allows designers to ignore the temporal contribute brought by these constructs whenever computing the overall process duration or inferring temporal dependencies among flow elements.

An exception is represented by *catching events* that, once enabled, may need to wait for a certain amount of time prior to being activated by a trigger. For instance, a *timer event* having attribute `timeDuration` set to “one hour” will be triggered one hour after being enabled [11]. Similarly, a *message event* could potentially wait for one second or to an indefinite amount of time for the corresponding message trigger to arrive. However, it is reasonable to think that events will be triggered after a finite and practical amount of time and that any two event instances cannot occur exactly at the same point in time, assuming sufficient clock precision [33]. Therefore, it is always possible to temporally order process events occurrence. Whenever it is necessary to model that two events happening contemporaneously, it is sufficient to show that their occurrence has no predefined order, that is, all the possible interleavings of such events are admitted.

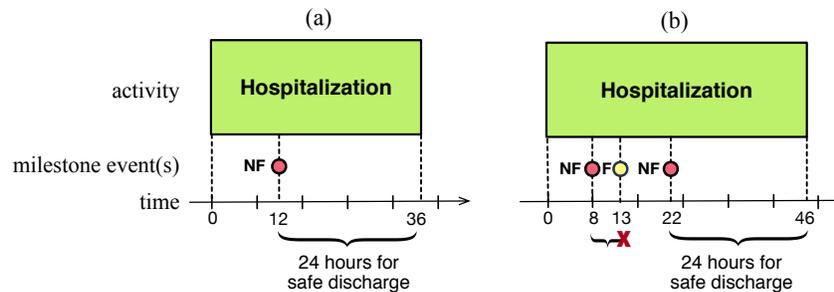


Fig. 3.2: (a) 24 hours are measured starting from the occurrence of event NF, which denotes the beginning of apyrexia. (b) The reoccurrence of F within the 24-hour interval, causes duration measurement to be reset.

Activities take time to be executed. A *duration* is defined as “an amount of time with known length, but no specific starting or ending instants” [136].

For example, a duration of “one week” is known to last seven days, but it can refer to any temporal block of seven consecutive days. It is widely accepted that duration is non-directional with respect to a timeline, that is, it is always positive. In this chapter, for simplicity, we deal with a discrete time domain, i.e., we assume that time is discrete and isomorphic to natural numbers. As a consequence, the shortest interval has a duration of one time unit.

An activity, being either an atomic task or a compound subprocess, is expected to last a precise amount of time, within a range delimited by minimum duration MIN and maximum duration MAX values ($d \in [\text{MIN}, \text{MAX}]$). *Minimum duration* specifies that a certain activity must not complete earlier than a preset time point, that is, it should take at least a specified amount of time to be performed. A suitable clinical example for describing a minimum duration constraint is antibiotic therapy, which would result ineffective if administered for less than a certain number of days. On the other hand, a *maximum duration* constraint is used to set the upper-most time limit after which the activity is intended to have terminated. For example, maximum duration for taking the database systems exam is fixed to 3 hours. Students that do not hand in their tests within 3 hours will not be evaluated.

Let us consider the activity life-cycle depicted in Fig. 2.2. In the remainder of this chapter, when speaking about activity duration, we refer to the amount of time during which the activity is RUNNING, i.e., we do not consider the time span during which the activity is enabled but not yet started [17].

3.2 Process Models for Specifying Activity Duration

In this section, we recall some useful foundational concepts and introduce the different kinds of duration constraints addressed in this chapter with the help of tables that summarize their main features and report meaningful examples.

According to Def. 2.1, a process model is a tuple $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$. Without loss of generality, in the remainder of this chapter, we consider dealing with process models conforming to the above definition, but such that $DN = \emptyset$, $TA = \emptyset$, $F = \emptyset$, $T = \emptyset$, and $R = \emptyset$. In other words, we leave the data and resource perspectives out to better focus on the temporal one.

The semantics of the introduced elements is assumed to be the one defined by BPMN [11, 101], and introduced in Sect. 2.1.2.

As for the notation used in this chapter, let θ be one of functions α , ϵ_{tr} , ϵ_{ty} , β , γ_r , and γ_{ty} belonging to a process model m . For practicality, we write $\theta(\{n_1, \dots, n_m\}) = val$, as a short-cut for $\forall n_i \in \{n_1, \dots, n_m\} \theta(n_i) = val$.

As discussed in Sect. 2.1.2, if a process model m has unique start event s , a unique end event e , and each node $n \in \{A \cup G \cup E_{int}\}$ lies on a path from s to e , then m is said to be structurally sound [137]. Despite being a desirable property,

we do not require end events to be unique as some exception flows, such as those outgoing of non-interrupting events, shall have with their own end event [11]. Instead, we always assume dealing with process models having a unique start event (cf. structural criteria (iv) and (v) on page 29).

When composing multiple process models into a comprehensive one, elements such as flow nodes or control flow edges may be used to ensure that their flows are correctly connected and that their event handling mechanisms are consistent. We refer to the collection of such process elements as *connecting kit*.

Definition 3.1 (Connecting Kit). A connecting kit $Ck = (N, C)$ consists of a finite set of flow nodes N and a finite non-empty set of control flow edges C representing a graph not necessarily connected.

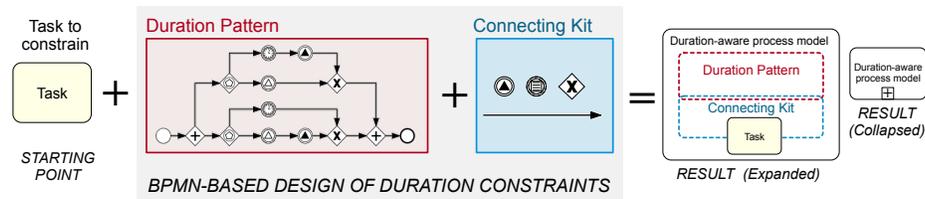


Fig. 3.3: Starting from a process Task (or region) whose duration needs to be constrained, we design a (set of) Duration Pattern(s) that can be attached to the task through suitable BPMN elements forming a Connecting Kit to obtain a complete Duration-aware process model.

As previously mentioned, our aim is to capture different kinds of duration constraints by means of dedicated process models, which we refer to as *duration patterns*. As outlined in Figure 3.3, starting from a process task (or region) to constrain, we design a duration pattern that captures the specified duration constraint. The duration pattern is attached to the activity through a set of additional BPMN flow and connecting elements, forming a *connecting kit* that is used to anchor the pattern to the task (or region) and to refine event handling. The resulting combination of the temporally constrained task (or region) with the duration pattern realized by the connecting kit, is a complete *duration-aware process model*.

Such duration-aware process models may include different duration patterns depending on the kind of duration constraint that is represented.

In the remainder of the section, tables are used to summarize and exemplify the different kinds of considered duration constraints.

Table 3.1 describes duration constraints for process activities, considering also tasks with attached boundary events, and considers the specification of simple duration for more complex process regions, such as SESE or non-SESE regions. Table 3.2 exemplifies some settings that require one to dynamically choose a

proper duration range for an activity, after its initiation (deferred duration). Finally, Table 3.3 introduces how to specify a shifted duration for an activity, considering also resetting it whenever environmental conditions change.

3.3 Specifying Simple Duration Constraints

In this section, we recall and formalize the structure of the duration-aware process model initially proposed in [52] for specifying the simple duration of a process activity. This initial solution is the basic building block for the modeling of more complex constraints, addressed later in this work.

The simple duration of an activity a can be specified through a structurally sound process model, namely duration pattern ϕ_{simple} , that is meant to be suitably combined with a through a connecting kit Ck_1 .

Formally, $\phi_{simple} = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \rho, \mathcal{L})$ has the following structure.

- $N = \{A \cup G \cup E\}$ is the set of flow nodes, where:
 - $A = \emptyset$; $G = \{g_1, g_2, g_3, g_4, g_5, g_6\}$; $E = \{E_{start} \cup E_{int} \cup E_{end}\}$ where $E_{start} = \{s\}$, $E_{int} = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, and $E_{end} = \{e\}$;
- $DN = \emptyset$;
- $C = \{(s, g_1), (g_1, g_2), (g_1, g_3), (g_2, e_1), (g_2, e_2), (g_3, e_3), (g_3, e_4), (e_4, e_6), (e_1, e_5), (e_5, g_4), (e_2, g_4), (e_3, g_5), (e_4, g_5), (g_5, g_6), (g_4, g_6), (g_6, e)\}$ is the set of control flow edges;
- $TA = \emptyset$;
- $F = \emptyset$;
- $T = \emptyset$;
- $R = \emptyset$;
- $\alpha_k = \emptyset$;
- $\alpha_t = \emptyset$;
- $\beta = \emptyset$;
- $\gamma_r(\{g_1, g_2, g_3\}) = split$, $\gamma_r(\{g_4, g_5, g_6\}) = merge$;
- $\gamma_{ty}(\{g_1, g_6\}) = parallel$, $\gamma_{ty}(\{g_4, g_5\}) = exclusive$, and $\gamma_{ty}(\{g_2, g_3\}) = event-based$;
- $\epsilon_{tr}(\{s, e_1, e_2, e_3, e_4\}) = catching$, $\epsilon_{tr}(\{e_5, e_6, e\}) = throwing$;
- $\epsilon_{ty}(\{s, e\}) = none$, $\epsilon_{ty}(\{e_1, e_3\}) = timer$, $\epsilon_{ty}(\{e_2, e_4, e_5, e_6\}) = signal$;
- $\epsilon_k = \emptyset$;
- $\rho = \emptyset$;
- $\mathcal{L}(e_2) = \mathcal{L}(e_4)$.

Functions α_k , α_t , β , ϵ_k , and ρ are empty since $A = \emptyset$, $E_B = \emptyset$, and $R = \emptyset$. Events e_2 and e_4 are given the same label to highlight that they have the same triggering mechanism, i.e., they catch the same trigger.

Connecting kit $Ck_1 = (N, C)$ where $N = \{e_7\}$ and $C = \{(g_1, a), (a, e_7), (e_7, g_6)\}$. Event e_7 is an intermediate throwing signal event that triggers both events e_2 and e_4 of ϕ_{simple} , whenever they are actively listening.

SIMPLE DURATION OF AN ACTIVITY (TASK OR SUB-PROCESS)	Sec. 3.3.3
<p>CONSTRAINT DESCRIPTION</p> <p>The time span during which a task is executed is restricted by minimum [MIN] or maximum [MAX] duration bounds.</p> <p>REAL-WORLD EXAMPLES</p> <ul style="list-style-type: none"> • The minimum acceptable duration of anticoagulation therapy for venous thromboembolism is 3 months. [MIN] • Nimesulide should not be given for periods longer than 7 days in the treatment of acute pain. [MAX] • A taxi ride from the airport to the city center will take from 30 to 60 minutes, depending on traffic. [MIN and MAX] 	
SIMPLE DURATION OF AN ACTIVITY WITH BOUNDARY EVENTS	Sec. 3.3.1
<p>CONSTRAINT DESCRIPTION</p> <p>The duration of a task is restricted by minimum [MIN] or maximum [MAX] duration bounds, and has attached interrupting [I] or non-interrupting [NI] events.</p> <p>REAL-WORLD EXAMPLES</p> <ul style="list-style-type: none"> • A fluid challenge must be infused within 30 minutes in critically ill patients, as slower rates of infusion are deemed less effective. Fluid infusion must be interrupted if cardiac filling pressures increase to critical levels. [MIN and MAX, I] • Aspirin therapy for prevention of thrombotic events should last 28 days. If the patient experiences any upper gastrointestinal complication, gastro-protective co-therapy may be initiated. [MIN and MAX, NI] 	
SIMPLE DURATION OF A SESE REGION	Sec. 3.3.2
<p>CONSTRAINT DESCRIPTION</p> <p>The time span during which a sequential [→], parallel [+], or exclusive [×] SESE region is executed is limited by minimum [MIN] and maximum [MAX] duration bounds.</p> <p>REAL-WORLD EXAMPLES</p> <ul style="list-style-type: none"> • Exercise stress test lasts at least 45 minutes, comprehensive of preparation and monitoring times. [→, MIN] • Therapy for H. Pylori eradication combines antibiotics and proton pump inhibitors for an overall duration of 7-10 days. [MIN and MAX, +] 	
SIMPLE DURATION OF A NON-SESE REGION	Sec. 3.3.2
<p>CONSTRAINT DESCRIPTION</p> <p>The time span during which a non-SESE region is executed is restricted by minimum [MIN] and maximum [MAX] duration bounds.</p> <p>REAL-WORLD EXAMPLES</p> <ul style="list-style-type: none"> • On-scene time for adults trauma should be less than 10 minutes, during which airway with cervical spine control, breathing, circulation and disability are checked, depending on the patient's gravity and the probability of head trauma [MAX]. 	

Table 3.1: Simple duration of process activities and regions.

DEFERRED DURATION OF AN ACTIVITY	Sec. 3.4
<p>CONSTRAINT DESCRIPTION</p> <p>A task can be associated to multiple duration intervals and a default one can be chosen prior to activity initiation. At run-time only one interval is selected, based on how activity execution evolves. Therefore, the choice of which duration range applies is <i>deferred</i> with respect to activity initiation.</p>	
<p>REAL-WORLD EXAMPLES</p> <ul style="list-style-type: none"> • Let us consider the treatment for pharyngitis. If the infection is viral, antibiotic therapy lasts 5 days, whereas for bacterial pharyngitis it should last 10 days. As it is hard to distinguish viral and bacterial causes based on symptoms alone, an empiric treatment is usually initiated, while a throat culture is grown. When results are obtained, treatment is specialized and continued based on the discovered nature of the infection. • Hospitalization following a diagnosis of appendicitis can last 1-2 days if surgery was routine, or up to 4 days if the removed appendix is found to be ruptured during the intervention. 	

Table 3.2: Deferred duration of an activity: the choice of which one among multiple duration ranges applies is made while the activity is being executed.

SHIFTED DURATION OF AN ACTIVITY	Sec. 3.5
<p>CONSTRAINT DESCRIPTION</p> <p>The duration of a task is restricted by minimum [MIN] or maximum [MAX] duration bounds, and it is measured starting from a specific (<i>shifted</i>) moment indicating that a certain property begins to hold. We call this particular moment <i>milestone event</i>. Duration count is reset [R] whenever this property stops holding earlier than the set minimum duration.</p>	
<p>REAL-WORLD EXAMPLES</p> <ul style="list-style-type: none"> • To prepare medium boiled eggs, put the eggs in a saucepan filled with cold water. Set the pan over medium-high heat and as soon as the water boils, start timing 4 to 5 minutes. [MIN and MAX] • Hospitalization must last between 24 and 36 hours, starting from the moment the patient defervesces (milestone event). If the patient has fever prior to 24 hours, duration count is reset and will re-start once fever disappears again. [MIN and MAX, R] 	

Table 3.3: Shifted duration of an activity, considering also the possibility of resetting the duration count.

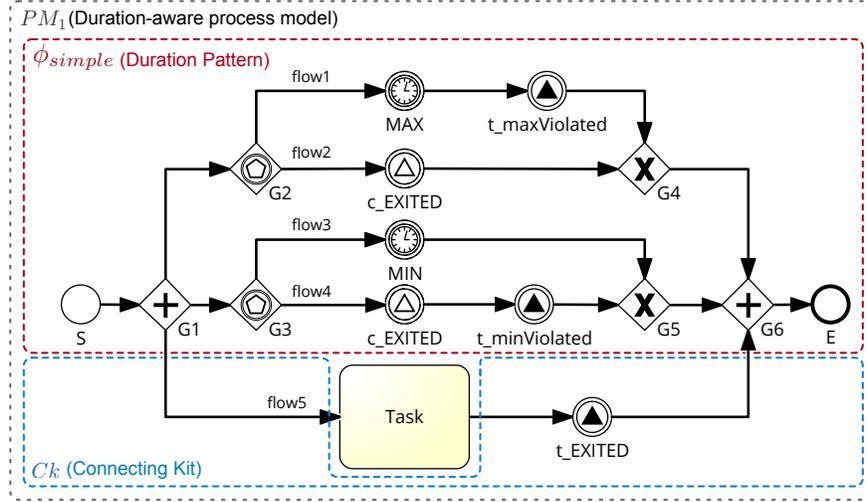


Fig. 3.4: BPMN duration-aware process model for specifying the simple duration $d \in [MIN, MAX]$ of Task. Considering Fig. 3.3, PM_1 corresponds to the RESULT obtained for capturing the simple duration of Task.

As outlined in Fig. 3.3, the combination of a , Ck_1 , and ϕ_{simple} results in a complete duration-aware process model $PM_1 = (N \cup \{a, e_7\}, DN, C \cup \{(g_1, a)\} \cup \{(a, e_7)\} \cup \{(e_7, g_6)\}, TA, F, T, R, \alpha_k \cup \{a \mapsto task\} \text{ or } \{a \mapsto subprocess\}, \alpha_t \cup \{a \mapsto abstract\}, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr} \cup \{e_7 \mapsto throwing\}, \epsilon_{ty} \cup \{e_7 \mapsto signal\}, \epsilon_k, \rho, \mathcal{L} \cup \mathcal{L}(a) \cup \mathcal{L}(e_7)\}$.

Process model PM_1 can be included within more complex parent processes and represented as a collapsed subprocess. In other words, a duration-aware process model can be considered as a subprocess template that designers can use to specify task duration [52].

Without loss of generality, in Fig. 3.4 activity a is represented as a task, i.e., $\alpha(a) = task$ and $\mathcal{L}(a) = Task$. The flow nodes of duration pattern ϕ_{simple} are labeled as follows: $\mathcal{L}(s) = S$, $\mathcal{L}(e_1) = MAX$, $\mathcal{L}(e_2) = c.EXITED$, $\mathcal{L}(e_3) = MIN$, $\mathcal{L}(e_4) = c.EXITED$, $\mathcal{L}(e_5) = t_maxViolated$, $\mathcal{L}(e_6) = t_minViolated$, $\mathcal{L}(e) = E$, $\mathcal{L}(g_1) = G1$, $\mathcal{L}(g_2) = G2$, $\mathcal{L}(g_3) = G3$, $\mathcal{L}(g_4) = G4$, $\mathcal{L}(g_5) = G5$, $\mathcal{L}(g_6) = G6$, while event e_7 of connecting kit Ck_1 is labeled as $\mathcal{L}(e_7) = t.EXITED$.

The designed process allows us to manage three possible Task execution behaviors with respect to a given duration range: (i) Task is completed within the allowed duration, thus it lasts longer than the minimum desired time limit but ends before the maximum one, (ii) Task ends before the minimum time expected for its completion, thus violating its minimum duration constraint, (iii) Task is still executing when the maximum time limit allowed for completion is reached, thus raising a maximum duration violation.

The process begins when its start event *S* is triggered. The token arrives at parallel gateway *G1*, and the flow is split into three branches: two of them are directed to event-based gateways *G2* and *G3*, while the last one is directed to *Task*. Event-based gateway *G2* represents a branching point in the process where only one path is chosen (either *flow1* or *flow2*) depending on which one of the events in its configuration is triggered first, i.e., event *MAX* on *flow1*, or event *c_EXITED* on *flow2*. Event-based gateway *G3* has the same behavior as *G2* but with respect to events *MIN* on *flow3* and *c_EXITED* on *flow4*.

When *Task* is completed, the token reaches the following (throwing) signal event *t_EXITED*, which is broadcast to be caught by corresponding (catching) signal events *c_EXITED* on *flow2* and *flow4*.

(i) If *Task* completes within the defined duration range, signal event *t_EXITED* is caught only by signal event *c_EXITED* on *flow2*, since timer event *MIN* was triggered previously and, thus, *flow4* was withdrawn.

(ii) If *Task* completes earlier than the minimum duration allowed signal event *t_EXITED* is caught by both events *c_EXITED* on *flow2* and *flow4*. In this case, signal event *t_minViolated* on *flow4* is broadcast to indicate that minimum duration has not been observed.

(iii) If *Task* is still executing when the maximum duration time limit allowed is reached, signal event *t_EXITED* is never caught, as both branches *flow2* and *flow4* are withdrawn (i.e., both timer events *MIN* and *MAX* are fired). Then, signal *t_maxViolated* located on *flow2* is triggered to acknowledge that the maximum duration has been violated.

Signal events *t_minViolated* and *t_maxViolated* are used to detect violations and to trigger other processes designed to handle them, as explained in Sect. 3.6.

To specify only minimum(maximum) duration, the process branch dealing with the maximum(minimum) duration may be simply be removed without altering the well-structuredness of the process.

In dealing with task duration specification, we allow designers to use the proposed BPMN process model PM_1 as a ready-to-use building block whose timer events may be properly tuned to represent a specific duration. For instance, in Fig. 3.5 shows an example of process model including subprocess *Duration-aware Antibiotic Therapy* represents task *Antibiotic Therapy* combined with is duration pattern. For conceptual modeling, designers could also represent the whole duration-aware subprocess as a simple task characterized by a specialized icon such as a “clock”, as done in [35, 119] and shown in the bottom part of Fig. 3.5. Anyhow, being conceptually and semantically defined through BPMN, the behavior of the underlying duration pattern remains clear, regardless of the chosen graphical representation.

3.3.1 Specifying the Duration of an Activity with Boundary Events

In order to provide a complete picture of simple duration specification of an activity, we considered the possibility of having boundary events influencing activity execution and potentially affecting its overall duration.

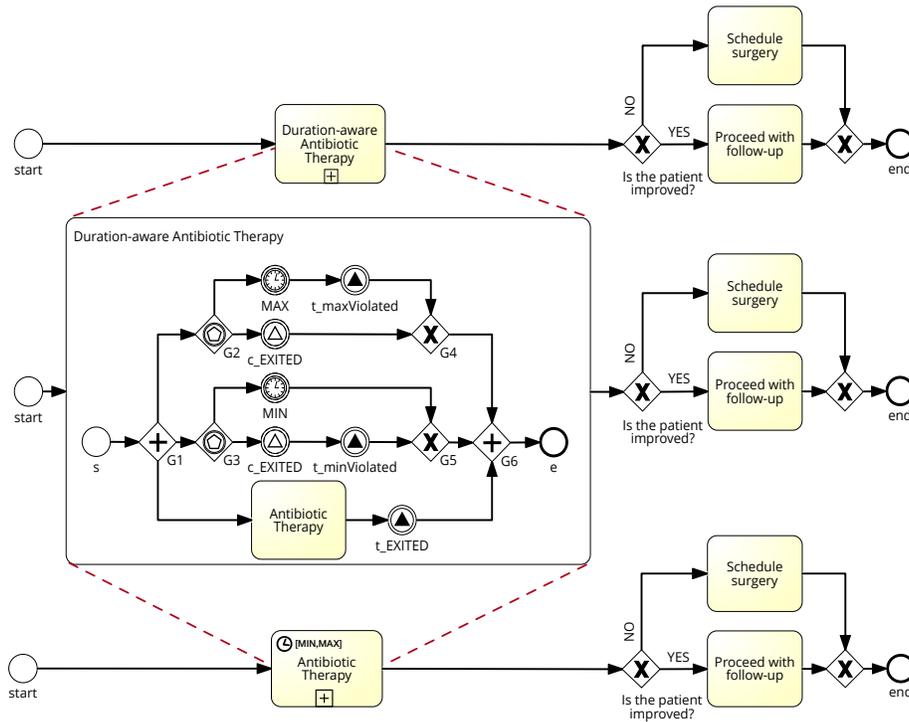


Fig. 3.5: Duration-aware process model for task Antibiotic therapy depicted as subprocess Duration-aware Antibiotic Therapy included in a more complex parent process.

When placed on the boundary of a task or subprocess, an intermediate event is used to represent exception or compensation handling [11].

Once the event occurrence is consumed, the activity can be interrupted if the event is an interrupting one, or activity execution can continue in parallel with the event handler, if the event is non-interrupting.

Although multiple interrupting boundary events can be attached to the same activity, only one of such handlers can be executed at a time, for obvious reasons. Conversely, an unlimited number of non-interrupting event handlers can be modeled and executed in parallel, while the activity continues its execution.

In this latter setting, it is important to recall the previously explained semantic states of a process activity, shown in Fig. 2.2. In particular, we would like to underline that a BPMN activity can switch to state *Completed* only when all the non-interrupting event handlers attached to its boundary are completed. That is, an activity moves from state *Active* to state *Completing* when its execution has finished, but the attached non-interrupting boundary handlers have not yet completed their execution.

The previously introduced duration-aware process model PM_1 can be revisited considering the addition of both interrupting and non-interrupting events attached to the boundary of the considered activity a .

Recalling the approach outlined in Fig. 3.3, we consider starting from an activity a having two intermediate boundary events ei and en , where ei is interrupting and en is non-interrupting.

The core duration pattern is again ϕ_{simple} , but the connecting kit must be re-defined in order to capture also the exception flows originating from the events.

In detail, connecting kit $Ck_2 = (N, C)$, where $N = \{a_1, e_7, e_8, e_9, g_7\}$ and $C = \{(g_1, a), (a, e_7), (e_7, g_7), (ei, e_8), (e_8, g_7), (g_7, g_6), (en, a_1), (a_1, e_9)\}$.

Given a , ei , en , and $\phi_{simple} = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$, and combining them with the elements of Ck_2 , we obtain the complete duration-aware process model $PM_2 = (N', DN', C', TA', F', T', R', \alpha'_k, \alpha'_t, \beta', \gamma'_r, \gamma'_{ty}, \epsilon'_{tr}, \epsilon'_{ty}, \epsilon'_k, \rho', \mathcal{L}')$, where:

- $N' = N \cup \{a, a_1, g_7, ei, en, e_7, e_8, e_9\}$, and $\{a, a_1\} \subseteq A'$, $g_7 \in G'$, and $\{e_7, e_8, e_9\} \subseteq E'$;
- $DN' = \emptyset$;
- $C' = C \cup \{(g_1, a)\} \cup \{(a, e_7)\} \cup \{(e_7, g_7)\} \cup \{(ei, e_8)\} \cup \{(e_8, g_7)\} \cup \{(g_7, g_6)\} \cup \{(ei, e_8)\} \cup \{(en, a_1)\} \cup \{(a_1, e_9)\}$.
- $TA' = \emptyset$;
- $F' = \emptyset$;
- $T' = \emptyset$;
- $R' = \emptyset$;
- $\alpha'_k = \alpha \cup \{\{a \mapsto task\} \text{ or } \{a \mapsto subprocess\}\} \cup \{a_1 \mapsto subprocess\}$;
- $\alpha'_t = \alpha \cup \{\{a \mapsto abstract\} \cup \{a' \mapsto abstract\}\}$;
- $\beta' = \beta \cup \{a \mapsto \{ei, en\}\}$;
- $\gamma'_r = \gamma_r \cup \{g_7 \mapsto merge\}$;
- $\gamma'_{ty} = \gamma_{ty} \cup \{g_7 \mapsto exclusive\}$;
- $\epsilon'_{tr} = \epsilon_{tr} \cup \{\{ei, en\} \mapsto catching\} \cup \{\{e_7, e_8, e_9\} \mapsto throwing\}$;
- $\epsilon'_{ty} = \epsilon_{ty} \cup \{\{e_7, e_8\} \mapsto signal\} \cup \{e_9 \mapsto none\}$;
- $\epsilon'_k = \epsilon_k \cup \{ei \mapsto interrupting\} \cup \{en \mapsto non-interrupting\}$;
- $\rho' = \rho = \emptyset$;
- $\mathcal{L}' \supseteq \mathcal{L}$ and $\mathcal{L}'(e_7) = \mathcal{L}'(e_8)$.

The obtained duration-aware process model is shown in Fig. 3.6. Without loss of generality, activity a is represented as an abstract task, i.e., $\alpha_k(a) = task$, $\alpha_t(a) = abstract$ and $\mathcal{L}'(a) = Task$, while boundary events are labeled as $\mathcal{L}'(ei) = InterruptingE$ and $\mathcal{L}'(en) = Non-InterruptingE$. The flow nodes of duration pattern ϕ_{simple} are labeled as in Fig. 3.4, while the flow nodes of Ck_2 are labeled as follows: $\mathcal{L}'(a_1) = Non-interrupting Event Handler$, $\mathcal{L}'(e_7) = t.EXITED$, $\mathcal{L}'(e_8) = t.EXITED$, $\mathcal{L}'(e_9) = exE$, and $\mathcal{L}'(g_7) = G7$.

As for the type of boundary events, in Fig. 3.6 we employ a *none* intermediate event placed on the activity boundary although this kind of event is not defined in BPMN as a possible catching event. We chose to use it as a graphical expedient

for the sake of simplicity, as the behavior of PM_2 remains the same for any of the following BPMN triggers: *Signal*, *Timer*, *Message*, *Conditional*, and *Escalation*.

Specifically, any *Signal* or *Message* event can be sent by any external participant or process instance. A *Conditional* event is triggered when a condition becomes true, whereas an *Escalation* event triggers a reaction to a specific business situation. *Timer* events are assumed to be triggered at a fixed *timeDate*, such as “February 23rd 2015”, *timeCycle* for example “every Monday at 9.00 a.m.” or *timeDuration*, such as “2 hours”. Of course, when defined with an absolute *timeDate*, their triggering depends on when the process is executed.

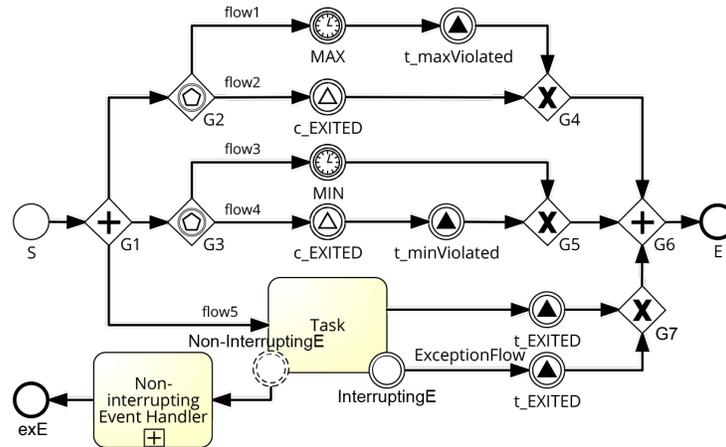


Fig. 3.6: Duration-aware process model representing the duration of a *Task* having attached interrupting and non-interrupting intermediate boundary events.

In the presented solution, the exception flow outgoing from interrupting event *InterruptingE* is alternative to the normal flow outgoing from *Task*. Compared to the process of Fig. 3.4, in this setting the semantic state of *Task* changes to *Terminated* if the interrupting event is triggered. Therefore, signal event *t.EXITED* can be replicated within *exceptionFlow* to be used in case of minimum constraint violation caused by early task interruption. Besides this, no substantial change in the model is required to detect constraint violation, as the behavior of ϕ_{simple} remains unvaried.

Regarding the occurrence of interrupting boundary events and timer events *MIN* and *MAX*, it is worth noticing that an interrupting event may occur either earlier or later than *MIN*, conjectured that it occurs before *Task* is completed. Conversely, an interrupting event can occur after the maximum duration only if *Task* is violating the constraint set by *MAX*. That said, we can conclude that interrupting boundary events can directly raise minimum constraint violations whenever *Task* is interrupted before the triggering of *MIN*. However, being de-

signed as an exception, this latter kind of violation is a predictable temporal exception, properly captured by event handlers in the process model.

As an example, consider the process model of Fig. 3.7, depicting activity fluid challenge constrained between 15–30 minutes.

Example 3.4 (Fluid Challenge). A Fluid challenge is the rapid infusion of intravenous fluids, usually crystalloids or colloids, for the reversal of hypovolemia in critically ill patients [138]. Fluid infusion is constrained between 15–30 minutes, as infusion times shorter than 30 minutes are proven more effective [138].

However, rapid fluid administration is known to increase cardiac filling pressures. Since increased cardiac filling pressures may result in life-threatening fluid overload, fluid infusion must be interrupted when critical levels of pressure are reached [139]. Fluid overload is initially treated with loop diuretics and then reversed with the help of Renal replacement therapy.

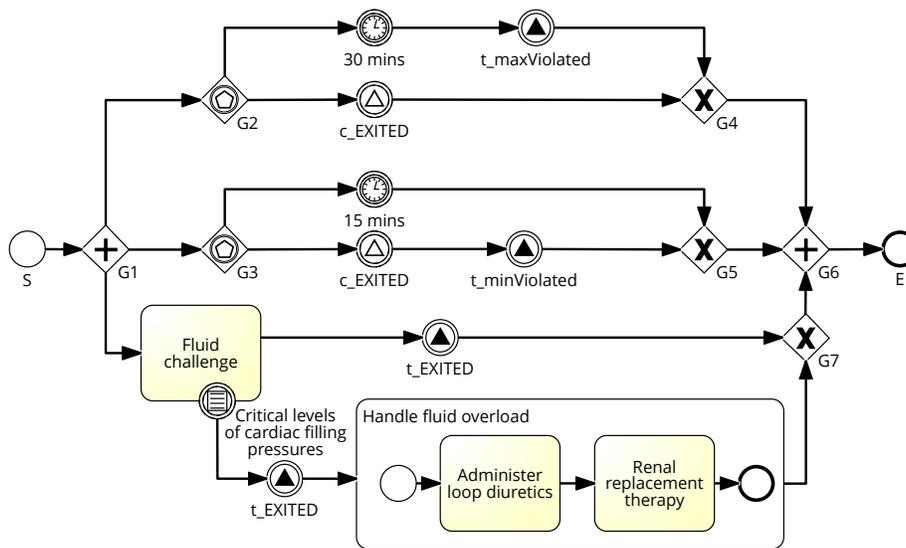


Fig. 3.7: Process model representing minimum and maximum duration constraints for the Fluid Challenge, which is interrupted if Critical levels of cardiac filling pressures are reached.

Instead, non-interrupting boundary events do not directly affect the activity execution with respect to duration constraints. However, the execution of event handlers may have some repercussion on the overall activity duration.

As previously mentioned, for a BPMN activity to be in state *Completed*, all attached non-interrupting event handlers must have reached a completion state. With respect to activity duration, this translates into a relaxation of the control

over the exact amount of time needed for **Task** to execute, since an activity that is completed within the expected duration range might have to wait for its non-interrupting event handlers to proceed. At the level of abstraction adopted in the process of Fig. 3.6, there is no means to distinguish if the maximum duration constraint is violated by the activity itself, thus the activity is still in state *Active*, or by the non-interrupting event handlers started by **Non-InterruptingE**, i.e., the activity is in state *Completing*.

Additionally, if we consider that exception handlers may themselves be composed of activities, constraining the duration of an activity subjected to the occurrence of handlers executed in parallel becomes challenging. Similarly, we may have a case of an activity that completes earlier than its minimum expected duration, but it remains in state *Completing* while it waits for attached non-interrupting event handlers to complete. In this case, its overall duration appears to be within the desired time constraints. From this setting, we can evince that, in order to model the real duration of an activity, any non-interrupting event handler associated to **Task** must end within the same maximum time limit set for its parent activity nor it should not delay activity completion.

3.3.2 Specifying Simple Duration Constraints of Process Regions

In this section, we discuss how to specify the simple duration of Single-Entry-Single-Exit(SESE) process regions either composed solely of tasks or delimited by split and merge parallel or exclusive gateways. Then, we consider specifying the duration of non-Single-Entry-Sigle-Exit (non-SESE) regions and extend the set of process patterns presented in [52] to constrain the duration of arbitrarily selected regions having multiple entry or exit nodes.

A SESE region $R_{(en,ex)}$ is a (subset of a) process model delimited by an entry node en and an exit node ex having the following properties [140]:

1. Every path from the start of the process to ex includes en ;
2. Every path from the end of the process to en includes ex ;
3. Every cycle containing en also contains ex and vice-versa.

Fig. 3.8 shows the previously introduced duration pattern ϕ_{simple} attached to three main kinds of SESE regions having distinct entry and exit nodes. Namely, we consider process regions composed by a sequence of tasks $t_1 \dots t_n$, process regions delimited by exclusive gateways (G_{en} and G_{ex}), and process regions delimited by parallel gateways (again, G_{en} and G_{ex}).

For readability, Fig. 3.8 depicts only two process branches within exclusive and parallel blocks, but the discussed solutions hold for SESE regions enclosing an arbitrary number of flow branches.

Starting from a SESE region $R_{(en,ex)}$ delimited by one entry en and one exit node ex , duration pattern ϕ_{simple} remains untouched and it is anchored to R through connecting kit $Ck_3 = (N, C)$, where $N = \{e_7\}$ and $C = \{(g_1, en), (ex, e_7), (e_7, g_6)\}$. Trivially, if R is a task, then Ck_3 is equal to Ck_1 defined in Sect. 3.3 and en and ex coincide with the task.

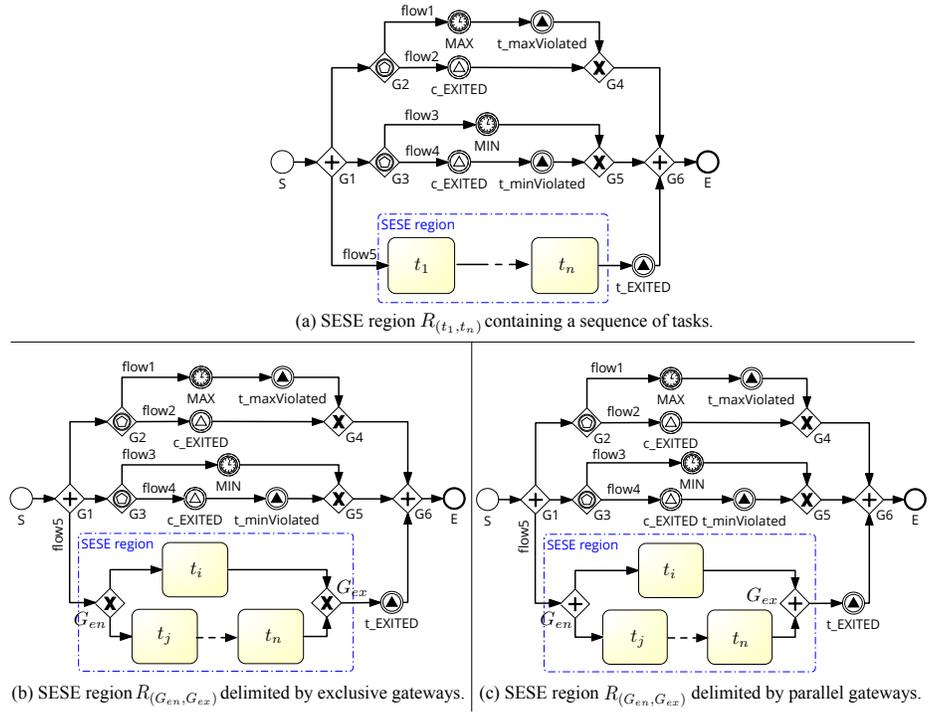


Fig. 3.8: Duration pattern ϕ_{simple} applied to different Single-Entry-Single-Exit (SESE) regions. (a) Sequence of tasks t_1, \dots, t_n ; (b) SESE region delimited by exclusive gateways G_{en} and G_{ex} ; and (c) SESE region delimited by parallel gateways G_{en} and G_{ex} .

The duration-aware process models obtained by combining SESE regions with duration pattern ϕ_{simple} and connecting kit Ck_3 are shown in Fig. 3.8.

First, let us consider a SESE region $R_{(t_1, t_n)}$ composed of a sequence of tasks t_1, \dots, t_n , as the one shown in Fig. 3.8(a). The duration of R is the time that elapses from the beginning of the first element of the sequence t_1 , which is the entry node of the region, to the ending of last one t_n , which is the exit node of the region. Signal event $e_7 \in Ck_3$ (labeled as t_EXITED) is placed after t_n .

A SESE region $R_{(G_{en}, G_{ex})}$ delimited by exclusive gateways G_{en} and G_{ex} is shown in Fig. 3.8(b). Its duration corresponds to the time that elapses from the entry node G_{en} to the exit node G_{ex} . Signal event $e_7 \in Ck_3$ (labeled as t_EXITED) is placed after the exit node of the region, i.e., G_{ex} .

Similarly, Fig. 3.8(c) considers duration of a SESE region $R_{(G_{en}, G_{ex})}$ delimited by parallel gateways G_{en} and G_{ex} .

SESE regions can be arbitrarily complex, as they may contain other nested SESE regions. However, since the proposed solution exploits modularity, parallel gateway g_1 of ϕ_{simple} can always be connected to the entry node of the SESE

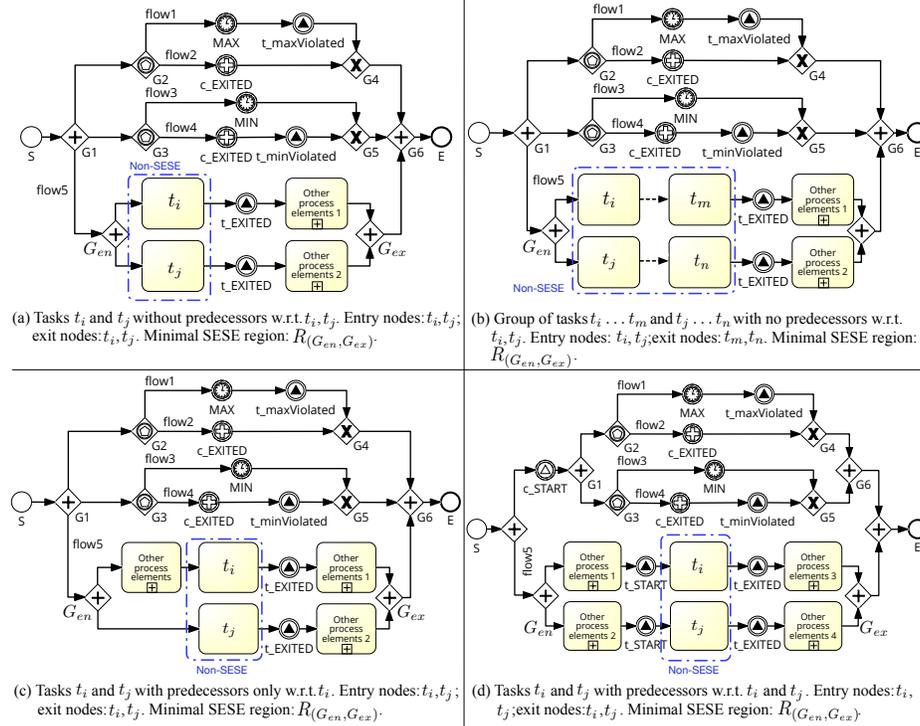


Fig. 3.9: Non-Single-Entry-Single-Exit (Non-SESE) regions distributed across two flow branches of a SESE region delimited by parallel gateways. (a) Coinciding entry and exit points, no predecessors; (b) No predecessors; (c) There is at least one entry node without predecessors; and (d) All entry nodes have predecessors.

region of interest, while the exit node of the region can be connected to signal event e_7 of Ck_3 that leads back to parallel gateway g_6 of ϕ_{simple} .

Compared to dealing with SESE regions, when considering non-Single-Entry-Single-Exit regions it is important to keep in mind that duration is influenced by multiple starting and ending points that need to be “synchronized” [52], i.e., a non-SESE region $NR_{(\{en_1, \dots, en_n\}, \{ex_1, \dots, ex_m\})}$ is delimited by a set of entry nodes and a set of exit nodes, where there exist a partial order between entry and exit nodes.

In addition, both the structure of the non-SESE region and the way it is connected to the remaining elements of the process model affect the structure of the related duration patterns.

Accordingly, we start by dealing with basic non-SESE regions and consider an increasing number of structural features throughout the remainder of this section, to show how duration patterns may be adapted to deal with complex non-SESE regions.

As a first step, we derive non-SESE regions from the basic kinds of SESE regions outlined in Fig. 3.8. Trivially, there is no way of defining a non-SESE region spanning a sequence of tasks. Similarly, considering non-SESE regions spanning multiple alternative process branches is nonsense, since only one of them is executed at a time. Thus, we are interested in non-SESE regions that span at least two parallel process branches.

By taking the SESE region of Fig. 3.8(c) as a reference, we retrieve four possible arrangements of activities enclosed within a SESE region $R_{(G_{en}, G_{ex})}$ delimited by parallel gateways. We report them in Fig. 3.9 and discuss how to specify the duration of the non-SESE region composed by such activities.

In Fig. 3.9, process elements located within $R_{(G_{en}, G_{ex})}$ but not belonging to the non-SESE region to constrain are represented as collapsed subprocesses labeled **Other process elements**. We refer to such process elements as *predecessors* or *successors*, based on their position with respect to the entry and exit nodes of the considered non-SESE region, framed by a (blue) dash and dotted line.

In particular, Fig. 3.9 shows all the possible arrangements of predecessors with respect to the entry nodes of the considered non-SESE region.

The non-SESE region of Fig. 3.9(a) is constituted by two tasks t_i and t_j that do not have predecessors, but both have successors. The entry and exit nodes of the non-SESE region coincide.

The non-SESE region of Fig. 3.9(b) generalizes the one in Fig. 3.9(a): it is constituted by groups of tasks $t_i \dots t_m$ and $t_j \dots t_n$ that do not have predecessors, but both have successors. Entry points are t_i and t_j , whereas exit points are t_m and t_n .

The non-SESE region of Fig. 3.9(c) is composed by two tasks t_i and t_j where only entry node t_i has predecessors. The entry and exit nodes of the non-SESE region coincide, but this case may be generalized to groups of tasks.

Finally, the non-SESE region of Fig. 3.9(d) is composed by two tasks t_i and t_j that both have predecessors and successors. The entry and exit nodes of the non-SESE region coincide, but this case may be generalized to groups of tasks.

When dealing with non-SESE regions, in order to maintain the design well-structured, duration patterns must be connected to the entry and exit nodes of the *smallest SESE region* enclosing the non-SESE region of interest, e.g., $R_{(G_{en}, G_{ex})}$ in Fig. 3.9. However, this scenario requires that process elements preceding and succeeding the entry and exit nodes of the considered non-SESE region are ignored by the duration pattern and properly managed. Indeed, multiple exit nodes must be synchronized and the presence of predecessors influences when the duration pattern must be enacted.

For this reason, duration pattern ϕ_{simple} must be refined to allow “synchronizing” the multiple entry and exit nodes of the non-SESE region. Similarly, events aimed to “delimit” the region to be constrained along multiple process branches must be included into the connecting kit.

Let us begin with considering only multiple exit nodes. If a non-SESE region has multiple exit nodes, they shall be synchronized, as the last one to complete determines the completion time of the whole region. Instead, since duration is

calculated starting from the first among all entry nodes being enacted, entry nodes do not need to be synchronized.

Consider duration pattern ϕ_{simple} , introduced in Section 3.3. In Fig. 3.8, signal events e_2 and e_4 of ϕ_{simple} , both labeled c.EXITED, are meant to catch the corresponding signal event t.EXITED of connecting kit Ck_1 , placed after the exit point of the considered region.

By following the same design principles, a throwing signal event t.EXITED shall be placed after each one of the exit nodes of the non-SESE region to capture and synchronize multiple exit nodes. However, in order to achieve synchronization, events c.EXITED must be of kind *parallel multiple*, that is, they are assigned arbitrary number of triggers and all of them are required for the event to fire [11]. In this way, events c.EXITED located in the context of event-based gateways G2 and G3, respectively, are triggered only when all the associated t.EXITED events placed on each exit edge of the non-SESE region have fired.

Minimum duration is violated when all events t.EXITED are triggered earlier than MIN, whereas maximum duration is violated when at least one element of the region lasts longer than the maximum time allowed for completion MAX.

We call ϕ_{nSESE} this variant of duration pattern ϕ_{simple} , where the only difference is given by $\epsilon_{ty}(\{e_2, e_4\}) = \textit{parallel multiple}$.

Minimum duration is violated when all the t.EXITED events are triggered earlier than MIN, whereas maximum duration is violated when at least one element of the region lasts longer than MAX, i.e., the maximum time set for completion. Duration pattern ϕ_{nSESE} can be seen in the duration-aware process models of Fig. 3.9(a)–(c), properly combined with the non-SESE region to constrain.

The process model of Fig. 3.9(d) is the only one not having G_{en} as entry node, i.e., each entry node of the non-SESE region has at least one predecessor. As duration pattern ϕ_{nSESE} is anchored to the entry node of the minimal SESE region $R_{(en,ex)}$ containing $NR_{(\{en_1, \dots, en_n\}, \{ex_1, \dots, ex_m\})}$, there must be a way of detecting when the first entry node of the constrained non-SESE region is enacted. To this end, a throwing signal event t.START is added to $R_{(G_{en}, G_{ex})}$, immediately preceding each entry node of the non-SESE region. This event, delimits the boundary of the non-SESE region and triggers the corresponding signal event c.START placed before G1 and used to enable the whole duration pattern whose core is ϕ_{nSESE} .

This more general case of non-SESE regions having predecessors before each entry node and successors following each exit node can be handled by using a combination of signal events t.START and t.EXITED, connected to every entry (respectively exit) node of the region.

We call $\phi_{nSESEPred}$ this variant of ϕ_{nSESE} able to handle predecessors with respect to the entry points of the considered non-SESE region.

$\phi_{nSESEPred} = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$ has the following structure.

- $N = \{A \cup G \cup E\}$ is the set of flow nodes, where:
 - $A = \emptyset$; $G = \{g_1, g_2, g_3, g_4, g_5, g_6\}$; $E = \{E_{start} \cup E_{int} \cup E_{end}\}$ where $E_{start} = \{s\}$, $E_{int} = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, and $E_{end} = \{e\}$;
- $DN = \emptyset$;
- $C = \{(s, g_1), (g_1, g_2), (g_1, g_3), (g_2, e_1), (g_2, e_2), (g_3, e_3), (g_3, e_4), (e_3, e_6), (e_1, e_5), (e_5, g_4), (e_2, g_4), (e_3, g_5), (e_6, g_5), (g_5, g_6), (g_4, g_6), (g_6, e)\}$ is the set of control flow edges;
- $TA = \emptyset$;
- $F = \emptyset$;
- $T = \emptyset$;
- $R = \emptyset$;
- $\alpha_k = \emptyset$;
- $\alpha_t = \emptyset$;
- $\beta = \emptyset$;
- $\gamma_r(\{g_1, g_2, g_3\}) = \textit{split}$, $\gamma_r(\{g_4, g_5, g_6\}) = \textit{merge}$;
- $\gamma_{ty}(\{g_1, g_6\}) = \textit{parallel}$, $\gamma_{ty}(\{g_4, g_5\}) = \textit{exclusive}$, and $\gamma_{ty}(\{g_2, g_3\}) = \textit{event-based}$;
- $\epsilon_{tr}(\{s, e_1, e_2, e_3, e_4\}) = \textit{catching}$, $\epsilon_{tr}(\{e_5, e_6, e\}) = \textit{throwing}$;
- $\epsilon_{ty}(\{s, e\}) = \textit{none}$, $\epsilon_{ty}(\{e_1, e_3\}) = \textit{timer}$, $\epsilon_{ty}(\{e_2, e_4\}) = \textit{parallel-multiple}$, $\epsilon_{ty}(\{e_5, e_6\}) = \textit{signal}$;
- $\epsilon_k = \emptyset$;
- $\rho = \emptyset$;
- $\mathcal{L}(e_2) = \mathcal{L}(e_4)$.

Connecting kit $Ck_4 = (N, C)$ for a non-SESE region $NR_{(\{en_1, en_2\}, \{ex_1, ex_2\})}$ having two entry and two exit nodes, and contained in a SESE region $R_{(G_{en}, G_{ex})}$ is defined as follows: $N = \{e_7, e_8, e_9, e_{10}, e_{11}, g_7, g_8\}$, while $C = \{(s, g_7), (g_7, e_9), (e_7, g_1), (g_6, g_8), (g_8, e), (g_7, G_{en}), (e_8, en_1), (e_9, en_2), (ex_1, e_{10}), (e_{10}, G_{ex}), (ex_2, e_{11}), (e_{11}, G_{ex}), (G_{ex}, g_8)\}$.

Fig. 3.9(d) shows an example of duration-aware process model obtained by combining a $NR_{(\{en_1, en_2\}, \{ex_1, ex_2\})}$, where $en_1 = ex_1 = t_i$ and $en_2 = ex_2 = t_j$, duration pattern $\phi_{nSESEPred}$ and connecting kit Ck_4 .

So far we have considered predecessors and successors as collapsed sub-processes, that is, as SESE regions. In this special scenario, all the process paths that start from the entry point of the minimal SESE $R_{(en, ex)}$ enclosing the non-SESE region of interest reach also the entry points of the latter one.

However, when predecessors cannot be collapsed into SESE blocks, it is not always true that all process paths including en also include elements of the non-SESE region. In particular, when exclusive gateways in $R_{(en, ex)}$ precede the entry nodes of $NR_{(\{en_1, \dots, en_m\}, \{ex_1, \dots, ex_m\})}$, process paths not leading to $NR_{(\{en_1, \dots, en_m\}, \{ex_1, \dots, ex_m\})}$ may be chosen at run-time. Since in this case there is no need to continue with measuring duration, the duration pattern must be able to allow the process prosecuting its flow without taking any action.

As an example, consider the duration-aware process model shown in Fig. 3.10. The non-SESE region $NR_{(\{Task1, Task2\}, \{Task1, Task2\})}$ has multiple process elements preceding its entry and exit nodes, including exclusive gateways EG1, EG2, and

EG3. This setting is more challenging than those introduced above, as predecessors include alternative paths that do not belong to the non-SESE region but may be chosen before reaching the entry nodes of the region (paths outgoing from EG1, EG2, and EG3 not leading to Alternative Activity 1, Alternative Activity 2, and Alternative Activity 3). Thus, besides events t_START , denoting when the non-SESE regions is enacted, we need to use signal events t_OTHER denoting that a path not belonging to the non-SESE region has been chosen and, thus, the duration pattern is no more needed.

The following guidelines summarize how to place signal events to delimit the non-SESE region to constrain in the discussed scenario:

- a throwing signal event t_START must be placed before every entry node of the non-SESE region;
- a throwing signal event t_OTHER must be placed on every edge outgoing from an exclusive gateway located within the smallest enclosing SESE region and *alternative* to the non-SESE region to constrain;

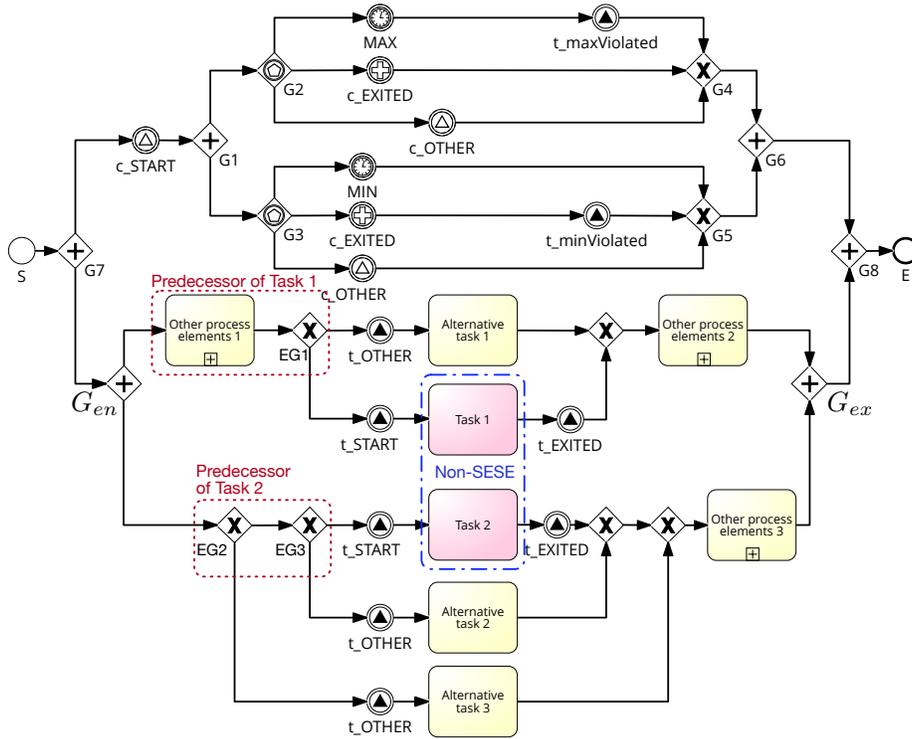


Fig. 3.10: Duration-aware process model showing duration pattern $\phi_{nSESEGen}$ combined with a non-SESE region whose entry nodes have predecessors including exclusive gateways.

– a throwing signal event `t_EXITED` must be placed after every exit node of the non-SESE region.

The placing of events `t_OTHER` is probably the trickiest one, as the concept of “alternative path” is transitive with respect to all nested exclusive gateways. That is, *all* paths that are alternative to the non-SESE region must be marked with a `t_OTHER` event.

As an example, consider again the process of Fig. 3.10. The minimal SESE region $R_{(G_{en}, G_{ex})}$ contains exclusive gateways EG1, EG2, and EG3 preceding the entry nodes of $NR_{(\{Task1, Task2\}, \{Task1, Task2\})}$. Since both gateways EG2 and EG3 have outgoing paths that are alternative to Task 2, a `t_OTHER` event must be placed on each of these paths. Starting from duration pattern $\phi_{nSESEPred}$, two catching signal events `c_OTHER`, corresponding to all events `t_OTHER` are placed in the context of event-based gateways g_2 and g_3 to let tokens flowing ineffectively through the duration pattern every time an alternative path is chosen.

We call $\phi_{nSESEGen}$ this variant of duration pattern $\phi_{nSESEPred}$ as it is the more general duration pattern for handling duration of SESE regions having an arbitrary structure. Parallel gateways belonging to predecessors do not generate such problems, as the all paths outgoing from them are taken.

In this thesis, we do not deal with duration constraints applied to process loops and other repetition structures, as their complexity would require a dedicated approach [141]. Therefore, the proposed duration pattern for addressing non-SESE regions does not apply to non-SESE regions containing parts of a loop beside other process elements. However, ϕ_{simple} can be used to specify the duration of complete loops forming SESE regions.

Thanks to the composition properties of SESE regions [140], the proposed modular design approach allows one to (i) easily nest temporally constrained SESE regions within each other and (ii) specify the duration of any arbitrarily selected non-SESE region in the process. Indeed, in the worst case, the whole process would be the minimal enclosing SESE region for a certain multi entry or exit block.

As a practical example, consider the process for managing intracranial hypertension shown in Fig. 3.11 and Fig. 3.12.

Example 3.5 (Intracranial Hypertension). Intracranial hypertension is a common neurologic complication in critically ill patients caused by a high pressure within the spaces that surround the brain and spinal cord.

The management of intracranial hypertension includes sedation, drainage of cerebrospinal fluid, the administration of glycerol, osmotherapy with mannitol, and hyperventilation [142]. Mannitol should be administered for patients with elevated intracranial hypertension. Oral glycerol can also be prescribed and, during administration, haematology should be systematically checked since it may induce haemolysis.

Let us assume the following duration constraints, outlined in Fig. 3.11. R1 is the SESE region starting with task T1 and ending with task T8 and must last

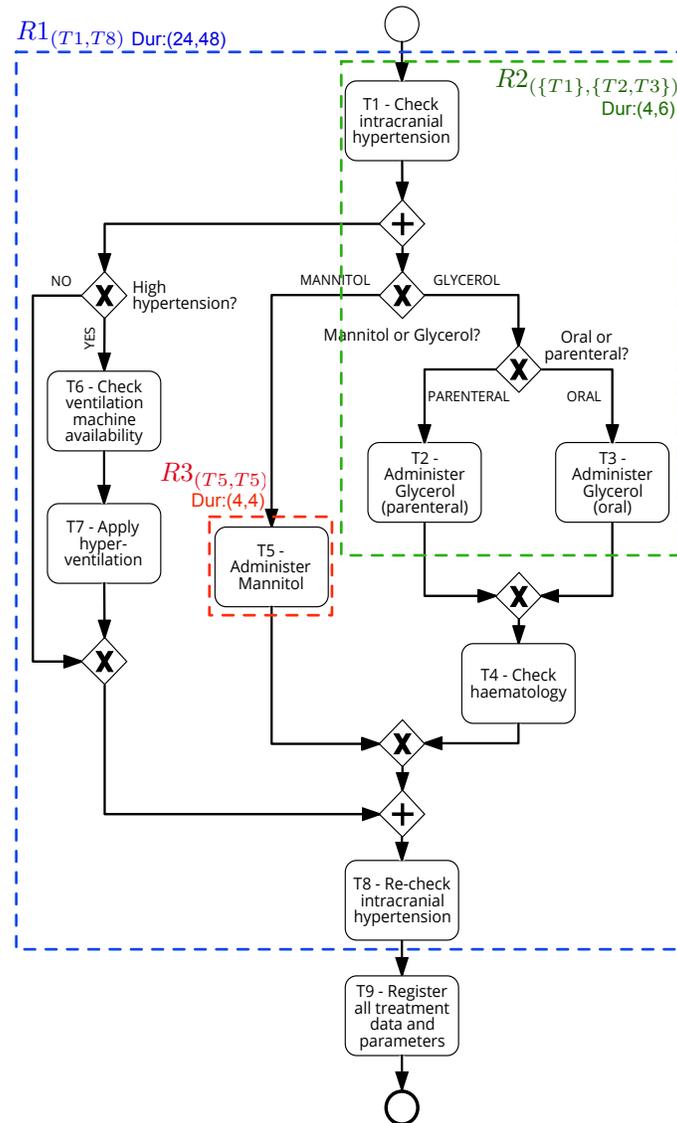


Fig. 3.11: Simple process for the management of intercranial hypertension. SESE region R1 must last 24 to 48 hours; non-SESE region R2 must last 4 to 6 hours; SESE region R3 (task T5) must last exactly 4 hours.

24 to 48 hours. R2 is the non-SESE region starting with T1 and ending with T2 or T3, depending on which administration route is chosen, and must last 4 to 6 hours. Finally, R3 is represented by task T5 which should last exactly 4 hours.

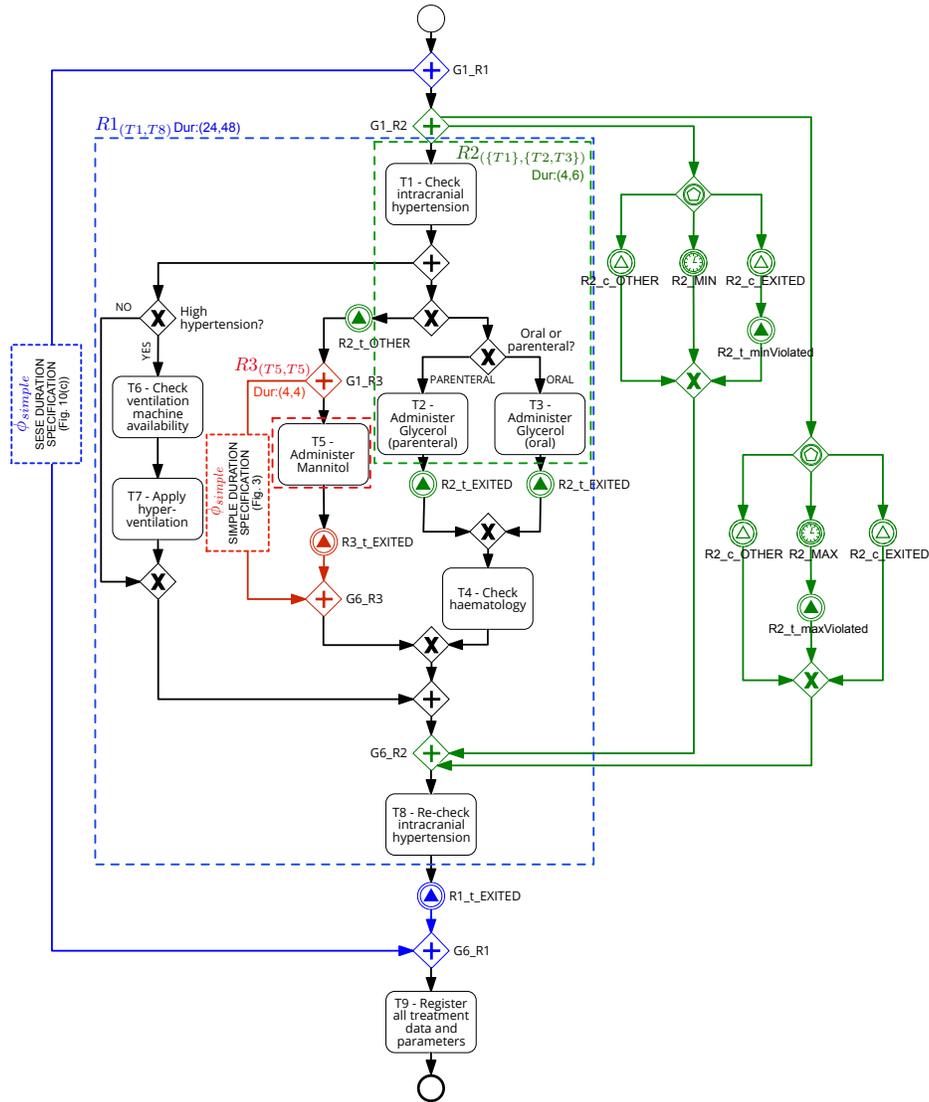


Fig. 3.12: Example of attached duration patterns to process regions R1, R2, and R3 of Fig. 3.11.

In order to specify the duration of R1, R2, and R3 we associate the most appropriate duration pattern to each region, by making sure that these are anchored correctly to the main process and by specializing signal events, so that the management of one process region does not affect the others. This approach facilitates the composition of process blocks, without compromising the generality

of our solution. Indeed, the process model maintains the structural relationships between different regions and, of course, an expert designer can also decide to reduce the number of used signal events by combining them when regions share common starting/ending points.

Fig. 3.12 shows where the duration patterns, and the signal events belonging to the related connecting kit can be positioned on the process of Fig. 3.11 (the constructs for constraining R1 and R3 are only sketched in Fig. 3.12, for readability). The duration patterns added for duration specification are not visible to final users as they may be collapsed in duration-aware subprocesses.

3.4 Specifying Deferred Activity Duration Constraints

In this section, we introduce the modeling of activity duration considering the case of having multiple duration ranges associated to one activity and assuming that the choice of which duration range applies is taken at run-time, that is, after activity initiation.

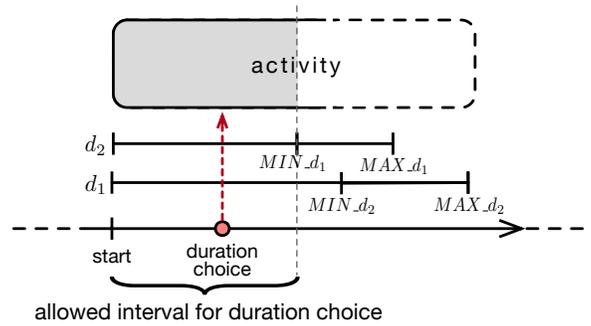


Fig. 3.13: Example showing two different duration ranges d_1 and d_2 associated with one activity. The one that applies is chosen after activity initiation but at a moment in time preceding the smallest between MIN_{d_1} and MIN_{d_2} .

In particular, we deal with a process activity that can be subjected to two or more alternative duration constraints d_1, d_2, \dots, d_n (where $d_i \in [MIN, MAX]$). The choice of which among the constraints applies to the activity is made after its initiation, but prior to the smallest minimum duration bound among those of d_1, d_2, \dots, d_n and cannot change afterwards. Fig. 3.13 shows the relationship between activity initiation, duration constraints, and duration choice. We refer to this kind of constraint as *deferred duration*, meaning that the choice of which duration range applies is deferred with respect to activity initiation.

As a motivating example, let us consider the following clinical setting, typical of antibiotic therapies.

Example 3.6 (Antibiotic therapy for bloodstream infections). *Staphylococcus aureus* is a common and virulent life-threatening bacterium causing bloodstream infections. The clinical management of staphylococcus aureus requires antibiotic treatment, which is administered taking into consideration the patient’s allergies and bacteria resistance patterns.

Typically, the therapy can last either 14 days or 4-6 weeks, depending on clinical judgement [143]. A 14-days therapy is administered only if:

- (C1) fever disappears within 72 hours after treatment initiation and
- (C2) blood cultures are negative.

However, the results of blood analyses are obtained 3 or 4 days after the beginning of the therapy. This means, that the choice of the required duration of the therapy is taken after empirical antibiotic administration has started (i.e., 3-4 days after therapy initiation), but earlier than the lowest minimum duration constraint (i.e., 14 days).

All the patients that do not observe criteria (C1) and (C2) must receive 4-6 weeks of therapy based on the extent of the infection.

Example 3.6 motivates the modeling of this novel kind of duration, explained in the remainder of this section. It is worth noticing that we do not require duration ranges to be disjoint, as only one of them can be chosen at a time.

Consider the process model depicted in Figure 3.14. For simplicity, we associate the Task depicted at the bottom with two desired duration intervals $d_1 \in [\text{MIN_D1}, \text{MAX_D1}]$ and $d_2 \in [\text{MIN_D2}, \text{MAX_D2}]$, but the same solution can be generalized to deal with more than two duration ranges.

The intuition behind the behavior of this process relies on the assumption that among the four duration extremes MIN_D1, MAX_D1, MIN_D2 and MAX_D2 there is always a smallest one, regardless of how the two duration ranges are related to each other (If MIN_D1= MIN_D2, then we can arbitrarily select one of them). This lowest minimum duration value is needed for ordering duration ranges based on the value of their minimum duration since it is crucial to know the latest time-instant at which Task duration must be chosen.

Without loss of generality, let us suppose that $d1$ is the time length having the lowest minimum duration, that is, MIN_D1 in the process model of Fig. 3.14. The behavior of the process can be explained as follows. After start event S is triggered, parallel gateway G1 splits the flow into five branches, one directed towards Task and the others ones leading to event-based gateways G2_MIN, G2_MAX, G3_MIN and G3_MAX.

From this point on, the process behavior depends on the moment of Task completion, on which duration range is chosen, and on the kind (if any) of violated duration (i.e., minimum or maximum).

Duration choice is represented by conditional events which are triggered by the environment whenever a certain condition is true [11]. In general, anything can be a condition and conditions are independent of processes. Conditional events exist only of type catching and are triggered when a data-based condition

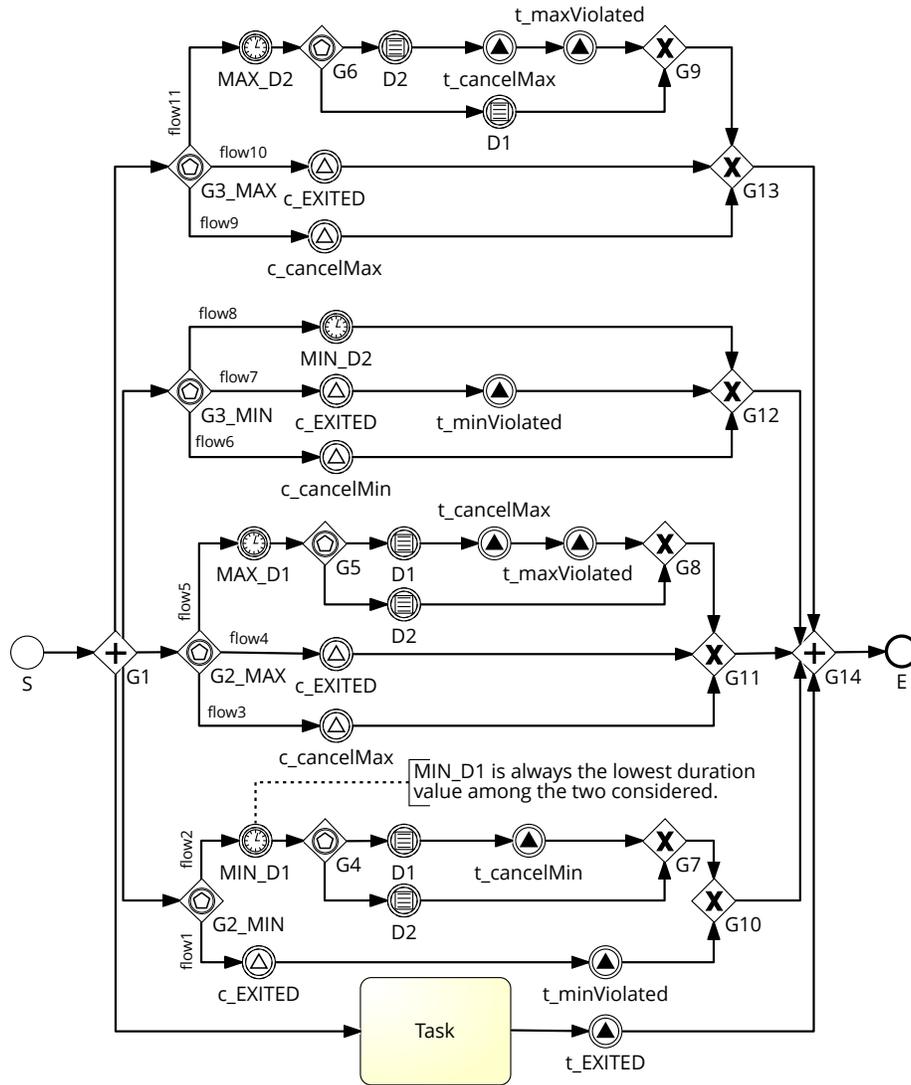


Fig. 3.14: Duration-aware process model for specifying to two different duration ranges D1 and D2 associated to a Task. The choice of which one applies is deferred after task initiation.

evaluates to true [101]. In this thesis, we assume that when a conditional event is enabled by a token, the process checks whether the associated condition is true and, if so, the event is triggered. Otherwise, the token will remain on the event (i.e., the event will remain enabled) until the associated condition becomes true. In particular, if d_1 is the chosen duration, then the condition associated to

conditional event D1 is true, otherwise the condition associated to conditional event D2 holds.

In the described scenario, minimum duration is violated when:

- (a) Task completes before its desired duration is chosen;
 - (b) Task ends before the lowest minimum value set for duration, that is MIN_D1, when the chosen duration is d_1 ;
 - (c) Task completes before MIN_D2 when the chosen duration is d_2 .
- (a) If Task completes earlier than knowing which is the chosen duration range, signal event t.EXITED is thrown to be caught by all the four corresponding events c.EXITED located on flow1, flow4, flow7, and flow10. Then, the violation of minimum duration constraint is signaled by both events t_minViolated on flow1 and flow7 before the whole process can complete. Since duration range has not yet been chosen, all minimum durations are violated.

(b) If Task completes earlier than MIN_D1, being MIN_D1 the smallest admissible duration value, a violation has to be signaled regardless of which is the chosen duration range. In this case, the process behaves as discussed in (a), as signal event t.EXITED is caught by all the corresponding listening events.

(c) If Task completes earlier than MIN_D2, a violation must be signaled only if the chosen duration range is d_2 . To this end, event-based gateway G4, enacted right after event MIN_D1, differentiates the process behavior with respect to which duration range has been chosen. If Task duration is d_1 , no minimum violation is signaled: signal event t_cancelMin is thrown to trigger the corresponding event c_cancelMin on flow6, thus discarding timer event MIN_D2. If Task duration is d_2 , then a minimum duration violation is observed. In particular, on flow2 timer event MIN_D1 has already fired as $\text{MIN_D1} \leq \text{MIN_D2}$. In the configuration of event-based gateway G4, conditional event D2 is triggered to let the process flow proceeding until exclusive gateway G7 without further signaling. The detection of the activity early completion is handled by signal events c.EXITED and t_minViolated located within flow7. Similarly, it is worth noticing that in case $d_1 < d_2$, also event D2 in the scope of G5 is triggered, as there is no need to check duration MAX_D1.

For all the discussed cases (a)–(c), the process flows branching from G2.MAX and G3.MAX have the following behavior. Regardless of which is the maximum duration considered, if the corresponding timer event has not yet been triggered, then either signal events c.EXITED on flow4 and flow10 or signal events c_cancelMax on flow3 and flow9 will be triggered letting the flow proceed without any further signaling.

When considering maximum duration violation the process behaves as follows. Despite durations d_1 and d_2 are ordered according to the lowest value for minimum duration, timer event MAX_D1 may be assigned a duration value smaller, equal, or greater than the one of timer event MAX_D2.

If the chosen duration is d_1 the condition associated to conditional event D1 is true. When maximum duration is violated, signal event t_cancelMax following event-based gateway G5 and event D1 is thrown to “cancel” event handlers for d_2 . In particular, if $\text{MAX_D1} < \text{MAX_D2}$, the triggered signal is caught by signal

event `c_cancelMax` on `flow9`. Then, event `t_maxViolated` on `flow5` indicates that the maximum duration has been violated. If $MAX_D1 > MAX_D2$, then signal `c_cancelMax` is never caught as `flow11` has been chosen.

Besides the discussed cases, we consider the possibility that either (i) $MIN_D1 = MIN_D2$, or (ii) $MAX_D1 = MAX_D2$. We prove that the model behaves soundly for the mentioned cases as follows.

- (i) If $MIN_D1 = MIN_D2$, the lowest minimum can be chosen arbitrarily. A minimum duration violation will involve both d_1 and d_2 , as for the previously discussed case (b).
- (ii) If $MAX_D1 = MAX_D2$, signal event `t_cancelMax`, thrown after the timer event representing the chosen maximum duration limit, is never caught. In detail, if we choose d_1 as the preferred Task duration, even if timer event MAX_D2 is triggered concurrently to MAX_D1 , conditional events D1 and D2 in the scope of event-based gateway G4 ensure that signal `t_maxViolated` is triggered only once. Specular behavior is expected if d_2 is chosen.

The process depicted in Fig. 3.14 is already the complete duration-aware process model obtained by combining the activity a to be constrained, a duration pattern for deferred duration, and a connecting kit. Below, we formalize the structure of duration pattern $\phi_{deferred}$ considering two different duration intervals associated to activity a .

$$\phi_{deferred} = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$$

where:

- $N = \{A \cup G \cup E\}$ is the set of flow nodes partitioned in:
 - $A = \emptyset$; $G = \{g_1, g_{2min}, g_{2max}, g_{3min}, g_{3max}, g_4, g_5, g_6, g_7, g_8, g_9, g_{10}, g_{11}, g_{12}, g_{13}, g_{14}\}$; $E = \{E_{start} \cup E_{int} \cup E_{end}\}$ where $E_{start} = \{s\}$, $E_{int} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}, e_{20}, e_{21}, e_{22}, e_{23}, e_{24}\}$, and $E_{end} = \{e\}$;
- $DN = \emptyset$;
- $C = \{(s, g_1), (g_1, a), (g_1, g_{2min}), (g_1, g_{2max}), (g_1, g_{3min}), (g_1, g_{3max}), (g_{3max}, e_1), (g_{3max}, e_2), (g_{3max}, e_3), (g_{3min}, e_4), (g_{3min}, e_5), (g_{3min}, e_6), (g_{2max}, e_7), (g_{2max}, e_8), (g_{2max}, e_9), (g_{2min}, e_{10}), (g_{2min}, e_{11}), (e_1, g_6), (g_6, e_{12}), (g_6, e_{15}), (e_{12}, e_{13}), (e_{13}, e_{14}), (e_{14}, g_9), (e_{15}, g_9), (g_9, g_{13}), (e_2, g_{13}), (e_3, g_{13}), (e_4, g_{12}), (e_5, e_{16}), (e_{16}, g_{12}), (e_6, g_{12}), (e_7, g_5), (g_5, e_{17}), (g_5, e_{20}), (e_{17}, e_{18}), (e_{18}, e_{19}), (e_{19}, g_8), (g_8, g_{11}), (g_{20}, g_8), (e_8, g_{11}), (e_9, g_{11}), (e_{10}, g_4), (g_4, e_{21}), (g_4, e_{23}), (e_{21}, e_{22}), (e_{22}, g_7), (e_{23}, g_7), (g_7, g_{10}), (e_{11}, g_{10}), (g_{13}, g_{14}), (g_{12}, g_{14}), (g_{11}, g_{14}), (g_{10}, g_{14}), (g_{14}, e)\}$;
- $TA = \emptyset$;
- $F = \emptyset$;
- $T = \emptyset$;
- $R = \emptyset$;
- $\alpha_k = \emptyset$;
- $\alpha_t = \emptyset$;
- $\beta = \emptyset$;

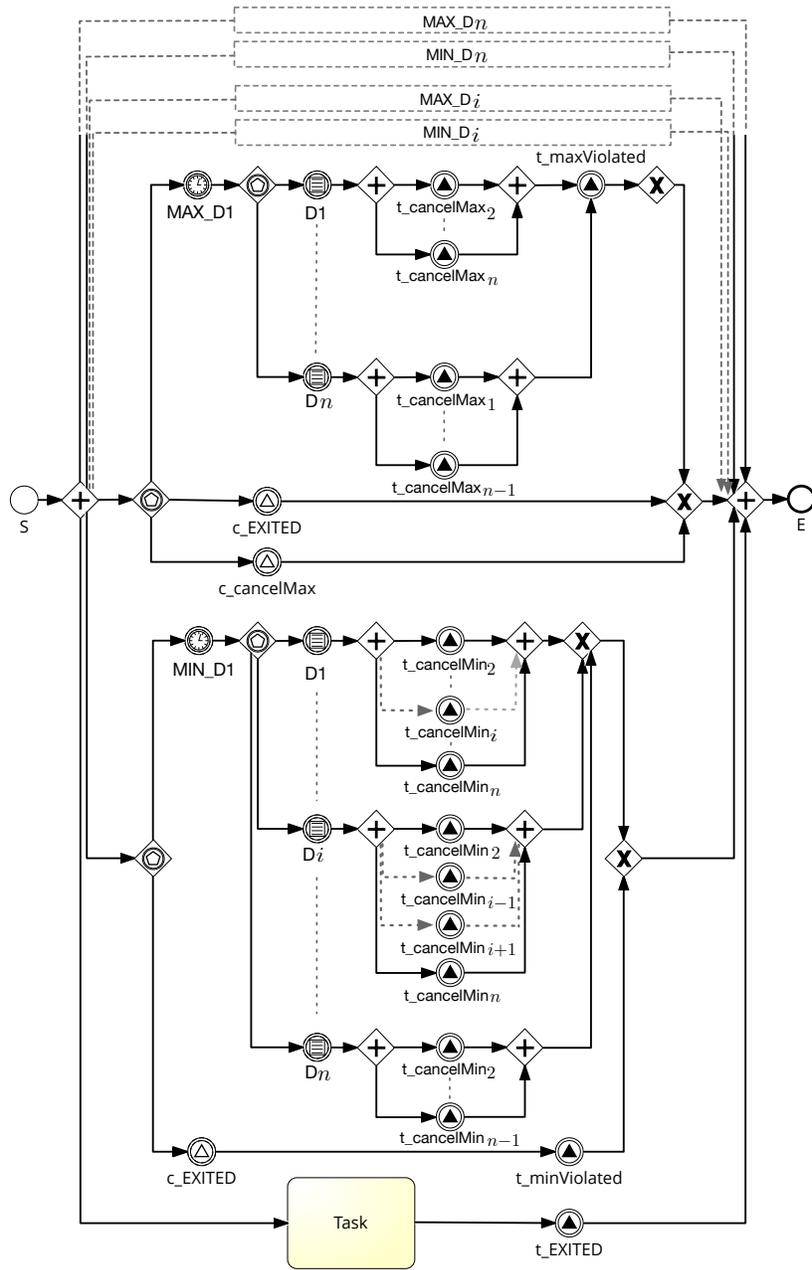


Fig. 3.15: Complete process combining control structures for capturing deferred duration which is chosen among an arbitrary number of duration ranges.

- $\gamma_r(\{g_1, g_{2min}, g_{2max}, g_{3min}, g_{3max}, g_4, g_5, g_6\}) = \textit{split}$, $\gamma_r(\{g_7, g_8, g_9, g_{10}, g_{11}, g_{12}, g_{13}, g_{14}\}) = \textit{merge}$;
- $\gamma_{ty}(\{g_1, g_{14}\}) = \textit{parallel}$, $\gamma_{ty}(\{g_{2min}, g_{2max}, g_{3min}, g_{3max}, g_4, g_5, g_6\}) = \textit{event-based}$, $\gamma_{ty}(\{g_7, g_8, g_9, g_{10}, g_{11}, g_{12}, g_{13}\}) = \textit{exclusive}$;
- $\epsilon_{tr}(\{s, e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{15}, e_{17}, e_{20}, e_{21}, e_{23}\}) = \textit{catching}$, $\epsilon_{tr}(\{e_{13}, e_{14}, e_{16}, e_{18}, e_{19}, e_{22}, e_{24}, e\}) = \textit{throwing}$;
- $\epsilon_{tr}(\{s, e\}) = \textit{none}$, $\epsilon_{tr}(\{e_1, e_4, e_7, e_{10}\}) = \textit{timer}$, $\epsilon_{ty}(\{e_2, e_3, e_5, e_6, e_8, e_9, e_{11}, e_{13}, e_{14}, e_{16}, e_{18}, e_{19}, e_{22}, e_{24}\}) = \textit{signal}$, $\epsilon_{tr}(\{e_{12}, e_{15}, e_{17}, e_{20}, e_{21}, e_{23}\}) = \textit{conditional}$;
- $\epsilon_k = \emptyset$;
- $\rho = \emptyset$;
- $\mathcal{L}(e_2) = \mathcal{L}(e_5) = \mathcal{L}(e_8) = \mathcal{L}(e_{11})$, $\mathcal{L}(e_{15}) = \mathcal{L}(e_{17}) = \mathcal{L}(e_{21})$, $\mathcal{L}(e_{12}) = \mathcal{L}(e_{20}) = \mathcal{L}(e_{23})$, $\mathcal{L}(e_3) = \mathcal{L}(e_9)$, $\mathcal{L}(e_{13}) = \mathcal{L}(e_{18})$, $\mathcal{L}(e_4) = \mathcal{L}(e_{19})$, and $\mathcal{L}(e_{16}) = \mathcal{L}(e_{14})$.

Connecting kit $Ck_5 = (N, C)$ where $N = \{e_{25}\}$ and $C = \{(g_1, a), (a, e_{25}), (e_{25}, g_{14})\}$. Event e_{25} is a throwing signal event that triggers all three events e_2 , e_5 , e_8 , and e_{11} of $\phi_{deferred}$ whenever they are actively listening.

In the duration-aware process model of Fig. 3.14 activity a is represented as an abstract task, i.e., $\alpha_k(a) = \textit{task}$, $\alpha_t(a) = \textit{abstract}$, and $\mathcal{L}(a) = \textit{Task}$, without loss of generality. The flow nodes of duration pattern $\phi_{deferred}$ are labeled as follows. $\mathcal{L}(e_1) = \textit{MAX_D2}$, $\mathcal{L}(e_2) = \textit{c_EXITED}$, $\mathcal{L}(e_3) = \textit{c_cancelMax}$, $\mathcal{L}(e_4) = \textit{MIN_D2}$, $\mathcal{L}(e_5) = \textit{c_EXITED}$, $\mathcal{L}(e_6) = \textit{c_cancelMin}$, $\mathcal{L}(e_7) = \textit{MAX_D1}$, $\mathcal{L}(e_8) = \textit{c_EXITED}$, $\mathcal{L}(e_9) = \textit{c_cancelMax}$, $\mathcal{L}(e_{10}) = \textit{MIN_D1}$, $\mathcal{L}(e_{11}) = \textit{c_EXITED}$, $\mathcal{L}(e_{12}) = \textit{D2}$, $\mathcal{L}(e_{13}) = \textit{t_cancelMax}$, $\mathcal{L}(e_{14}) = \textit{t_maxViolated}$, $\mathcal{L}(e_{15}) = \textit{D1}$, $\mathcal{L}(e_{16}) = \textit{t_minViolated}$, $\mathcal{L}(e_{17}) = \textit{D1}$, $\mathcal{L}(e_{18}) = \textit{t_cancelMax}$, $\mathcal{L}(e_{19}) = \textit{t_maxViolated}$, $\mathcal{L}(e_{20}) = \textit{D2}$, $\mathcal{L}(e_{21}) = \textit{D1}$, $\mathcal{L}(e_{22}) = \textit{t_cancelMin}$, $\mathcal{L}(e_{23}) = \textit{D2}$, $\mathcal{L}(e_{24}) = \textit{t_minViolated}$, while gateways $g_1 \dots g_{14}$ are assigned labels $G1 \dots G14$. Event e_{25} of connecting kit Ck_5 is such that $\mathcal{L}(e_{25}) = \textit{t_EXITED}$.

When dealing with more than two duration ranges, signal events labeled as $\textit{t_cancelMin}$, $\textit{c_cancelMin}$, $\textit{t_cancelMax}$, and $\textit{c_cancelMax}$ in Fig. 3.14 must be specialized in $\textit{t_cancelMin}_i$, $\textit{c_cancelMin}_i$, $\textit{t_cancelMax}_i$, and $\textit{c_cancelMax}_i$ in order to let the process flowing through the branches designed to represent duration ranges D_i that are not chosen by the activity. Besides, one conditional event D_i must be added for each possibly chosen duration d_i . Fig. 3.15 sketches how duration pattern $\phi_{deferred}$ can be extended with specialized conditional and signal events to handle more than two duration ranges.

3.5 Specifying Shifted Duration Constraints

In this section, we introduce patterns for specifying shifted duration constraints that extend and complete the preliminary proposal described in [53].

Shifting the activity duration means that duration is measured only after a certain condition is met, i.e., the duration of the activity is evaluated starting

from a particular moment in time that is *shifted* forward in time with respect to activity initiation.

Example 3.7 (Antibiotic treatment of pneumococcal pneumonia). As an example, let us consider the treatment of uncomplicated *pneumococcal pneumonia*, a common lung infection leading to hospitalization. Antibiotic therapy must be initiated immediately after the onset of the infection. Then, depending on the extent of the infection and the patient clinical response, antibiotic administration is continued for 5–7 days after defervescence. The overall duration of therapy should not exceed 10 days. If the patient does not defervesce within 3–5 days, antibiotic susceptibility must be reviewed.

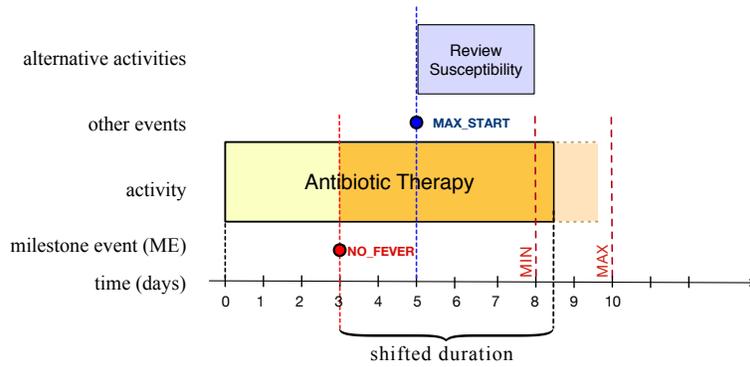


Fig. 3.16: Shifted duration exemplified with respect to activity Antibiotic Therapy to treat pneumococcal pneumonia.

The concept of shifted duration for the introduced example is shown in Fig. 3.16. Antibiotic Therapy begins, but its duration $d \in [5, 7]$ days is measured starting from *milestone event (ME)* NO_FEVER. If the latter does not occur within 5 days (the maximum time allowed for defervescence), event MAX_START triggers alternative activity Review Susceptibility.

To represent shifted duration of an activity, we designed duration pattern $\phi_{shifted} = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$ whose structure is formalized as follows.

- $N = \{A \cup G \cup E\}$ is the set of flow nodes, where:
 $A = \{a_1\}$; $G = \{g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8, g_9, g_{10}\}$; $E = \{E_{start} \cup E_{int} \cup E_{end}\}$
 where $E_{start} = \{s\}$, $E_{int} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}\}$, and $E_{end} = \{e\}$;
- $DN = \emptyset$;
- $C = \{(s, g_1), (g_1, g_2), (g_2, e_1), (g_2, e_2), (g_2, e_3), (e_3, e_4), (e_4, g_9), (e_1, a_1), (a_1, g_9), (e_2, g_3), (g_3, g_4), (g_3, g_5), (g_4, e_5), (g_4, e_6), (e_6, g_6), (g_5, e_7), (g_5, e_8),$

- $(e_5, e_9), (e_9, g_6), (g_6, g_8), (e_8, e_{10}), (e_7, g_7), (e_{10}, g_7), (g_7, g_8), (g_8, g_9), (g_9, g_{10}), (g_{10}, e)$ is the set of control flow edges;
- $TA = \emptyset$;
- $F = \emptyset$;
- $T = \emptyset$;
- $R = \emptyset$;
- $\alpha_k(a_1) = \textit{subprocess}$;
- $\alpha_t(a_1) = \textit{abstract}$;
- $\beta = \emptyset$;
- $\gamma_r(\{g_1, g_2, g_3, g_4, g_5\}) = \textit{split}, \gamma_r(\{g_6, g_7, g_8, g_9, g_{10}\}) = \textit{merge}$;
- $\gamma_{ty}(\{g_1, g_3, g_8, g_{10}\}) = \textit{parallel}, \gamma_{ty}(\{g_6, g_7, g_9\}) = \textit{exclusive},$ and $\gamma_{ty}(\{g_2, g_4, g_5\}) = \textit{event-based}$;
- $\epsilon_{tr}(\{s, e_1, e_2, e_3, e_5, e_6, e_7, e_8\}) = \textit{catching}, \epsilon_{tr}(\{e_4, e_9, e_{10}, e\}) = \textit{throwing}$;
- $\epsilon_{ty}(\{s, e\}) = \textit{none}, \epsilon_{ty}(\{e_1, e_5, e_7\}) = \textit{timer}, \epsilon_{ty}(\{e_3, e_4, e_6, e_8, e_9, e_{10}\}) = \textit{signal}, \epsilon_{ty}(\{e_2\}) = \textit{conditional}$;
- $\epsilon_k = \emptyset$;
- $\rho = \emptyset$;
- $\mathcal{L}(e_3) = \mathcal{L}(e_6) = \mathcal{L}(e_8),$ and $\mathcal{L}(e_4) = \mathcal{L}(e_{10}).$

Connecting kit $Ck_6 = (N, C)$ where $N = \{e_{11}\}$ and $C = \{(g_1, a), (a, e_{11}), (e_{11}, g_{10})\}$. Event e_{11} is a throwing signal event that triggers all three events $e_3, e_6,$ and e_8 of $\phi_{shifted}$ whenever they are actively listening.

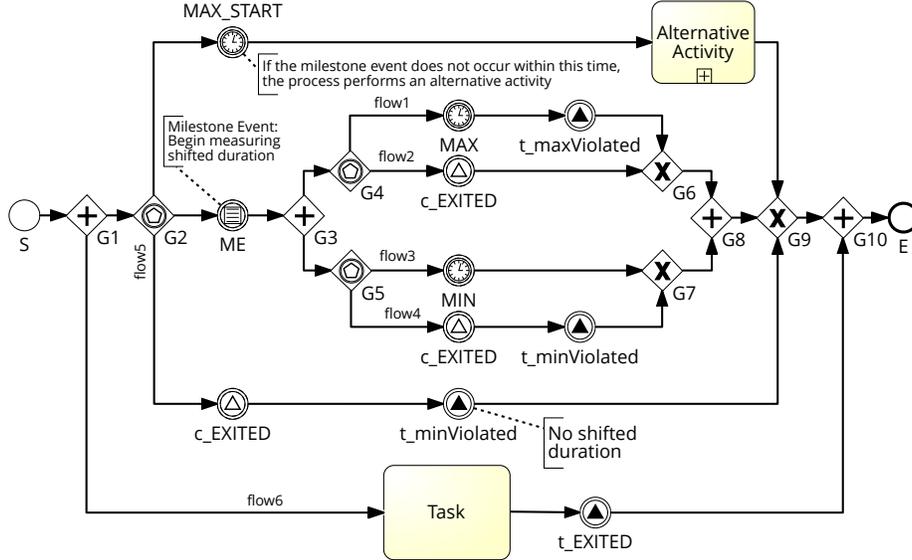


Fig. 3.17: Duration-aware process model for specifying the shifted duration of a Task.

Fig. 3.17 depicts the complete duration-aware process model for specifying the shifted duration of activity a , obtained by combining activity a to be constrained with duration pattern $\phi_{shifted}$ through connecting kit Ck_6 .

Without loss of generality, activity a is represented as an abstract task, i.e., $\alpha_k(a) = task$, $\alpha_t(a) = abstract$, and $\mathcal{L}(a) = Task$.

The flow nodes elements of duration pattern $\phi_{shifted}$ are labeled as follows: $\mathcal{L}(a_1) = \text{Alternative Activity}$, $\mathcal{L}(s) = S$, $\mathcal{L}(e_1) = \text{MAX_START}$, $\mathcal{L}(e_2) = \text{ME}$, $\mathcal{L}(e_3) = \text{c_EXITED}$, $\mathcal{L}(e_4) = \text{t_minViolated}$, $\mathcal{L}(e_5) = \text{MAX}$, $\mathcal{L}(e_6) = \text{c_EXITED}$, $\mathcal{L}(e_7) = \text{MIN}$, $\mathcal{L}(e_8) = \text{c_EXITED}$, $\mathcal{L}(e_9) = \text{t_maxViolated}$, $\mathcal{L}(e_{10}) = \text{t_minViolated}$, $\mathcal{L}(e) = E$, $\mathcal{L}(g_1) = G1$, $\mathcal{L}(g_2) = G2$, $\mathcal{L}(g_3) = G3$, $\mathcal{L}(g_4) = G4$, $\mathcal{L}(g_5) = G5$, $\mathcal{L}(g_6) = G6$, $\mathcal{L}(g_7) = G7$, $\mathcal{L}(g_8) = G8$, $\mathcal{L}(g_9) = G9$, $\mathcal{L}(g_{10}) = G10$. Finally, event e_{11} of connecting kit Ck_6 is such that $\mathcal{L}(e_{11}) = \text{t_EXITED}$.

Given that a is the activity whose shifted duration must be specified, it is worth noticing that $\phi_{shifted}$ is attached to a in the same way ϕ_{simple} is attached to a to measure simple duration. Indeed, connecting kits Ck_1 and Ck_6 have the same number and kinds of elements.

The complete duration-aware process model, shown in Fig. 3.17 behaves as follows. Once the process is started, the flow is split by G1 into two branches: flow6 is directed towards Task, which can begin its execution, while on the other branch event-based gateway G2 is enabled. This realizes a race condition between the occurrence of timer event MAX_START, which ensures that something alternative is done if the milestone event does not occur, milestone event ME, and signal event c.EXITED, which handles the case in which Task ends before any of the other two events has occurred. In this latter “borderline scenario”, minimum violation occurs, as captured by the follow signal event t_minViolated, but the measurement of shifted duration has not yet started. As mentioned in Sect. 3.4, conditional event ME can fire when it is enabled by the incoming token and the associated condition is true.

The detection of shifted duration violations is realized by using a simple signal-based communication pattern. When Task completes its execution, signal event t.EXITED is broadcast to be caught by any of the corresponding c.EXITED events that are active at the moment of broadcasting. Based on when Task ends, any of the following mutually exclusive scenarios can occur.

Minimum shifted duration is violated if Task completes earlier than MIN. In this case, both signals c.EXITED are triggered, whereas the other process branches outgoing of event-based gateways G4 and G5 are withdrawn. Then, signal event t_minViolated, is used to capture the violation. Finally, once synchronization has occurred at G10, the process can conclude.

If Task ends anytime between MIN and MAX no violation occurs and signal event t.EXITED is caught only by the corresponding c.EXITED on flow2, as timer event MIN has already fired.

Maximum shifted duration is violated whenever Task execution lasts longer than MAX. In this case, right after MAX, signal t_maxViolated is broadcast to detect that maximum shifted duration has been violated. Trivially, as timer

event MIN had also been triggered before MAX, the process can complete once synchronization has occurred at G10.

In [53], we also propose a pattern for specifying a shifted duration with reset. Indeed, ME captures the beginning of a certain condition that must hold for the whole period of shifted duration. When such condition is not more valid, we must capture the change and reset shifted duration.

For example, let us consider patient discharge from hospital (cf. Example 3.3). Discharge criteria require patients to have been afebrile for at least 24 hours to be safely dismissed [53]. This means that, whenever fever (re-)appears the patient must wait for temperature to lower within normal ranges and, from that moment, stay in hospital for 24 fever-free additional hours. To specify the shifted duration of activity “hospitalization”, which should be of at least 24 hours after “fever disappears”, our milestone event we should consider reset condition “fever comes back”, which requires physicians to wait until fever disappears again before re-counting 24 hours.

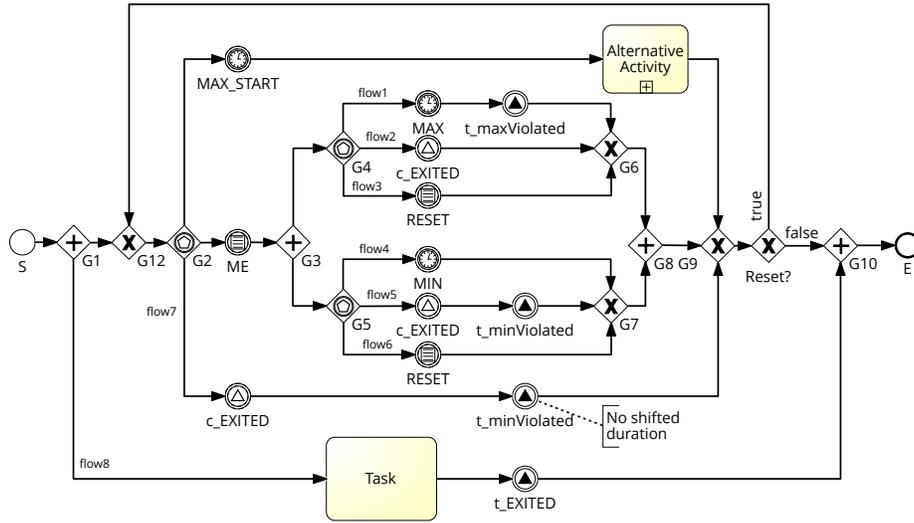


Fig. 3.18: Process model for specifying resettable shifted duration of a task.

Formally, the structure of duration pattern $\phi_{shiftRes} = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$ for specifying shifted duration with reset is as follows.

- $N = \{A \cup G \cup E\}$ is the set of flow nodes, where:
 $A = \{a_1\}$; $G = \{g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8, g_9, g_{10}, g_{11}, g_{12}\}$; $E = \{E_{start} \cup E_{int} \cup E_{end}\}$ where $E_{start} = \{s\}$, $E_{int} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\}$, and $E_{end} = \{e\}$;
- $DN = \emptyset$;

- $C = \{(s, g_1), (g_1, g_{12}), (g_{12}, g_2), (g_2, e_1), (g_2, e_2), (g_2, e_3), (e_3, e_4), (e_4, g_9), (e_1, a_1), (a_1, g_9), (e_2, g_3), (g_3, g_4), (g_3, g_5), (g_4, e_5), (g_4, e_6), (g_4, e_7), (e_6, g_6), (e_7, g_6), (g_5, e_8), (g_5, e_9), (g_5, e_{10}), (e_5, e_{11}), (e_{11}, g_6), (g_6, g_8), (e_9, e_{12}), (e_8, g_7), (e_{12}, g_7), (e_{10}, g_7), (g_7, g_8), (g_8, g_9), (g_9, g_{11}), (g_{11}, g_{10}), (g_{11}, g_{12}), (g_{10}, e)\}$ is the set of control flow edges;
- $TA = \emptyset$;
- $F = \emptyset$;
- $T = \emptyset$;
- $R = \emptyset$;
- $\alpha_k(a_1) = \textit{subprocess}$;
- $\alpha_t(a_1) = \emptyset$;
- $\beta = \emptyset$;
- $\gamma_r(\{g_1, g_2, g_3, g_4, g_5, g_{11}\}) = \textit{split}$, $\gamma_r(\{g_6, g_7, g_8, g_9, g_{10}, g_{12}\}) = \textit{merge}$;
- $\gamma_{ty}(\{g_1, g_3, g_8, g_{10}\}) = \textit{parallel}$, $\gamma_{ty}(\{g_6, g_7, g_9, g_{11}, g_{12}\}) = \textit{exclusive}$, and $\gamma_{ty}(\{g_2, g_4, g_5\}) = \textit{event-based}$;
- $\epsilon_{tr}(\{s, e_1, e_2, e_3, e_5, e_6, e_7, e_8, e_9, e_{10}\}) = \textit{catching}$, $\epsilon_{tr}(\{e_4, e_{11}, e_{12}, e\}) = \textit{throwing}$;
- $\epsilon_{ty}(\{s, e\}) = \textit{none}$, $\epsilon_{ty}(\{e_1, e_5, e_8\}) = \textit{timer}$, $\epsilon_{ty}(\{e_3, e_4, e_6, e_9, e_{11}, e_{12}\}) = \textit{signal}$, $\epsilon_{ty}(\{e_2, e_7, e_{10}\}) = \textit{conditional}$;
- $\epsilon_k = \emptyset$;
- $\rho = \emptyset$;
- $\mathcal{L}(e_3) = \mathcal{L}(e_6) = \mathcal{L}(e_9)$, $\mathcal{L}(e_4) = \mathcal{L}(e_{12})$, and $\mathcal{L}(e_7) = \mathcal{L}(e_{10})$.

Connecting kit Ck_6 can be reused to connect $\phi_{\textit{shiftRes}}$ to activity a .

Fig. 3.18 shows the complete duration-aware process model which combines activity a , duration pattern $\phi_{\textit{shiftRes}}$ and connecting kit Ck_6 .

Without loss of generality, activity a is represented as an abstract task, i.e., $\alpha_k(a) = \textit{task}$, $\alpha_t(a) = \textit{abstract}$, and $\mathcal{L}(a) = \textit{Task}$. The flow nodes of duration pattern $\phi_{\textit{shiftRes}}$ are labeled as follows. $\mathcal{L}(a_1) = \textit{Alternative Activity}$, $\mathcal{L}(s) = \textit{S}$, $\mathcal{L}(e_1) = \textit{MAX_START}$, $\mathcal{L}(e_2) = \textit{ME}$, $\mathcal{L}(e_3) = \textit{c_EXITED}$, $\mathcal{L}(e_4) = \textit{t_minViolated}$, $\mathcal{L}(e_5) = \textit{MAX}$, $\mathcal{L}(e_6) = \textit{c_EXITED}$, $\mathcal{L}(e_7) = \textit{RESET}$, $\mathcal{L}(e_8) = \textit{MIN}$, $\mathcal{L}(e_9) = \textit{c_EXITED}$, $\mathcal{L}(e_{10}) = \textit{RESET}$, $\mathcal{L}(e_{11}) = \textit{t_maxViolated}$, $\mathcal{L}(e_{12}) = \textit{t_minViolated}$, $\mathcal{L}(e) = \textit{E}$, $\mathcal{L}(g_1) = \textit{G1}$, $\mathcal{L}(g_2) = \textit{G2}$, $\mathcal{L}(g_3) = \textit{G3}$, $\mathcal{L}(g_4) = \textit{G4}$, $\mathcal{L}(g_5) = \textit{G5}$, $\mathcal{L}(g_6) = \textit{G6}$, $\mathcal{L}(g_7) = \textit{G7}$, $\mathcal{L}(g_8) = \textit{G8}$, $\mathcal{L}(g_9) = \textit{G9}$, $\mathcal{L}(g_{10}) = \textit{G10}$, $\mathcal{L}(g_{11}) = \textit{Reset?}$, and $\mathcal{L}(g_{12}) = \textit{G12}$. Event e_{11} of connecting kit Ck_6 is again labeled as $\mathcal{L}(e_{11}) = \textit{t_EXITED}$.

Basically, compared to $\phi_{\textit{shifted}}$, duration pattern $\phi_{\textit{shiftRes}}$ includes another conditional event **RESET**, which leads back to **ME** in order wait for a new occurrence of the milestone event. We can assume that the condition associated to event **RESET** is set to *false* when the process begins, and can change during execution, starting after the occurrence of milestone event **ME**.

3.6 Detecting and Managing Duration Violations

Whereas a first step towards the management of minimum and maximum duration constraints can be achieved by detecting constraint violations and by informing the process engine or the activity performer about potential problems/delays caused by temporal violations, appropriate constraint management deals with the specification of (temporal) exception handlers.

However, temporal exception management relies on the semantics and the nature of the violated constraints, and are strongly dependent upon the kind of activity being performed.

For instance, when a maximum duration constraint is violated, either sudden termination can be imposed or a side procedure can be used to expedite activity completion. Indeed, some activities need additional time to be correctly finished and they cannot be suddenly interrupted even if they violate maximum duration. As an example, we can think of a surgical intervention which is taking longer than planned: obviously, the surgeon cannot abandon the operating room, but additional workforce may be called to speed up the intervention.

Depending on their potential to interrupt activity execution, we distinguish between weak and strong maximum duration constraints. A *weak* duration constraint does not cause immediate activity interruption, but additional side activities can be performed with the goal of addressing constraint violation. Conversely, a *strong* duration constraint causes the sudden interruption of the on-going activity. For example, whenever drug therapy causes an unexpected allergic reaction, this must immediately stopped.

The introduced definition of constraint strength does not apply to minimum duration constraints, as there is no need for activity interruption in such context.

The process models proposed in the remainder of the section, are designed to manage duration violations at a high level of abstraction, since the nature of the repairing actions is highly dependent on the kind of task being performed.

To this end, we identify possible general and common actions that duration violation handlers are entitled to perform (see Table 3.4). Besides, we can also consider the following two extreme actions that hold for every constraint, that is, “Do nothing” (i.e., the handler simply alerts the activity performer that something went differently from expected) and “Terminate process execution”.

Regardless of which is the action taken to handle the violation, we assume that handlers are designed to resolve violations without violating other temporal constraints. That is, a minimum duration handler must resolve minimum duration violations without violating other temporal constraints (e.g., maximum duration constraints).

In [144], the authors discuss the importance of interrupting activities safely, i.e., by preserving their context. Context preservation refers to the capability of the system to save data associated with the activity at the right time. In this direction, duration constraints can be seen as information related to activity execution, and thus, they must be dealt with when the activity is interrupted.

MINIMUM DURATION VIOLATION	
Wait	The process waits before proceeding to the following element or a delay is added until minimum duration is observed.
Repeat	The whole activity or part of it is repeatedly executed until minimum duration is met.
Compensate	A compensation handler reverses the effects of the activity. In BPMN, compensation is used to revert the effects of a <i>Completed</i> activity that are no more desired [11].
MAXIMUM DURATION VIOLATION (WEAK)	
Escalate	Dedicated activities are performed to expedite completion. In BPMN, escalation identifies a situation that presumes some sort of reaction by the process and it is used to implement measures to expedite the completion of a process activity [11].
Extra Workforce	The activity is assigned to multiple resources that execute parts of it in parallel.
MAXIMUM DURATION VIOLATION (STRONG)	
Skip	The activity is interrupted and the remaining is skipped.
Undo	The activity is interrupted and its effects are reversed by some other activity.

Table 3.4: Possible behaviors of duration handlers, enacted in case of violation.

For both minimum and maximum duration violations, although we did not detail which activities are executed by the temporal exception handler, we adopt a modeling level of abstraction sufficient to ensure that handlers are correctly blended within the process.

In [52], we discussed different approaches for managing violations of simple duration constraints in a weak and a strong way. Let us start from the duration-aware process model depicted in Fig. 3.4. Signal events `t_minViolated` and `t_maxViolated` detect duration violations and are used to trigger the corresponding violation handlers, that may have either a weak or a strong behavior.

For managing minimum duration violations caused by early activity interruption we used *compensation*, i.e., a way of undoing steps of a successfully completed activity whose results are no more desired and must be reversed. Minimum duration violations are managed by compensation handlers, that are constituted by a set of activities that are not connected to other portions of the BPMN model [11]. Instead, maximum duration violations are managed by event subprocesses. In BPMN, event subprocesses are a specialized kind of subprocess, that is included within a parent process but it is not part of the control flow. In

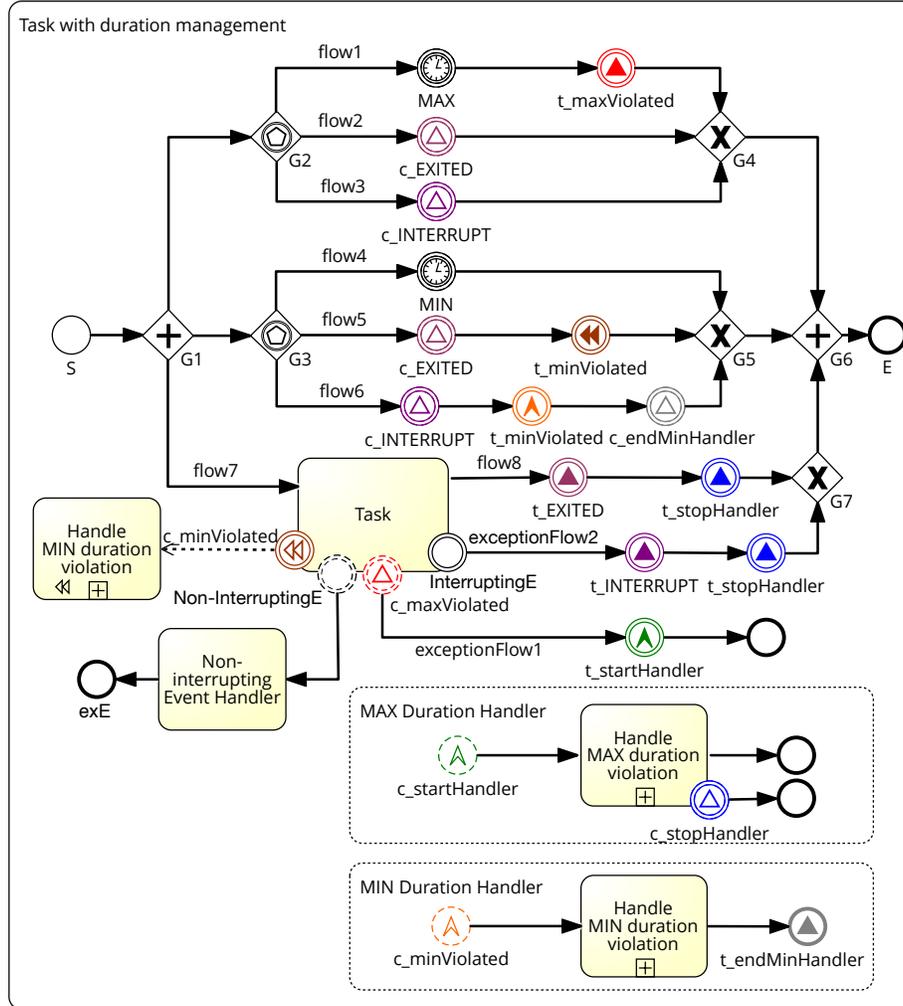


Fig. 3.19: Process model for managing minimum and *weak* maximum duration constraints of activities having interrupting and non-interrupting intermediate boundary events.

other words, event subprocesses are *inline handlers* triggered by events coming from the parent process [11].

In the remainder of this section, we consider the more complex case of activities having both interrupting and non-interrupting intermediate boundary events illustrated in Fig. 3.6 and discuss how violation handlers defined in [52] have to be extended to deal with such a scenario.

When considering boundary events attached to a running activity, different interruption scenarios may occur. We begin with considering an activity having a weak maximum duration constraint, as depicted in Fig. 3.19. We distinguish three different violation management behaviors depending on when Task is interrupted by the boundary event: (i) interruption occurs earlier than the minimum duration set for Task, (ii) interruption occurs within the expected duration range of Task or (iii) interruption occurs after the maximum duration limit set for Task.

For all cases (i), (ii) and (iii), it is necessary to distinguish the interruption of Task due to boundary event `InterruptingE` from regular task completion. To this end, a different signal `t_INTERRUPT` within `exceptionFlow2` is added. This is essential to deal with minimum duration violation, as the activity can be either in state *Completed* or *Terminated* (cf. Fig. 2.2). The different name is chosen to highlight that the end of Task depends on causes other than anticipated completion.

(i) If `InterruptingE` occurs before the minimum duration set for Task, a temporal exception handling mechanism different from compensation must be enacted, as compensation only applies to successfully completed activities. For this reason, an event subprocess `MIN Duration Handler` is added to the parent process `Task with duration management`. Escalation event `t_minViolated` within `flow6` is added to trigger the handler. Signal event `c_endMinHandler` waits for the event subprocess to complete, before letting the process flow proceed towards G5. Signal event `t_stopHandler` following signal `t_INTERRUPT` within `exceptionFlow2` has no corresponding listening events on the process branches entitled of minimum duration management, as it is used for handling maximum durations.

(ii) If `InterruptingE` occurs within the expected duration range, signal event `t_INTERRUPT` is thrown to be caught by the corresponding event `c_INTERRUPT`, located on `flow3`. The signal behaves as previously explained for event `c_EXITED`, as the only difference is the state *Terminated* of Task. Again, signal `t_stopHandler` is thrown ineffectively, as no handler was initiated.

(iii) If `InterruptingE` occurs after `MAX`, as the activity is interrupted, we assume any possibly executing instance of subprocess `MAX Duration Handler` is also interrupted by signal event `t_stopHandler`. This interruption behavior is strong and, thus, it is in contrast with the notion of weak maximum constraint that has been addressed so far. However, as external occurrences presuppose strong interruption, there is no mean to preserve activity execution while the handler is operating on it.

When the event attached to Task boundary is non-interrupting, neither the management of minimum nor maximum duration constraints violations is affected. In case of weak maximum duration, we assume that the handler is designed to expedite Task completion and, thus, takes care of completing all the related non-interrupting event handlers attached to the activity.

The corresponding model for the management of violations of strong maximum duration constraints is reported in Fig. 3.20.

In this setting, only two possible process behaviors are expected with respect to activity interruption: (i) the boundary event interrupts Task before `MIN` or (ii)

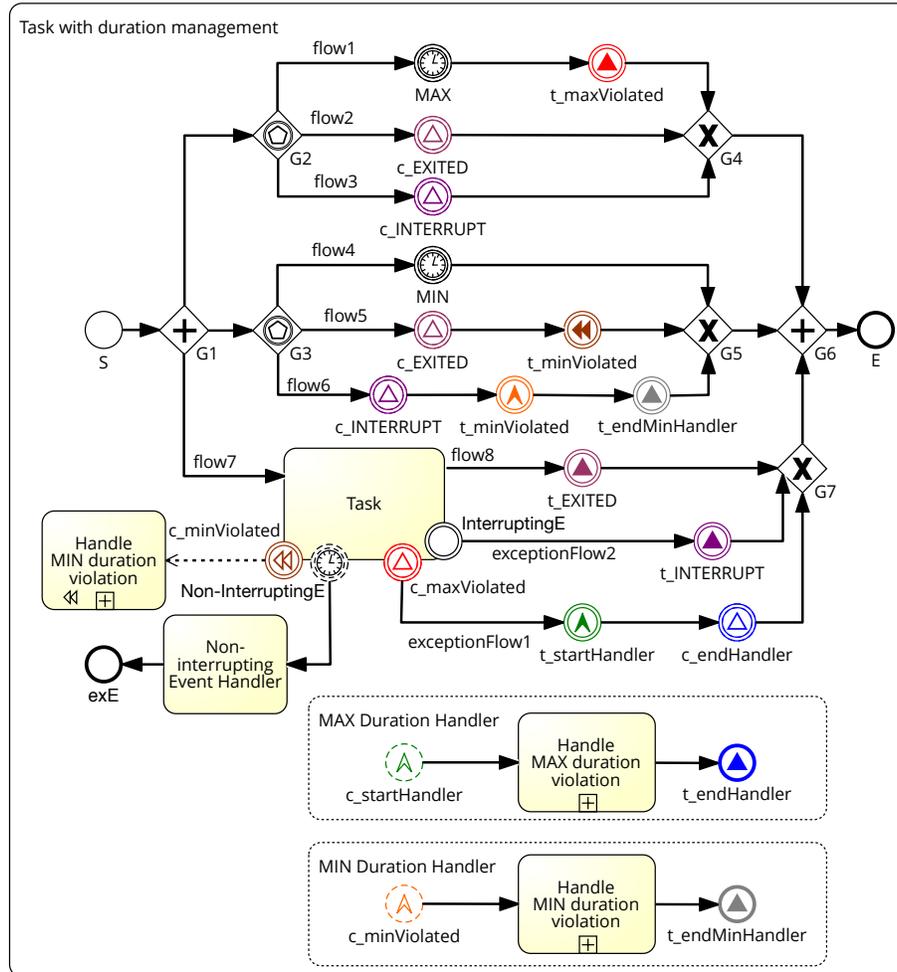


Fig. 3.20: Process model for managing minimum and *strong* maximum duration constraints of activities having interrupting and non-interrupting intermediate boundary events.

at any moment within the desired time limits set for Task duration. Indeed, the interrupting nature of signal event $c_{maxViolated}$ prevents any other interrupting event from occurring after MAX.

(i) If minimum Task duration is violated, signal event $t_INTERRUPT$ is thrown. The different name is chosen to highlight that the end of Task depends on causes other than anticipated completion. As already explained, escalation event $t_{minViolated}$ within flow6 triggers the corresponding event subprocess MIN Duration Handler.

(ii) If `InterruptingE` is triggered within the desired duration range, no duration violation handler is enabled. In this case, signal event `t_INTERRUPT` is thrown to be caught by the corresponding event `c_INTERRUPT` within `flow3`. Even if the final state of `Task` is *Terminated*, no further action is needed with respect to duration management.

3.7 Process Verification with Time Petri Nets

The goal of this section is to provide an unambiguous operational semantics for the proposed BPMN processes by exploiting time Petri nets to verify their behavior and prove their correctness.

Born to be understood by different users and to be employed in several organizational domains, process modeling languages often lack of a fully-specified, formal semantics. When a formal semantics exists, it is mostly defined in terms of transition systems or Petri nets [2], which have been introduced in Sect. 2.1.2.

Accordingly, the execution behavior of a process model can be specified in terms of Petri nets, as their formal semantics is particularly suitable for disambiguating that one of BPMN and to check the correctness of process models [78]. In general, Petri nets can be derived from BPMN processes by applying the mappings introduced in [73] and in [79], which differ from each other on the level of abstraction used to capture the life-cycle of a process activity.

In this thesis, in order to be able to express the temporal dimension associated to BPMN activities and timer events, we adapt the mappings presented in [73, 79] to time Petri nets, that is, Petri nets with a possibly infinite time interval associated to each transition [65, 120]. Thereby, we can capture the temporal aspects related to BPMN activities and timer events. Time Petri nets have been previously used in [126] for deadline constraints modeling.

Last but not least, we ensure that the obtained time Petri nets conform to workflow nets in order to exploit the interesting properties of the latter ones.

To begin with, let us consider the classical definition of workflow net $WN = (P, T, A, M_0, e, c)$, provided by Def. 2.5 in Sect. 2.1.2.

By adding a transition t^* which connects the sink place c to the source place e we obtain a strongly connected net $\overline{WN} = (P, T \cup t^*, A \cup \{(c, t^*), (t^*, e)\}, M_0, e, c)$ which is referred to as “short-circuited net” [77].

A necessary and sufficient condition for soundness relating workflow nets to their corresponding short-circuited nets is reported below [77].

Theorem 3.2. A workflow net WN is sound if and only if the corresponding short-circuited net \overline{WN} is live and bounded.

As introduced in Sect. 2.1.2 well-structured process descriptions are guaranteed to be sound if they are live [105].

In this chapter, we consider workflow nets, as they result more intuitive in depicting the structure and execution steps of a business process. However, in

order to capture temporal aspects, we consider a particular class of high level Petri nets, i.e., time Petri nets. Time Petri nets have been initially introduced by Merlin [120] and aim to support performance evaluation, safety determination, or behavioral properties verification in systems where time appears as a quantifiable and continuous parameter [65].

A time Petri net is a Petri net with a possibly infinite time interval associated to each transition [65]. The left extreme of the interval is the minimal time that must elapse from the time that all the input conditions of a transition are enabled until this can fire. The right extreme of the interval denotes the maximum time that the input conditions can be enabled and the transition does not fire. After this time, the transition must fire.

The formal definition of time Petri net, adapted from [65], is provided below.

Definition 3.3 (Time Petri Net). A time Petri net is defined as a tuple $N = (P, T, A, M_0, I)$, where:

- (P, T, A, M_0) is a Petri net;
- $I : T \rightarrow \{\mathbb{R}^+, \mathbb{R}^+ \cup \{\infty\}\}$ is a firing time function that associates an interval $[\downarrow I(t), \uparrow I(t)]$, called static firing interval, to each transition t .

In line with this definition, each transition $t_i \in T$ has an associated time interval $[a, b]$, where $a \leq b$, and such that, once t_i has been enabled:

- a ($0 \leq a$), is the minimum time that t_i must remain continuously enabled until it can fire;
- b ($0 \leq b \leq \infty$), is the maximum time that t_i can remain continuously enabled without firing.

According to this scenario, any Petri net $N = (P, T, A, M_0)$ can be represented by an equivalent time Petri net, having an interval $[0, \infty]$ associated to each transition $t_i \in T$.

In the context of time Petri nets, the concept of state and of firing rule need to be revised. In particular, a state S of a time Petri net N is defined as the couple $S = (M, IS)$ consisting of:

- (i) a marking M , which denotes the distribution of tokens in places and describes the logical part of the state;
- (ii) a firing interval set IS , which is a vector of possible transition firing times and denotes the timed part of the state. The number of (ordered) elements of IS corresponds to the number of enabled transitions in M and each element $i \in IS$ is the time interval $[a, b]$ associated to enabled transition t_i .

According to the above definition of time Petri net (cf. Def. 3.3), transitions are *enabled* when every input place for a transition t_i has at least one token, as for Petri nets.

However, an enabled transition t_i , associated to time interval $[a, b]$ cannot *fire* before a and later than b . Therefore, the relative firing time θ , which is related to the absolute enabling time ρ , must not be smaller than a of transition t_i and

not greater than the smallest of the b 's of all the transitions enabled by marking M (i.e., a of $t_i \leq \theta \leq \min\{b\text{'s of } t_k\}$, where k ranges over the set of transitions enabled in M).

Example 3.8 (Firing of a time Petri net). As an example, let us consider the time Petri net depicted in Fig. 3.21, represented in state $S = (M_0, IS_0)$.

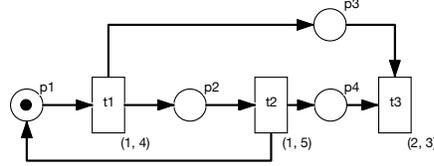


Fig. 3.21: Example of time Petri net, where $P = \{p1, p2, p3, p4\}$, $T = \{t1, t2, t3\}$ and initial marking $M_0 = p1(1), p2(0), p3(0), p4(0)$.

To show how the considered net fires, we need to check whether both conditions (i) and (ii) hold for the current state S . Given a marking $M_0 = p1(1), p2(0), p3(0), p4(0)$, as $t1$ is the only *enabled* transition, $IS_0 = \{(1, 4)\}$, that is $1 \leq \theta_1 \leq 4$. Let us suppose that $\theta_1 = 2$, then

$$S \xrightarrow{(t1, \theta_1)} S' \begin{cases} M_0 = p1(1), p2(0), p3(0), p4(0) \xrightarrow{t1, \theta_1} M_1 = p1(0), p2(1), p3(1), p4(0) \\ IS_0 = \{(1, 4)\} \xrightarrow{t1, \theta_1} IS_1 = \{(1, 5)\} \end{cases}$$

This leads to state $S' = (M_1, IS_1)$, where $M_1 = p1(0), p2(1), p3(1), p4(0)$ and $IS_1 = \{(1, 5)\}$, where $t2$ is the only enabled transition. If $\theta_2 = 1$, we have that

$$S' \xrightarrow{(t2, \theta_2)} S'' \begin{cases} M_1 = p1(0), p2(1), p3(1), p4(0) \xrightarrow{t2, \theta_2} M_2 = p1(1), p2(0), p3(1), p4(1) \\ IS_1 = \{(1, 5)\} \xrightarrow{t2, \theta_2} IS_2 = \{(1, 4), (2, 3)\} \end{cases}$$

Now, we obtain $S'' = (M_2, IS_2)$, where $M_2 = p1(1), p2(0), p3(1), p4(1)$ and $IS_2 = \{(1, 4), (2, 3)\}$. In this case, both $t1$ and $t3$ are enabled: $t1$ can be fired for $\theta_1 \in (1, 3)$ and $t3$ can be fired for $\theta_3 \in (2, 3)$.

To summarize, below we show to possible alternative sequences of firings, one with $\theta_1 = 3$ and the other with $\theta_3 = 2$.

$$S'' \xrightarrow{(t1, \theta_1)} S''' \begin{cases} M_2 = p1(1), p2(0), p3(1), p4(1) \xrightarrow{t1, \theta_1} M_3 = p1(0), p2(1), p3(1), p4(1) \\ IS_2 = \{(1, 4), (2, 3)\} \xrightarrow{t1, \theta_1} IS_3 = \{(1, 5), (0, 0)\} \end{cases}$$

$$S'' \xrightarrow{(t3, \theta_3)} S'''' \begin{cases} M_2 = p1(1), p2(0), p3(1), p4(1) \xrightarrow{t3, \theta_3} M_4 = p1(1), p2(0), p3(0), p4(0) \\ IS_2 = \{(1, 4), (2, 3)\} \xrightarrow{t3, \theta_3} IS_4 = \{(0, 2)\} \end{cases}$$

In the remainder of this chapter, we make use of time Petri nets [65] to provide a formal foundation of the semantics of the proposed BPMN processes and to exploit existing tools for simulating their execution.

In general, process activities can be modeled with Petri nets according to two different paradigms, that is, either by transitions or by places [2].

In the first case, the marking of the net indicates a situation of rest and transitions model activities that may lead to a new state of “rest”. Oppositely, transitions can stand for instantaneous events and places may reflect a particular state of the net.

Within the BPM community, a widely applied mapping approach was proposed in [73]. An activity or an intermediate event is mapped onto a transition with one input place and one output place that models its execution, as shown in Fig. 3.22(a) for BPMN activity A.

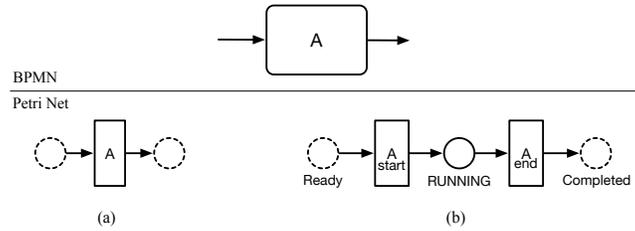


Fig. 3.22: Mapping of a BPMN process activity A onto a Petri net (a) abstracting from its life-cycle, as presented in [73], and (b) taking into account activity life-cycle, as introduced in [79]. Places drawn in dashed borders are shared with other net modules, as they are used as connecting elements.

A start or end event is mapped onto a silent transition, i.e., a transition whose firing cannot be observed, that signals when the process starts or ends. Gateways, except event-based and OR-split gateways, are also mapped onto silent transitions: AND-splits and AND-joins correspond to Petri net forks and joins, respectively, while BPMN XOR-split and XOR-joins correspond to Petri nets choice and return modules. In detail, data-driven exclusive gateways, are modeled as silent transitions having a common place as input and competing for a single token, so that the choice of which transition will fire is non-deterministic. For event-based gateways, the race condition between events is captured by having the corresponding transitions compete for tokens in the net place corresponding to the input flow of the gateway.

However, the mapping introduced in [73] does not consider the BPMN activity life-cycle (cf. Fig. 2.2). In some circumstances, it is important to represent the different states that occur from the creation of an activity to its completion, as explained in [79]. To this end, each activity can be mapped to a net having three places corresponding to states Ready, RUNNING, and Completed, respec-

tively. Transitions A_{start} and A_{end} denote the beginning and ending instants of the activity.

When a token is put on place `Ready`, the activity enters the state `ready`. This is properly represented by the firing rule of the Petri net: Transition A_{start} can fire and, then, a token is removed from place `Ready` and put on place `RUNNING`. This token remains on place `RUNNING`, representing the execution of the activity until transition A_{end} fires, thus removing the token from place `RUNNING` and putting it on place `Completed`.

The mapping of BPMN activities to Petri nets by considering their life-cycle is shown in Fig. 3.22(b).

In this section, we start from the mappings introduced in [73, 79] and map the proposed duration-aware processes onto time Petri nets in order to be able to express the temporal dimension associated to activities and timer events. In particular, we consider 1-bounded (or safe) time Petri nets having a delay-based semantics. That is, the time interval associated to a transition expresses the minimum and maximum delays before an enabled transition can fire.

By applying the mapping in [73], we could model activities as transitions having one input and one output place, and associated to a positive time interval. However, when considering timed transitions, this mapping approach is imprecise, as the time interval associated to the transition represents the delay that exists from its enabling to its firing and, thus, it does not capture the duration of a running activity (i.e., the transition firing is instantaneous).

Therefore, we consider activity states in the obtained time Petri nets and model an activity A as a time Petri net, having three places (that correspond to states `ready`, `running`, and `completing`) and two (timed) transitions, whose firing capture the beginning and ending of the activity. In this way, activity duration, which is the time span that goes from the firing of A_{start} to the firing of A_{end} , is captured by the time interval $[MIN, MAX]$ associated to the second transition that captures activity ending. Moreover, as we consider that activity duration does not cover the time span between the activation and the beginning of the activity, we consider transition A_{start} to be associated to a time interval of the form $[0, 0]$.

Similarly, timer events are modeled as transitions have one single input and one single output place and associated to a positive time interval. In detail, as we consider BPMN timer events that fire once a specified amount of time d has elapsed from their activation, the corresponding transition on the time Petri net will be associated to a time interval $[a, b]$ with $a, b = d$. Fig. 3.23 shows the mapping of BPMN activity A and timer event t onto the corresponding time Petri nets modules.

As for temporal aspects, all the remaining transitions in the net (i.e., transitions corresponding to gateways or other event types) are assumed to be associated to a time interval of the form $[0, \infty[$.

Finally, in order to model in Petri nets throwing and catching events that interact withing the same process, transitions corresponding to such events must be properly connected to each other. However, as the catching event may not

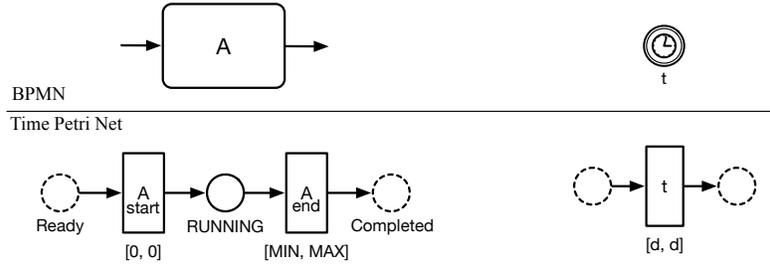


Fig. 3.23: Mapping of a BPMN process activity A and of a timer event t onto the corresponding time Petri nets. Places drawn in dashed borders are shared with other net modules, as they are used as connecting elements.

be listening when the corresponding throwing event fires, deadlocks can occur or tokens may remain in the net, unconsumed. This is particularly true for signal events, due to their broadcast semantics [11, 101]. As unconsumed tokens prevent the net from completing properly, for each catching signal event in the BPMN process, two transitions are needed in the time Petri net: one that captures the fact that the event is caught, the other one that captures the situation in which the event is not caught.

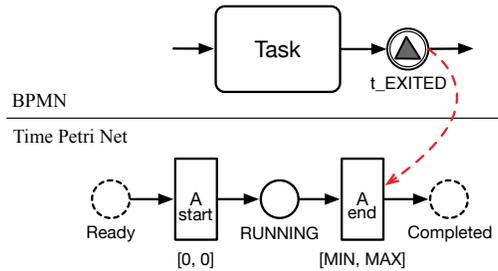


Fig. 3.24: Mapping of a BPMN task towards Petri net considering activity lifecycle [79]. Signal event t_EXITED is mapped to transition end_TASK.

Fig. 3.25 shows the time Petri net obtained from the duration-aware process of Fig. 3.4 as discussed above. As the meaning of signal event t_EXITED in the process is to detect Task completion, we map it directly to transition end_TASK, as shown in Fig. 3.24. The time interval [a, b] is written near each transition, but it is omitted for untimed transitions (i.e., when $[a, b] = [0, \text{inf}[$). Transitions NOT c_EXITED capture the non-catching of event t_EXITED in the BPMN process, to guarantee proper completion of the net. As transitions c_EXITED are both in competition with transitions MIN and MAX (i.e., race condition of event-

based gateway), the firing of MIN(MAX) must be related to the firing of NOT c_EXITED.

In order to verify the time Petri nets obtained from the proposed duration patterns and complete duration-aware processes and to check their soundness, we designed and checked the time Petri nets in Romeo [121] and TINA (Time Petri Net Analyzer) [122].

Both tools support abstract state space constructions that preserve specific properties of state classes, such as absence of deadlocks, linear time temporal properties, or bisimilarity. Informally, a state class groups all the states of a net according to all the possible firing times that are reachable from a given marking.

For instance, let us consider the net of Fig. 3.21 and its state $S'' = (M_2, IS_2)$, where $M_2 = p1(1), p2(0), p3(1), p4(1)$ and $IS_2 = \{(1, 4), (2, 3)\}$. In this setting, state class $C2 = (M_2, D_2)$ is given by marking $M2$ and a domain D_2 , which is the union of all firing intervals of the states, i.e., $D_2 = (1 \leq \theta_1 \leq 4 \cup 2 \leq \theta_3 \leq 3)$.

Besides working on state classes, both Romeo and TINA operate on standard time Petri nets. Therefore, in order to analyze our nets with respect to the soundness of workflow nets, we had to provide their short-circuited version. To do so, we connected the output place of transition E, with the input place of transition S by means of an additional transition. Fig. 3.26 shows the *stepper simulation* of the obtained short-circuited time Petri net in TINA. In the depicted scenario, transition end_TASK is expected to fire after transition MIN and transition MAX.

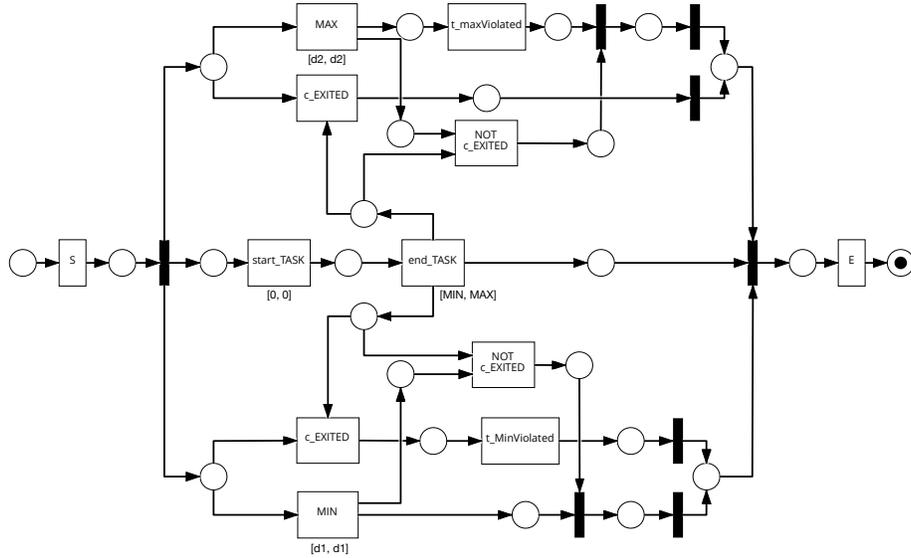


Fig. 3.25: Time Petri net obtained by applying the mapping proposed in [79] to the process of Figure 3.4, but considering the theory of time Petri nets [65].

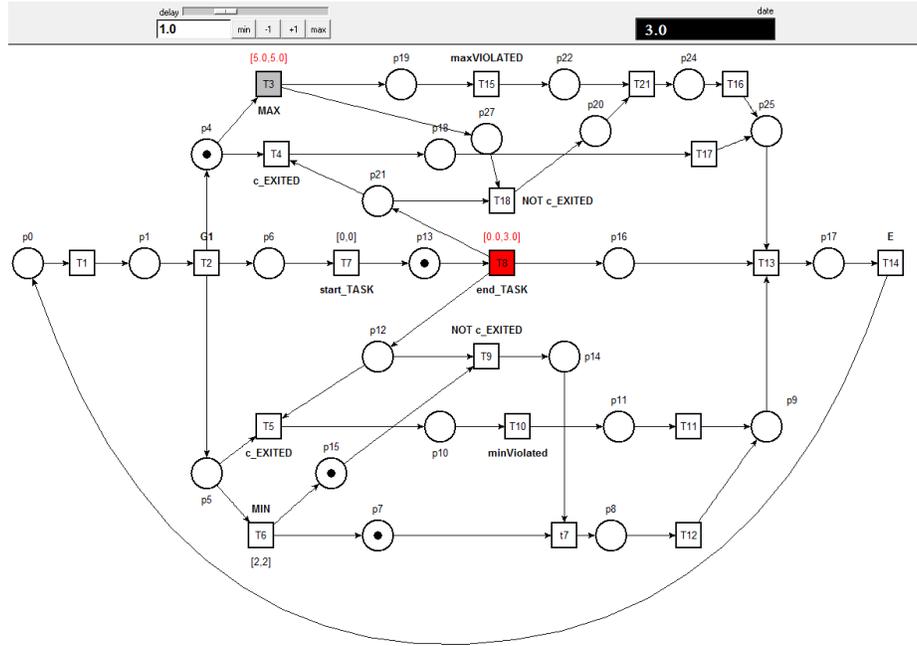


Fig. 3.26: Short-circuited time Petri net opened in TINA’s stepper simulator. The delay represents the current relative delay, whereas the black bar on the top right reports the total execution time of the net. Transition end_TASK is enabled and can be fired within the next three time units.

By assigning different time intervals to transitions end_TASK, MIN, and MAX, we derived multiple process execution traces that capture all the previously explained behaviors of the duration-aware process of Fig. 3.4. By stepping through the net model in TINA and Romeo, we were able to observe that the time Petri nets reproduced the expected behavior of all the considered process traces. Romeo’s simulation interface is shown in Fig. 3.28.

Finally, by analyzing the short-circuited time Petri net with TINA, we evinced that the net is live and bound, as reported in Fig. 3.27. Thus, thanks to theorem 3.2, we can state that the obtained time Petri net is sound.

3.8 Conclusion

In this chapter, we addressed the modeling of different duration constraints of activities and process regions, by using the BPMN standard. Structured and re-usable process models for specifying duration constraints at design time are proposed in Sect. 3.3–3.5 and they are enriched to provide detection and management of constraint violations at run-time as explained in Sect. 3.6.

```

LIVENESS ANALYSIS -----
possibly live
possibly reversible

0 dead classe(s), 71 live classe(s)
0 dead transition(s), 20 live transition(s)

STRONG CONNECTED COMPONENTS:

0 : 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37
36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

SCC GRAPH:

0 -> T16/0, T12/0, T14/0, T21/0, T11/0, t7/0, T13/0, T15/0, T18/0, T10/0, T9/0, T17/0, T5/0, T6/0, T3/0,
T4/0, T8/0, T7/0, T2/0, T1/0

0.000s
ANALYSIS COMPLETED -----

# net taskDuration, 24 places, 20 transitions #
# bounded, possibly live, possibly reversible #
# abstraction count props psets dead live #
# states 71 24 ? 0 71 #
# transitions 149 20 ? 0 20 #
    
```

Fig. 3.27: An excerpt of TINA’s full textual output analysis results for the time Petri net shown in Fig. 3.25.

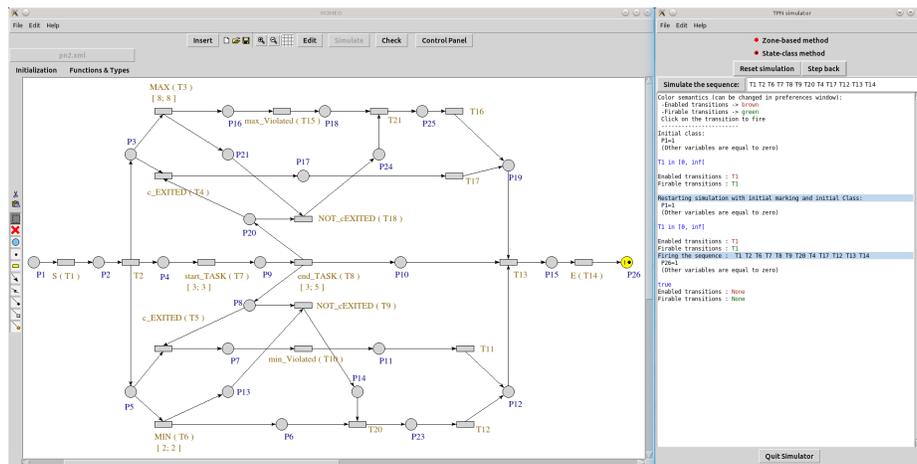


Fig. 3.28: Simulating time Petri net with Romeo [121]. The simulated sequence is reported at the top right, and the *State-class method* [65] has been chosen for simulation. As done for TINA, the short-circuited net has been checked for absence of deadlocks.

The different kinds of duration constraints and process models have been designed after studying real-world (clinical) settings, whose complex temporal aspects cannot be captured by simple duration constraints.

- The main steps of our approach can be summarized as follows.
- We proposed a set BPMN ready-to-use duration-aware process models enclosing duration patterns for specifying different kinds of duration constraints, and

for detecting possible constraint violations occurring at run-time. We entirely relied on the BPMN standard [11] for the definition of process semantics or on existing literature [33, 73, 79, 101, 145] whenever we considered that the semantics described in the BPMN standard was underspecified.

- We simulated the obtained processes with the Signavio Process Manager [24] to assess the dynamic behavior of the proposed models. We were able to conduct a “step-by-step” simulation as, to our knowledge, no current BPM software supports single or multiple-case simulation of processes having both event-based gateways and signal events (maybe Scylla [146] could be extended in this direction). However, we used Signavio step-by-step simulation to assess process behavior for all the cases introduced in Sect. 3.3. This kind of simulation allows one to decide which of the process elements ready for execution can be enabled. An example of step-by-step simulation for the process model of Fig. 3.4 is shown in Fig. 3.29.

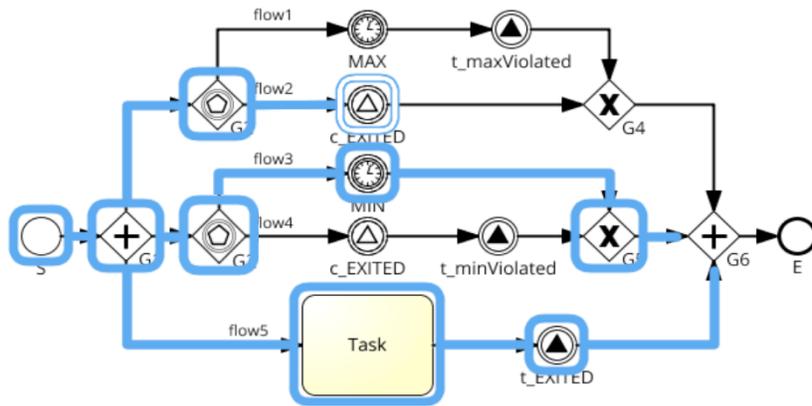


Fig. 3.29: Example of step-by-step simulation of the duration-aware process model of Fig. 3.4. Active execution traces are highlighted in light-blue.

- To verify the correctness of the designed process models, in Sect. 3.7 we manually derived an equivalent representation of the designed process models in terms of time Petri nets [65], considering both the mappings proposed in [73] and in [79], and results obtained on the soundness of *workflow nets*. The goal of this last step was to analyze the behavioral aspects of the derived nets to make sure that the specification of the operational semantics of the proposed process models was sound. Then, we also used both Romeo [121] and TINA (TIme Petri Net Analyzer) [122] for checking basic properties such as boundedness, liveness, and soundness of the obtained nets.

For future work, we plan to consolidate our modeling approach by evaluating how the proposed patterns blend with complex real-world process models including also other kinds of temporal constraints and by simulating the obtained time-aware process models directly with BPM tools.

Modeling and Enforcing Temporal Constraints in BPMN Processes

This chapter is based on results published in [54].

Time management plays a crucial role in all phases of business process design, enactment and analysis [16]. Business processes develop over time and their execution is often subjected to temporal restrictions: deadlines and calendar constraints have to be observed, timely reactions to unexpected occurrences must be ensured, and resources must be scheduled.

Moreover, different kinds of temporal conditions can constrain how a specific process path is preferred among others, during process execution. For instance, the temporal distance between milestone events can influence the routing of the process flow, parallel executions involving shared resources require coordination, and time lags between non-consequent activities can be adjusted based on which process path is taken.

The main contribution of this chapter is to lift the lid on the relationship between temporal constraints and the choice of alternative process paths, and to propose BPMN-based solutions that can be easily interpreted by existing tools.

In detail, we focus on three general kinds of temporal constraints stemming from studied real-world clinical scenarios [147] and propose a set of structural rules to guide their design in BPMN. In particular, we (i) deal with different kinds of temporal constraints in BPMN processes, focusing on temporal aspects that constrain the choice of specific execution paths; (ii) use a combination of BPMN patterns to enforce such constraints during process execution; (iii) give a formal semantics based on timed automata for the used BPMN constructs, in order to specify the expected behavior of the obtained processes; (iv) consider a real clinical scenario as a motivating application example, without compromising the generality of the proposed solution.

The remainder of the chapter is organized as follows. Sect. 4.1 outlines the basics of timed automata. Sect. 4.2 introduces a motivating clinical example and illustrates the related temporal constraints. Sect. 4.3 shows the modeling of temporal constraints in BPMN and provides a set of structural rules that can be

used as modeling guidelines. Sect. 4.4 proposes a mapping of BPMN elements and constructs onto timed automata, with the aim to provide a rigorous semantics the designed process models. Finally, Sect. 4.5 reviews the contribution of this chapter.

4.1 Foundations of Timed Automata

In this section, we introduce the basic theory of timed automata, originally introduced by Alur and Dill in [66], extending finite automata with (dense time) clocks with the aim to enable the specification of real-time systems.

However, the definitions provided in the remainder of this section are slightly different from those proposed in [66].

Firstly, we introduce the notions of *clock* and *clock constraints*. A clock x is a real non-negative variable ($x \in \mathbb{R}$ and $x \geq 0$) over which a clock constraint can be defined. A clock constraint is a condition of the form $x \leq r$, $x < r$, $x = r$, $x \geq r$, or $x > r$, with $r \in \mathbb{R}^+$. Given a sets of clocks X , we denote as $C(X)$ the set of all possible clock constraints defined on the clocks of X for every possible $r \in \mathbb{R}^+$. Given a a set of clocks X , a non-negative *clock assignment function* $ca : X \rightarrow \mathbb{R}$ and a clock constraint $c = x * k$, with $*$ $\in \{<, \leq, =, \geq, >\}$, in $C(X)$, we say that ca *satisfies* c , i.e., $ca \models c$, if and only if $ca(x) * k$. As it will soon be clarified, every configuration of an automaton is equipped with a clock assignment function.

A timed automaton TA is a quintuple $TA = (Q, q_0, X, M, \Delta)$ where:

1. Q is a finite set of states;
2. $q_0 \in Q$ is the initial state;
3. X is a finite set of clocks;
4. M is a finite set of channels;
5. $\Delta \subseteq Q \times C(X) \cup \{\epsilon\} \times M \cup \{\epsilon\} \times Q \times X \cup \{\epsilon\} \times M \cup \{\epsilon\}$ is a set of *transitions*. Each transition features two occurrences of elements in $M \cup \{\epsilon\}$: the first occurrence denotes a message received on the specified channel whereas the second one denotes a message sent on the specified channel. For clarity sake, we append to the channel of the received message the symbol $?$ and to the channel of the produced message the symbol $!$. Then, the resulting form of a transition is $(q, c, m?, q', x, m!)$. Moreover, we assume $\epsilon \notin X \cup M$.

Informally, an automaton TA may “fire” a transition $(q, c, m?, q', x, m!)$ in Δ if and only if its current state is q , its clock assignments satisfy either c or $c = \epsilon$, and it receives a message either on channel m or $m = \epsilon$. In the resulting configuration, the new state is q' , and the clock assignment function has been incremented by a non-negative value $r \in \mathbb{R}$ for all the clocks but x (if $x \neq \epsilon$).

The value for x in the new configuration is 0. After the firing of the transition, a message on the channel m' is produced. If $m \neq \epsilon$, the transition is said to be m -consuming, whereas if $m' \neq \epsilon$ the transition is said to be m' -producing.

A configuration of TA is a pair (q, ca) , where $q \in Q$ and ca is a clock assignment function. For the sake of simplicity, we suppose that there exists a clock

$x_g \in X$ that measures the global time of each configuration in a computation. This means that $x_g \neq x$ for every $(q, c, m?, q', x, m!) \in \Delta$ and, thus, $ca(x_g)$ is the global time of the configuration.

A transition $(q, c, m?, q', x, m!)$ is *ready* in a configuration (q'', ca) at time $r \geq ca(x_g)$ if and only if $q = q''$ and it holds that $ca'(x) \models c$, where ca' is the clock assignment function such that for every $x \in X$ we have $ca'(x) = ca(x) + (r - ca(x_g))$. A *trace* of TA is a finite sequence $T = (q_0, ca_0) \xrightarrow{\delta_0} \dots \xrightarrow{\delta_{n-1}} (q_n, ca_n)$, such that: 1. $ca_0(x) = 0$ for every $x \in X$; 2. for every $0 \leq i < n$, let $\delta_i = (q, c, m?, q', x, m!)$ we have that: $q = q_i$, $q' = q_{i+1}$, $ca_i \models c$ if $c \neq \epsilon$, $ca_{i+1}(x) = 0$ if $x \neq \epsilon$, and for every $x' \neq x$ in X we have that there exists a non-negative $r \in \mathbb{R}$ such that $ca_{i+1}(x) = ca_i(x) + r$. In a trace $T = (q_0, ca_0) \xrightarrow{\delta_0} \dots \xrightarrow{\delta_{n-1}} (q_n, ca_n)$ of TA , for every non negative real $r \in \mathbb{R}$, given a message $m \in M$, we say that TA is *m-deaf* at r if and only if every m -consuming transition δ in Δ is not ready in the configuration (q_i, ca_i) such that $ca_i(x_g) \leq r < ca_{i+1}(x_g)$.

Given a finite set of temporal automata $\mathcal{TA} = \{TA^0, \dots, TA^n\}$ with $TA^i = (Q^i, q_0^i, X^i, M^i, \Delta^i)$, with $X^i \cap X^{i'} = \emptyset$, for every $0 \leq i \neq i' \leq n$, we define a *parallel trace* of \mathcal{TA} as any set of computations $\mathcal{T} = \{T^0, \dots, T^n\}$ where $T^i = (q_0^i, ca_0^i) \xrightarrow{\delta_{i,0}} \dots \xrightarrow{\delta_{i,n_i-1}} (q_{n_i}^i, ca_{n_i}^i)$ for every $0 \leq i \leq n$ and such that:

1. for every $0 \leq i \leq n$, T^i is a trace of TA^i ;
2. for every non-negative $r \in \mathbb{R}$ and every $m \in \bigcup_{0 \leq i \leq n} M^i$, there exists at most one index i and at most one index j for which the configuration (q_j^i, ca_j^i) features $ca_j^i(x_g) = r$ and transition $\delta_{i,j}$ is m -producing (i.e., a channel may be used by at most one automata at time in order to produce a message);
3. for every non-negative $r \in \mathbb{R}$ and every $m \in \bigcup_{0 \leq i \leq n} M^i$, there exist at most one index i and at most one index j for which the configuration (q_j^i, ca_j^i) features $ca_j^i(x_g) = r$ and the transition $\delta_{i,j}$ is m -consuming (i.e., a channel may be used by at most one automaton at one time);
4. for every index $0 \leq i \leq n$ and every $0 \leq j \leq n_i$, if $\delta_{i,j}$ is m -consuming then there exist $0 \leq i' \leq n$ and $0 \leq j' \leq n_{i'}$ such that $\delta_{i',j'}$ is m -producing and $ca_j^i(x_g^i) = ca_{j'}^{i'}(x_g^{i'})$ (i.e., consumed messages must have been produced);
5. for every index $0 \leq i \leq n$ and every $0 \leq j \leq n_i$ if $\delta_{i,j}$ is m -producing, either there exist j', i' such that $ca_j^i(x_g^i) = ca_{j'}^{i'}(x_g^{i'})$ and $\delta_{i',j'}$ is m -consuming, or, for every i' , we have that $TA^{i'}$ is m -deaf in $ca_j^i(x_g^i)$ (i.e., an automaton cannot ignore a message if it can consume it).

Given a set of states $Q_f \subseteq \bigcup_{0 \leq j \leq n} Q^j$, we say that a parallel trace $\mathcal{T} = \{T^0, \dots, T^n\}$ *accepts* Q_f if and only if $Q_f \subseteq \{q_{n_i}^i : 0 \leq i \leq n\}$.

4.2 Catheter-Related Bloodstream Infections

In this section, we introduce the temporal constraints considered in this chapter by means of a real-world motivating clinical example addressing some major steps in the treatment of Catheter-Related Bloodstream Infections (CR-BSIs) [148,

147, 149]. Starting from the framework introduced in [23], we consider time-lags between arbitrary events (cf. TP3 in [23]) affecting the choice of alternative process paths, temporal mutual exclusion (cf. TP6 in [23]), and time-lags between activities (cf. TP1 in [23]). All of the listed constraints occur quite often in real-world clinical processes [51, 147], especially those dealing with treatment and monitoring tasks.

Example 4.1 (Catheter-Related Bloodstream Infections). Vascular catheters are essential to treat critically ill patients. In the United States, more than 150 million intra-vascular catheters are purchased by clinics and hospitals each year [148]. However, their use increases the risk of developing deadly bloodstream infections, caused by pathogens that colonize the catheter.

Infection treatment depends on the patient conditions and on the kind and virulence of the involved microorganism responsible for causing the infection. Different pathogens need to be treated for a specific period of time, and with the most suitable antibiotic according to their resistance features.

Generally, the risk of incurring a CR-BSI depends on multiple factors, such as the type of catheter used, the frequency with which the catheter is accessed, the duration of catheter placement and the individual characteristics of the catheterized patient. Additionally, CR-BSI risk is conditioned by the education and the experience of healthcare personnel and by the actuation of proper prevention measures. Indeed, catheter-related bloodstream infections increase hospital costs and length of stay and they seem to increase mortality incidence, especially in critically ill patients.

Various clinical guidelines have been developed to prevent infection development and to define performance indicators to monitor treatment success. Clinical guidelines underline the need for modeling temporal constraints, as they are temporally oriented, they involve temporal reasoning, they can be easily described in terms of conditional decisions, and their applicability is often constrained by time conditions [26, 112].

The BPMN process of Fig. 4.1 shows the main steps for the detection and treatment of CR-BSIs in intensive care units, simplified from the well-known IDSA guideline [148].

As clinical findings are often unreliable for diagnosing bacteremia due to their lack of sensitivity and specificity, blood and/or catheter cultures are used to support the diagnosis of CR-BSI. When considering diagnostic methods that do not require catheter removal, i.e., that are exclusively based on blood cultures, clinicians **Draw blood samples** to be cultured. Blood samples are drawn from two sites: one sample is obtained through the central venous catheter suspected to be the source of the infection, whereas two additional samples are drawn from a peripheral vein. After withdrawing blood, physicians can **Administer an Empirical Therapy** to relieve the patient, until diagnosis is confirmed and a more appropriate treatment can be initiated.

Among catheter-saving criteria, Differential Time to Positivity (DTP) is the most used for diagnosing CR-BSIs [148]. DTP measures how much time is nec-

essary for a catheter-drawn blood culture to become positive for a given micro-organism with respect to a simultaneously-drawn peripheral blood culture. If the catheter culture becomes positive at least 2 hours earlier than the peripheral one, the catheter is considered to be the source of bacteremia. If this condition is not observed, physicians **Look for other sources of Infection**.

Let us call C+ the catheter culture and P+ the peripheral culture. When blood cultures are grown, one of the following scenarios is expected to occur.

- (i) If none of C+ or P+ turn positive, there is no evidence of infection in the blood;
- (ii) if only P+ turns positive, the catheter is not likely to be the source of the infection;
- (iii) if only C+ turns positive, the catheter is probably colonized;
- (iv) if both C+ and P+ turn positive, but the differential time is less than 2 hours, then the catheter is not likely to be the source of the infection;
- (v) if both C+ and P+ turn positive and the differential time between them is equal or more than 2 hours, than the DTP criterion for diagnosing a CR-BSI is fulfilled.

Based on the obtained culture results, physicians can **Confirm the onset of CR-BSI**. If a CR-BSI is diagnosed, appropriate treatment should be initiated, depending on the detected micro-organism. For simplicity, we present only the treatments of CR-BSIs caused by Coagulase-negative Staphylococci (CONS) and Enterococcus species.

When treating an infection caused by CONS, practitioners **Administer the Antibiotic Treatment** believed the most effective for the patient conditions. Besides, antibiotic or heparin lock therapy can be prescribed. **Catheter Lock** is a method to be used in conjunction with systemic antimicrobial therapy for sterilizing the catheter lumen and for preventing clot formation when the catheter is not in use. In general, the duration of the treatment is the same for both antibiotic and lock therapies.

Similarly, when dealing with Enterococcus species, Vancomycin is administered, but alternatives are considered if antibiotic resistance is observed. Enterococcal bacteremia is often associated to endocarditis, a life-threatening inflammation of heart valves. To diagnose endocarditis, physicians perform a **Trans-Esophageal Echocardiography (TEE)**, after having eventually sedated the patient. Both the introduced treatments for CONS and Enterococcus species end within a predefined amount of time, functional to patient improvement.

Due to their temporal and decisional characterization, clinical guidelines turn out to be a challenging setting for modeling time-based inter-activity constraints and decisions [114].

The following list introduces three different kinds of important temporal constraints that stem from the described clinical scenario and that influence the enactment of a certain process path over another.

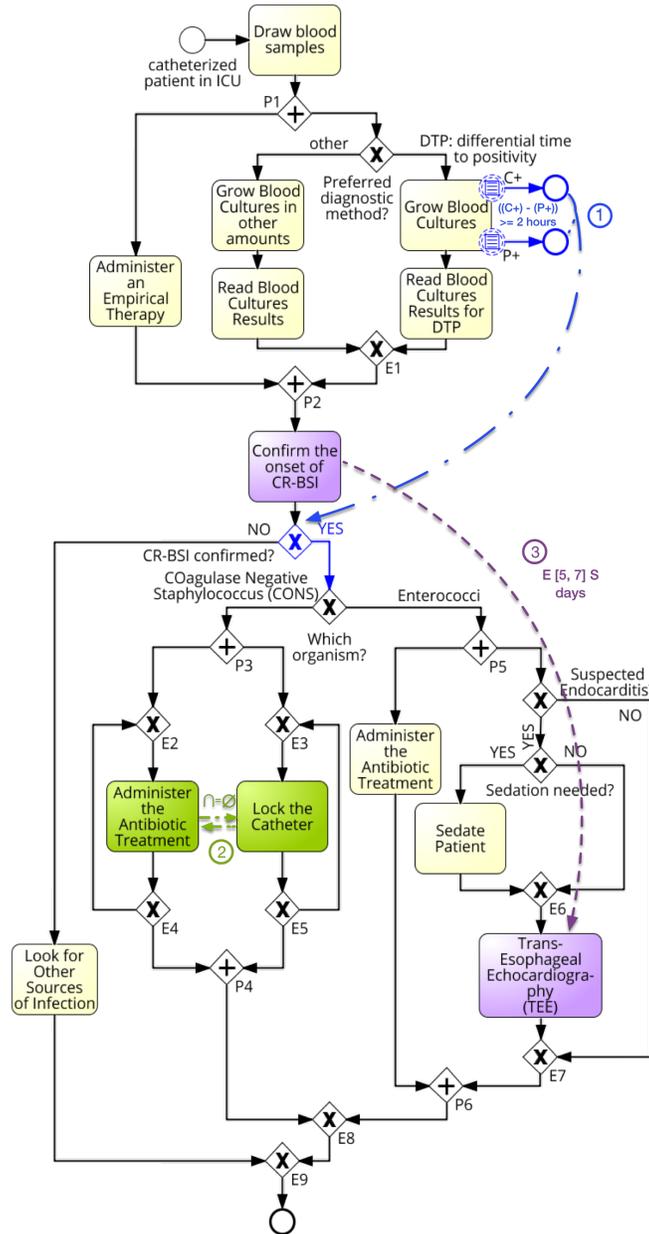


Fig. 4.1: BPMN process for CR-BSIs treatment with added time constraints: ① temporal delta driven decision, ② temporal mutual exclusion between concurrent activities, and ③ relative time constraint.

- ① *Temporal Delta Driven Decision, TD^3* . Consider two milestone events occurring in the context of a decisional task. The time elapsing between their occurrence determines which is the process path to be chosen at a split node located downstream in the process flow.

A compact representation of TD^3 constraint is shown in Fig. 4.1, considering the introduced DTP criterion. A blue dashed-and-dotted edge connects the occurrence of two conditional events to the exclusive gateway that represents the point in the process where the made decision is applied and another edge of the same sort highlights which is the chosen outgoing path.

Non-interrupting conditional events C+ and P+ are attached to the boundary of task **Grow Blood Cultures**, in order to represent the occurrence of specific (milestone) conditions during task execution. If both events occur, and the time span between such happenings conforms to the DTP criterion, i.e., the temporal distance between them is ≥ 2 hours, the “Yes” branch of gateway **CR-BSI confirmed** is taken.

- ② *Temporal Mutual Exclusion between Concurrent Activities*. A common scenario in BPMN processes is that of having multiple concurrently executing activities operating on a shared resource [23]. In some situations, concurrency cannot be avoided due to the semantics of the modeled reality. However, synchronization must be achieved in order to prevent conflicts involving the shared resource.

In Fig. 4.1, temporal mutual exclusion regulates the conjunct administration of systemic antibiotic treatment and catheter lock. Both therapies have to be administered synergistically for a predefined amount of time, but a single catheter cannot be used contemporaneously. Indeed, even though both therapies are expected to last the same number of days and treatment effectiveness relies on their synergistic administration, a single catheter cannot be used to deliver both treatments, at the same time. For this reason, any administration must prevent other administrations to be performed during the same temporal span.

In Fig. 4.1, such condition is highlighted by a couple of oppositely oriented lime-green dashed arrows, labeled with $\cap = \emptyset$, meaning that the executions of the two tasks **Administer the Antibiotic Treatment** and **Catheter Lock** cannot overlap.

- ③ *Relative Time Constraint*. Relative constraints limit the temporal distance between any non-consecutive process elements. We consider enforcing relative time constraints defined between the starting/ending instants of two tasks and having the form $T_1I [start, end] T_2I granularity$. T_1I (T_2I) is the instant (either E or S, for end and start, respectively) of the first (second) task involved in the constraint, $[start, end]$ denotes the possible values of the temporal distance between the tasks, whilst *granularity* is the chosen granularity for the constraint. We also assume that $start \neq end$.

In Fig. 4.1, a relative constraint between **Confirm the onset of CR-BSI** and **Trans-Esophageal Echocardiography (TEE)** is depicted through a purple dashed edge, labeled with $E[5, 7]S days$. If endocarditis is suspected, a TEE

is performed 5–7 days after the onset of bacteremia, in order to minimize false-negative results.

The focus of this work is on modeling and enforcing TD^3 constraints in BPMN, as they have a great impact on the process flow and, to our knowledge, their modeling has not been addressed to date. Models for the other introduced constraints can be obtained by applying design principles that are similar to those presented in Sect. 4.3 for TD^3 .

4.3 Modeling Temporal Constraints in BPMN

The illustrated temporal constraints ①–③ can be represented and enforced in BPMN by making use of synchronization patterns based on signal events.

Fig. 4.2 shows a simple pattern that supports synchronization between any couple of concurrent process branches. We call this event-based pattern *handshake*. A handshake allows a process that needs to be temporally constrained to interact with a *controller* branch, which is added through two parallel gateways, PG1 and PG2. In Fig. 4.2, the dashed sequence flow is a graphical expedient for representing the possible presence of other process elements. The semantics of this pattern is quite straightforward: the controller broadcasts a SYNCHRO_SIGNAL to be caught by the corresponding signal event on the main process. Signal events are used in BPMN for broadcast communication within a process. Once the synchronization signal has been caught, signal event ACK is triggered, to acknowledge synchronization. However, the token traversing the main process, may not have reached the catching SYNCHRO_SIGNAL yet. For this reason, an event-based gateway is added to the controller branch: if signal ACK is not caught within a certain amount of time, defined by timer event CLOCKTIME, SYNCHRO_SIGNAL is re-broadcast, until synchronization is achieved. Without loss of generality, the amount of time represented by CLOCKTIME is intended to be the smallest possible with respect to the application domain.

Let us refer to the process of Fig. 4.1. The TD^3 constraint, which holds between blood cultures and the diagnosis of bacteremia when the DTP criterion is

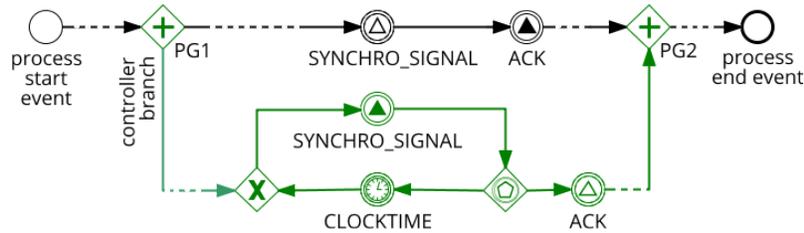


Fig. 4.2: BPMN-based handshake pattern for synchronizing parallel process flows.

fulfilled, can be enforced with a combination of handshake patterns. Let `C+_true` (`P+_true`) identify the fact that catheter (peripheral) culture turns positive. With respect to CR-BSI treatment, any of the following mutually exclusive scenarios can occur.

- Another diagnostic method is used, that is, the process branch containing the `Grow Blood Cultures` task is skipped, i.e., `NOT_CHOSEN`. In this case, the DTP criterion is not applicable and, trivially, TD^3 does not hold.
- The process branch containing the `Grow Blood Cultures` task is `CHOSEN`, but either none of the two events `C+_true` and `P+_true` occur, or `P+_true` occurs alone. Also in this case, the DTP criterion is not satisfied, and nor is TD^3 .
- The process branch containing the `Grow Blood Cultures` task is `CHOSEN`, and `C+_true` occurs. As the DTP criterion may be applied, the time spanning from the occurrence of `C+_true` is measured. Then:
 - a) if `P+_true` occurs, but the differential time to `C+_true` is less than 2 hours, the DTP criterion is not satisfied, and, thus, TD^3 constraint is not observed.
 - b) if `P+_true` occurs, and the measured time span is at least 2 hours (i.e., `Timedelta_true`), the DTP criterion is satisfied, and TD^3 holds.
 - c) if 2 or more hours have passed from the occurrence of `C+_true`, but `P+_true` does not occur (it might have occurred earlier than `C+_true`), the DTP criterion is not satisfied, and TD^3 is not observed.

Each of the presented scenarios can be expressed with the help of a handshake pattern, having as synchronization signal the result of the decision to be taken, i.e., `NOT_CHOSEN` if the DTP criterion is not applicable, `NO_BRANCH` if it is not satisfied, and `YES_BRANCH` if it is fulfilled. As we are considering human tasks in a clinical domain, we can set the handshake timer event to “1 min”. An event-based gateway drives the flow according to which situation takes place. The temporal distance between the occurrence of events `C+` and `P+`, is measured by a timer event having value “2 hours” and triggered immediately after `C+_true` has occurred. Timer event `Timedelta_true` is used to signal when the required temporal distance of 2 hours has elapsed.

A set of construction rules for connecting the combination of handshake patterns to the main process of Fig. 4.1 is provided to maintain the whole diagram well-structured and to prevent undesired modeling pitfalls.

Let us call D the exclusive gateway where the decision is evaluated (e.g., `CR-BSI confirmed?` in Fig. 4.1). Similarly, let us call T the decisional activity located upstream in the context of which the two decision-driving events are expected to occur (e.g., `Grow Blood Cultures` in Fig. 4.1). We will refer to the process D -block as the process SESE region delimited by D and its corresponding merge node, both of which are excluded from the block.

Let us define a T -path as any reverse-flow path in the process that starts from T and ends with the process start event. $Pre(T)$ is the ordered set of split nodes

(i.e., gateways of any sort) that are encountered when traversing any T -path, and that do not have a corresponding merge node in the path.

In Fig. 4.1, $Pre(\text{Grow Blood Cultures}) = \{\text{Preferred diagnostic method, P1}\}$. Similarly, let us define a D -path as any path in the process starting from D , included, and ending with the process end event. $Post(D)$ is the ordered set of merge nodes that are encountered when traversing any D -path and that do not have a corresponding split node in the path.

In Fig. 4.1, $Post(\text{CR-BSI confirmed}) = \emptyset$.

In order to soundly connect to the main process the combination of handshake patterns that are used represent and enforce TD^3 , the following rules must be observed.

- (i) Two conditional non-interrupting events, $e1$ and $e2$ (C+ and P+ in Fig. 4.1), are attached to the boundary of T , to represent the possible occurrence of the two decision-driving events. On the flow outgoing from each one of them, throwing signal events $e1_true$, $e2_true$ are added to detect event occurrence: $e1_true$ (C+_true in Fig. 4.1) is thrown only if $e1$ occurs, and the same holds for $e2$.
- (ii) A throwing signal event T_done is added to the sequence flow outgoing of T , to be caught by the handshake patterns only if none out of $e1$ and $e2$ occur.
- (iii) A throwing signal event $CHOSEN$ is added to the sequence flow entering T , to indicate that the process branch containing T has been chosen.
- (iv) A throwing signal event NOT_CHOSEN is added to any of the flows outgoing from each exclusive gateway belonging to $Pre(T)$ and alternative to a T -path. If no such gateway exist in $Pre(T)$, this rule is not applied.
- (v) The two parallel gateways, PG_1 and PG_2 , that limit the handshake pattern, have to be connected to the process flow right before the last element of $Pre(T)$ and immediately after the last element of $Post(D)$, respectively. If $Pre(T) = \emptyset$, then PG_1 is attached to the entry edge of T . If $Post(D) = \emptyset$, then PG_2 is attached to the exit edge of the D -block.
- (vi) D must become an exclusive event-based gateway, having either two or three catching signal events in its context.
 - A signal event NO_branch is followed by the branch of the D -block that evaluates decision D to “No”.
 - A signal event YES_branch is followed by the branch of the D -block that evaluates decision D to “Yes”.
 - A signal event NOT_CHOSEN must be added if and only if $|Pre(T)| \neq 0$. This event is followed by a D -block, and it indicates that D is not evaluated, that is, process execution does not follow a T -path.
- (vii) After each one of the signal events introduced in 6, a throwing signal event ACK is added to acknowledge synchronization.

The BPMN process obtained by applying the introduced rules to the diagram of Fig. 4.1 is shown in Fig. 4.3. Such complex diagram represents the TD^3 con-

straint in BPMN, but it is not meant to be read by a business user, who would rather see a simplified representation such as in Fig. 4.1.

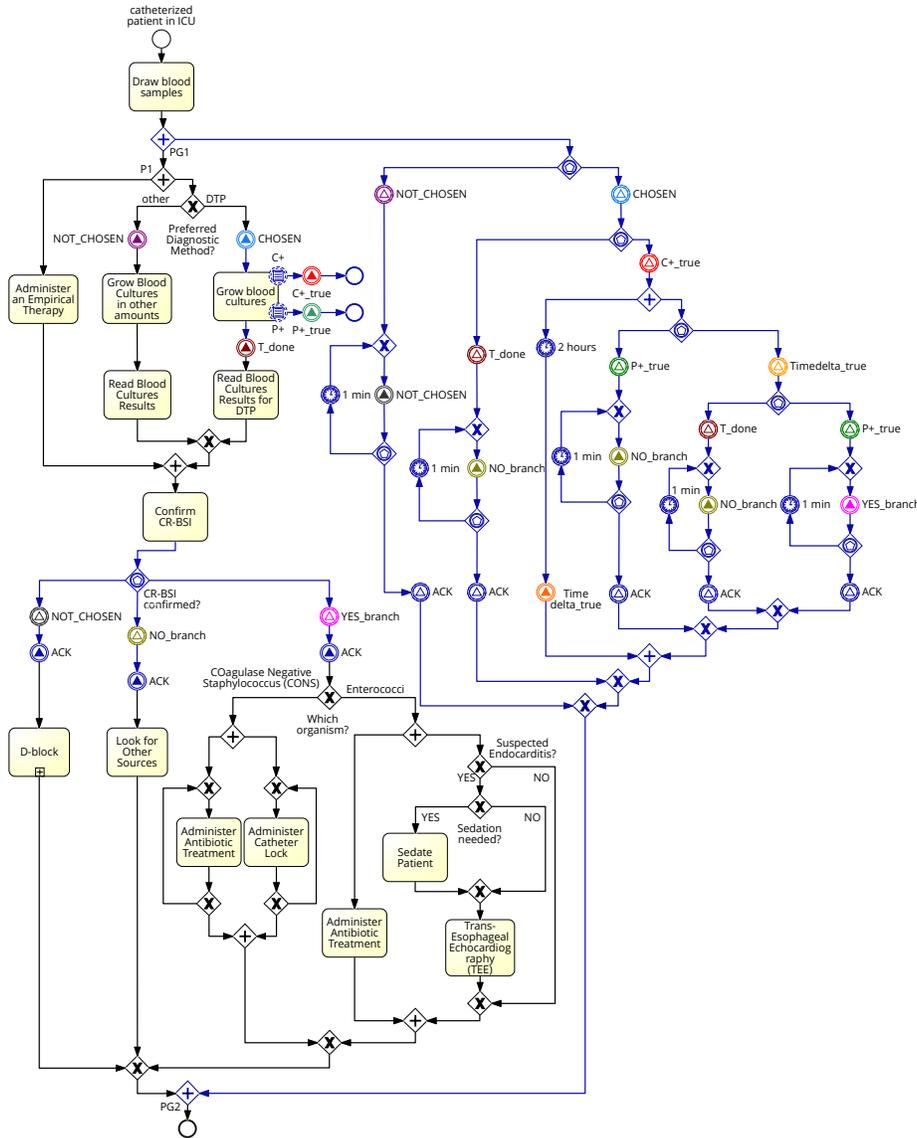


Fig. 4.3: BPMN diagram for the TD^3 constraint, obtained by applying the rules described in Sect. 4.3.

4.4 Mapping BPMN onto Timed Automata

In this section, we provide the semantics in terms of timed automata of the BPMN modeling elements used in the introduced process models. The goal of the proposed mapping is to formalize the semantics of selected BPMN elements, thus enabling model checking with known tools.

At a high level of abstraction, a BPMN process is represented by a set of temporal automata \mathcal{TA} , and any correct execution is represented by a parallel trace accepting a particular final state of \mathcal{TA} . Accordingly, each (combination of) BPMN element(s) is mapped onto a temporal automaton. In detail, given a BPMN element e , we call input component *in* any process element immediately preceding it on the sequence flow, and output component *out* any element immediately following e on the sequence flow.

All the automata introduced share a common behavior, explained as follows. The automaton TA representing e waits in its initial state for a message to arrive on channel c_{in} . If such message arrives (not all the components are active during the execution), the automaton can start its computation, thus reproducing the semantics of e . When execution is completed, the automaton moves to a final state, producing a message on channel c_{out} , which is directed towards the element's output component *out*. Intuitively, the process sequence flow is represented as an ordered pair of input/output components (*in*, *out*), where *in* corresponds to the source and *out* corresponds to the target of the sequence flow. The definition of input/output components for a process element can be extended to SESE regions: we refer to any BPMN element located on the region's entry edge as input component and to any BPMN element located on the region's exit edge as output component for that region.

The proposed encoding may be easily translated into the semantics used by common simulators/model-checkers employed for timed automata verification (e.g., UPPAAL [123]).

In order to represent the semantics of BPMN correctly, we have to specify a particular set of accepting states Q_f that guarantees that all and only the correct parallel traces are considered. In Fig. 4.4, the elements of Q_f are depicted as double circled states.

The timed automata corresponding to the *start* and *end* events of a BPMN process are depicted in Fig. 4.4(a). The automaton proposed for the *start_event*, does not wait for any message to arrive, as it simply produces an output message on the channel directed towards its output component *out*. On the contrary, the automaton corresponding to the *end_event* waits on channel *in* for a message generated by the immediately preceding flow element, that is, its *in* component. Once the message is received, the automaton moves to its final state, producing a message on channel *end.com*, meaning that the end of the process has been committed.

The timed automaton corresponding to a generic BPMN task is depicted in Fig. 4.4(b), and its functioning is described as follows. The automaton waits for an initialization message, coming from its *in* component. If the message arrives,

the automaton resets its clock and waits. When the automaton is in state q_1 , it may decide to stop its execution in a non-deterministic way (i.e., when the task has been completed), provided that its clock is greater than zero (i.e., the task has required a positive amount of time to be completed). When the task is completed, a message to its *out* component is sent through the c_{out} channel and the automaton's state is reset to q_0 . Such transition allows the task to be repeated an arbitrary number of times, as, for instance, when the task is part of a loop.

The automaton representing a task with non-interrupting events attached to its border is clearly more complex, as shown in Fig. 4.4(f).

In the presented case, we consider having only two non-interrupting events attached to the task border, but the design approach may be applied to tasks having an arbitrary number of boundary events. Once the automaton is initialized, its execution spans over a positive amount of time during which, either $ev1$ or $ev2$ may occur, in any order. Since we assume events to occur instantaneously, we model them as messages, sent on appropriate channels. It is important to notice that we require the task to be listening for both the events during execution: this ensures that, whenever an event occurs at the task starting/ending instants, it is not considered.

According to BPMN semantics, a non-interrupting event triggers a new parallel execution, which is independent from the executing task, but needs to end any time prior to task completion. Therefore, whenever one of the attached events ev is triggered (i.e., a message is produced on the ev channel to which the automaton is listening) the automaton produces a message on the channel of the output component for that event, called $in.ev$. Then, the automaton must wait until the execution started in $in.ev$ halts in end . In this case, the automaton cannot halt until the message $end.com$ for ev has been produced.

The region of the automaton consisting of the states going from q_2 to q_{10} deals with all the possible inter-leavings of $ev1$ and $ev2$ and their relative triggered executions, including all the combination of cases in which either one event or both events do not occur. Finally, it is worth to point out that simultaneous signals on distinct channels, representing, for example, the case having the two events occurring at the same time, are handled by the fact that transitions without clock constraints may be fired without spending time. As an example, let us suppose the automaton is in state q_1 , with $x > 0$, that is, we are listening for events. Let us assume that messages on $ev1$ and $ev2$ are produced by two distinct automata. If the messages on channels ev_1 and ev_2 are received at the same time, we can move either to q_2 and then to q_5 or to q_3 and then to q_5 , without having to increment the clock.

On the other hand, we cannot construct automata that consume an arbitrary number of messages at the same time. Indeed, according to the semantics given in Sect. 4.1, the pair r, m , where r is the time and m is the message produced/consumed, is required to be unique and, thus, the number of messages that can be produced/consumed at the same time in a parallel trace for the whole set of automata \mathcal{TA} is bounded by $|\bigcup_{0 \leq j \leq n} M^j|$.

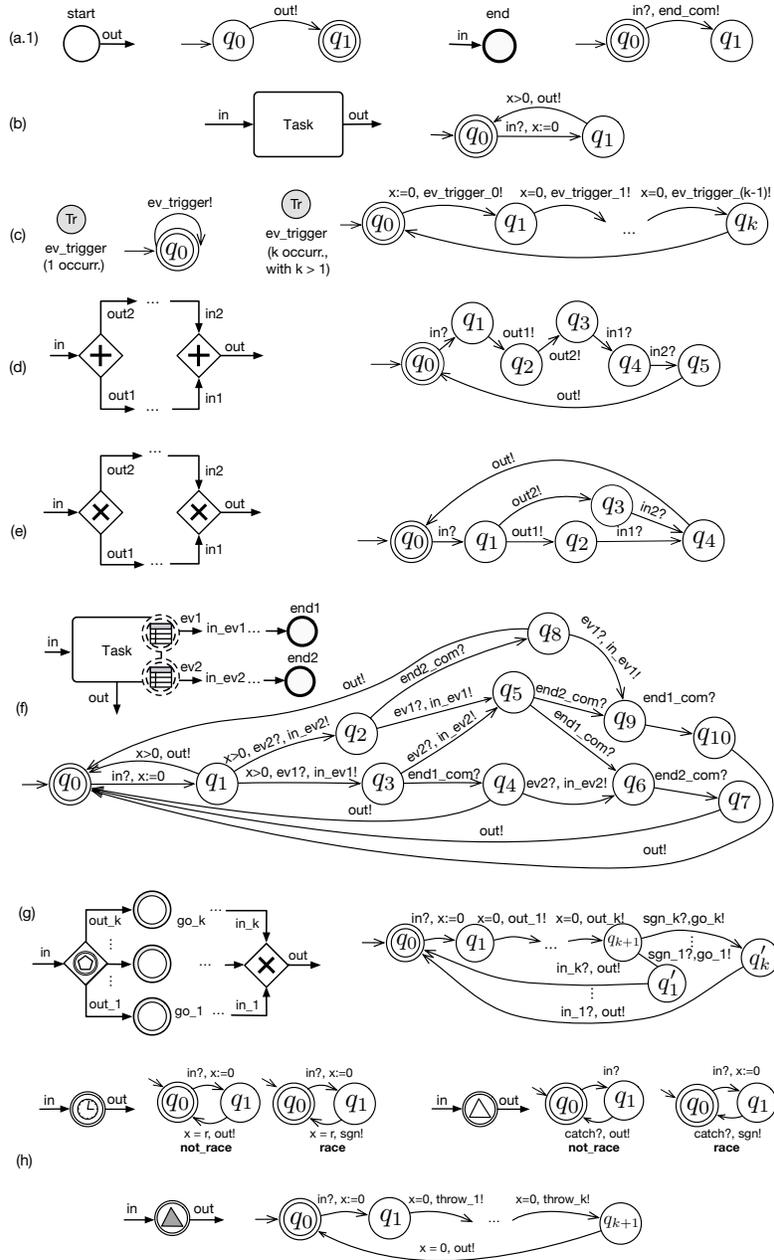


Fig. 4.4: Timed automata corresponding to the subset of BPMN elements used in this chapter.

The automata corresponding to generic intermediate events are depicted in Fig. 4.4(c). When dealing with events, we distinguish two cases: the case in which the event is considered by only one task, depicted in Fig. 4.4(d), on the left, and the case in which the same event is considered by more than one task, depicted on in Fig. 4.4(d), on the right. The behavior of the automaton in the first case is quite straightforward: the automaton can decide in a non-deterministic way if waiting a certain amount of time before triggering the event or, if not triggering the event, it may stay in its initial state which is also a final state.

In the second case, multiple “instances” of the same event are handled by the system. As mentioned earlier, different event instances have diverse names, such as $event_0, \dots, event_k$, but, in reality, they refer to the same event and, thus, we have to guarantee that the corresponding messages are simultaneous. As in the previously discussed case, the automaton may decide that the event does not happen. However, if the event occurs, this must be propagated instantaneously to each channel to which it is linked. This can be achieved by resetting a clock x , while the message on the channel $event_0$ is sent and, then, each successive message produced on the channel $event_j$, with $0 \leq j \leq k$, is required to occur without incrementing x , as expressed by the clock constraint $x = 0$. Since x is not reset, starting from q_1 , we have to ensure that all the messages are produced simultaneously in any successive state. It is worth to notice that having $q_0 \in Q_f$, avoids incurring in the scenario in which the automaton remains stuck in some of the states q_1, \dots, q_k after having triggered an event.

Let us consider the automaton corresponding to a SESE region delimited by split and merge parallel gateways, depicted in Fig. 4.4(d)

Once initiated, the automaton waits for a trigger message in before launching the automata for the parallel branches by means of messages $out1$ and $out2$. The automaton triggers its output component out if and only if it has received both messages $in1$ and $in2$, each one representing a process branch converging to the merging parallel gateway.

The automaton corresponding to a SESE region delimited by split and merge exclusive gateways is shown in Fig. 4.4(e). After being initiated by a message received on its in channel, the automaton chooses non-deterministically to trigger for execution one among the two automata representing the components connected to its split node. Then, the automaton waits for the message for the branch that has been activated and triggers the automaton representing the component connected to the output of its join node.

The automaton corresponding to the event-based gateway is depicted in Fig. 4.4(g). In this case, the automaton must choose the branch corresponding to the first catching event that is triggered. After being initiated by a message received on its in channel, the automaton must reproduce the simultaneous activation of the catching events located in the context of the event-based gateway. Simultaneity is achieved as follows. The clock x is set to 0 and each transition from q_i to q_{i+1} for $1 \leq i \leq k$ is guaranteed to be executed at the same time by the clock constraint $x = 0$.

As it will be clarified later, catching events have a different behavior when located in the context of an event-based gateway (i.e., racing condition). By now, let us assume that the automaton representing the i -th catching event, with $1 \leq i \leq k$, is triggered for execution by a message sent on channel out_i . Then when the i -th catch component succeeds it does not trigger for execution directly its go_i component but it produces a message sig_i . By the semantics of the timed automata, the automaton waits until the first message arrive. Let us suppose that the catch component i for some $1 \leq i \leq k$ sends a message on sig_i before all the other catch components $i' \neq i$ with $1 \leq i' \leq k$.

Then the automaton consumes the message sig_i and moves to the state q'_i producing a message only on the channel go_i and thus activating only the output component of the i -th catch component. Then the automaton waits for the output of the i -th branch he has activated on the channel in_i . When the message in_i is consumed, we have that the automata moves back to the state q_0 and it produces a message on the out channel in order to trigger for execution the automaton associated to its output component.

Finally, we detail the semantics of the event trigger types used in this work, considering both throwing and catching events. The automata corresponding to timer and signal events are depicted in Fig. 4.4(h). Catching events can have a different semantics, depending on the fact that they are located in the context of an event-based gateway (i.e., racing condition) or not. Therefore, for both signal and timer catching events, two different automata, namely “race” and “not_race”, are introduced. Both automata for the timer event behave as follows. Following initiation, the automaton sets the clock x to 0, waits in q_1 until $x = r$, and then it either produces a message on the sgn channel, if it is in race condition, or it sends a message to its output component. The automaton corresponding to the catching signal event shows a similar behavior. After the automaton is initiated, it waits in q_1 until it receives a message on its $catch$ channel, and then it either produces a message on the sgn channel, if it is in race condition, or it sends a message to its output component. Finally messages for the catch-event component are produced by the throw-event component provided that the channel $throw$ and $catch$ have the same name. The automaton corresponding to a throwing signal event produces a message on the channel $throw$ (transition from q_1 to q_2) and at the same time it triggers for execution its output component (transition from q_2 to q_0).

4.5 Concluding Remarks

The role of temporal conditions embedded within decisions that drive the process flow is still unexplored to date. In this chapter, we tackled a BPMN-based representation of a few temporal constraints that contribute to direct the process flow towards alternative paths. We focused on modeling and enforcing such constraints by means of BPMN modeling elements, in order for the obtained process models to be interpreted by existing tools supporting the standard notation.

A formal semantics, based on timed automata, is provided to characterize the expected behavior of the designed processes. Besides, the given formal semantics allows one to check the proposed models by using existing tools and techniques for the verification of timed automata.

Managing Decision Tasks in Time-aware Business Process Models

This chapter is based on results published in [55].

In the last years, temporal features of process models have been widely considered and studied with a focus on different intertwined aspects. Among them, we mention the modeling and checking of temporal constraints at design time [27, 119, 150, 151], the management of uncertainty for task duration [29], the modular design of time-aware processes [132], the specification of time patterns [152], and the modeling of temporal constraints in BPMN [31, 52, 153].

In general, temporal features of process models have to be dealt with by considering how they relate to the semantics of process elements. Particularly, decision tasks and events [11] are important concepts to consider jointly with temporal constraints, as they represent points in the process flow where information is acquired and used to determine the following flow of process execution. Indeed, information about decisions is used by exclusive gateways to choose one among alternative execution flows, based on the evaluation of conditions previously set by decision tasks or related to event occurrence.

Thus, during execution time-aware processes have to face two different kinds of uncertainty, one related to activity duration, which is known only after activity completion, the other one stemming from the outcomes of decision tasks and from events that determine which process path to follow. Such uncertainties are solved only when tasks have been executed and events have occurred.

At design time, given a time-aware business process model, it is desirable to know whether it is possible to execute it in a correct way, by considering all the possible combinations of activity durations and decision outcomes. However, such durations and outcomes are not under the control of a process engine. In this scenario, if a process engine can plan the execution of future steps considering only the history of already executed elements and made decisions, and guaranteeing that all the specified temporal constraints are satisfied, we say that the process is *dynamically controllable*.

In this chapter, we propose a new *time-aware* well-structured process model based on BPMN [11] to handle the subtle relations between temporal constraints and decisions and show how to check if process cases can be executed successfully. The main novelties of our approach can be summarized as follows:

- (i) we add temporal features to process elements, considering also the impact of event occurrence on temporal constraint management,
- (ii) we discuss the relation between the making and the use of decisions,
- (iii) we conceptually distinguish two novel types of decisions,
- (iv) we describe how to deal with both the uncertainty related to the effective duration of executed activities and the uncertainty related to decisions made during the process execution,
- (v) we define the notion of *dynamic controllability*(DC) for such processes, and
- (vi) we show a mapping of time-aware well-structured BPMN processes onto suitable temporal constraint networks to check the dynamic controllability of such processes.

The remainder of this Chapter is structured as follows. Sect. 5.1 exemplifies temporal constraints that may be found in real processes. Sect. 5.2 introduces time-aware BPMN processes. Then, Sect. 5.3 discusses dynamic controllability checking and shows how time-aware BPMN processes can be mapped onto Conditional Simple Temporal Network with Uncertainty and Decision (CSTNUds) [67] for checking dynamic controllability.

5.1 The Treatment of Knee Osteoarthritis as a Motivating Example

As a motivating scenario, let us consider the management of patients diagnosed with knee osteoarthritis (knee OA), whose main steps are shown in the process diagram of Fig. 5.1. The core of the diagram is designed by using BPMN [11], which is enriched with different kinds of temporal constraints, such as activity/gateway durations, event waiting times, sequence flow delays, and relative temporal constraints.

Example 5.1 (Knee Osteoarthritis). Knee OA is a common degenerative joint disease involving cartilage and nearby tissues [154]. Patients affected by knee OA complain about joint pain, which is worsened by excessive body weight and physical activity, stiffness, and limited mobility. The treatment of knee OA is based on a combination of nonpharmacologic and pharmacologic modalities, which are administered simultaneously to reduce pain and functional impairment, and to improve joint mobility. In general, nonpharmacologic therapy includes weight loss and exercise programs, use of assistive devices, and personalized social support all aimed at improving the quality of life.

In Fig. 5.1 we focus on pharmacologic treatment, thus leaving nonpharmacologic treatment represented as collapsed subprocess NonPhTr. Moreover, we

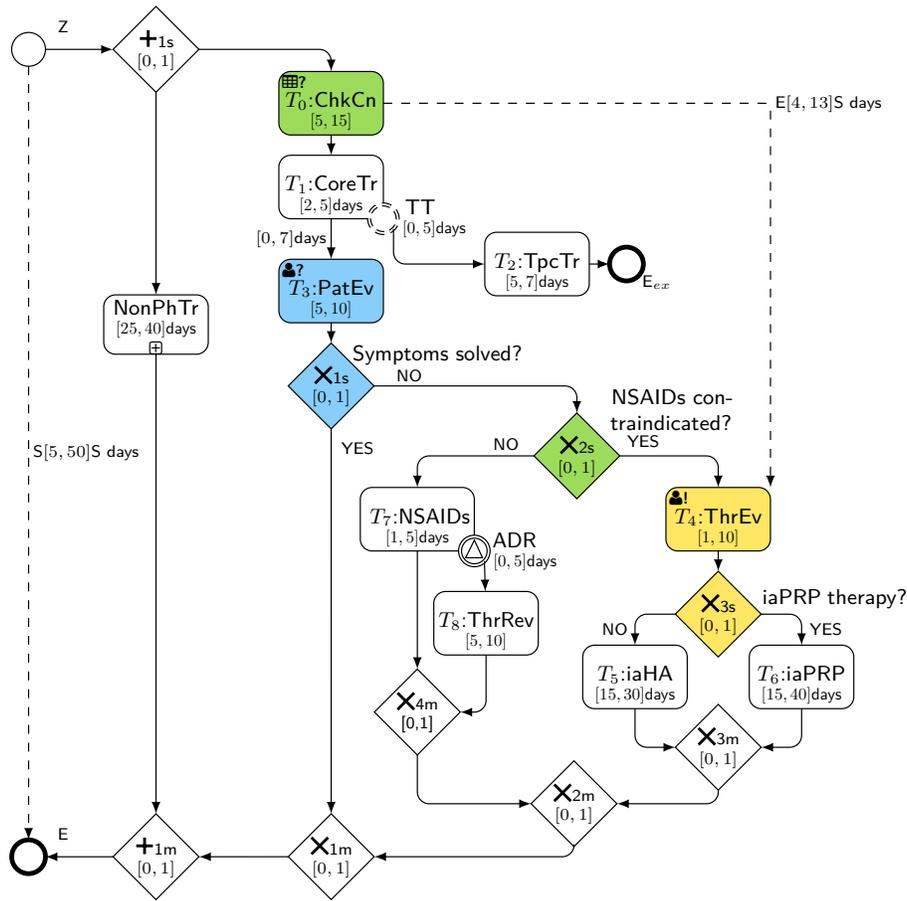


Fig. 5.1: BPMN process diagram showing the main steps for treating knee osteoarthritis. If not specified, the granularity of temporal ranges in the diagrams is assumed to be minute(s).

specify a type only for decision tasks: following DMN we assign type *user* to tasks representing human decision-making and type *business rule* to those enclosing a detailed decision logic [47].

Pharmacological treatment follows an incremental approach, that is, more powerful drugs are prescribed as the disease progresses and symptoms worsen.

Prior to prescribing treatments, a physician in charge must *Check Contraindications* (business rule task T_0) to commonly administered drugs, such as paracetamol, Non-Steroidal Anti-Inflammatory Drugs (NSAIDs), and opioids.

Being potentially life-threatening, absolute contraindications are precisely defined in clinical guidelines to avoid misinterpretation (hence the use of a business rule task).

Depending on the assessed drug tolerance, different treatments are prescribed, in a stepwise manner. The *Core Treatment* (task T_1) is essentially based on paracetamol, but topical preparations may be added during core treatment to improve pain control.

In Fig. 5.1 the need for topical treatment is represented by non-interrupting event TT, which leads to the *Topical Treatment* itself (task T_2). However, paracetamol often shows inadequate efficacy, even if combined with topical treatment. Thus, a *Patient Evaluation* (task T_3) is scheduled afterwards. If symptoms persist (gateway \times_{1s}), an advanced treatment must be prescribed.

The choice (gateway \times_{2s}) of the best treatment for the patient depends on the contraindications evaluated in T_0 . In case of contraindications, intra-articular therapy is preferred. Among existing alternatives, intra-articular hyaluronic acid (iaHA) and platelets-rich plasma (iaPRP) have shown similar efficacy [155].

Therefore, during *Therapy Evaluation* (user task T_4) the physician can choose the therapy among the available ones.

This decision differs from those made in T_0 and T_3 since not all of the possible outcomes are required to be available at run-time to guarantee that the process can be executed successfully. Indeed, it is sufficient that at least one of the available therapies iaHA and iaPRP can be chosen. Then, gateway \times_{3s} uses the decision made in T_4 to route the process flow towards either T_5 or T_6 .

If there are no contraindications, NSAIDs drugs (task T_7) may be administered. However, severe adverse drug reactions (ADRs) may occur while taking NSAIDs: if an ADR is reported, the treatment is immediately interrupted. In Fig. 5.1, this scenario is captured by signal boundary event ADR, whose occurrence interrupts task T_7 and leads to *Therapy Re-evaluation* (task T_8).

Example 5.1 shows how process execution relies also on temporal and decision aspects. On the one hand temporal constraints must be satisfied to guarantee that the process is completed successfully. On the other hand, decision tasks determine which process path is preferred over another.

5.2 Characterization of Time-aware BPMN Processes

In this section, we introduce temporal aspects, distinguish the two types of decisions that characterize the novel time-aware BPMN and discuss their relations. Then, we propose the notion of dynamic controllability for time-aware BPMN processes. Hereinafter, we consider only well-structured processes as they offer several advantages in terms of comprehension, modularity, and robustness [107, 150], as introduced in Sect. 2.1.3.

5.2.1 Specification of Temporal Properties and Constraints

In many application domains it is desirable to execute processes while observing certain temporal properties. Moreover, studying such temporal aspects at design time helps preventing undesired run-time scenarios and improves planning in terms of resource allocation [151].

In this section, we enrich BPMN processes by adding a temporal dimension to a relevant subset of BPMN elements and suitable temporal constraints based on concepts presented in [29, 119, 150]. The obtained *time-aware* BPMN fosters the temporal characterization of tasks, gateways, events, sequence flow edges, and time lags between process elements.

We describe the introduced temporal aspects by referring to the example of Fig. 5.1 and borrowing the notions of “activity/event activation” and “event triggering/handling” from the BPMN standard [11].

Activities. Activities have a duration attribute represented as a range $[x, y]G$ with $0 < x < y < \infty$, where x/y is the minimum/maximum allowed time span for an activity to go from state “started” to “completed” [152] and G (Granularity) stands for the time unit used (e.g., seconds, . . .).

At run-time, the real duration of an activity cannot be fixed by the process engine, but only observed after who is in charge of executing it completes the activity (*contingent duration*). Process engine takes into account the real duration to properly enact the following elements. Who is in charge of executing the activity must observe the two bounds x and y .

For example, T_3 has a duration $[5, 10]min$: physicians in charge of T_3 may take between 5 and 10 min to execute it.

Intermediate catching events. Intermediate catching events have a temporal property, $[x, y]G$ with $0 \leq x \leq y < \infty$, representing the minimum/maximum amount of time during which they may be triggered (*event waiting time*). When x is 0, it means that the event may be triggered as soon as it is activated. y is the upper bound on the amount of time allowed for event triggering that prevents the process to wait infinitely for the occurrence of an event.

Since the triggering of an event is not controlled by the process engine, the actual event waiting time is only known at run-time. When a catching event is attached to the boundary of an activity, its waiting time is implied by the duration of the activity. Specifically, if activity T has a duration $[x, y]mG$ and a boundary event e , then the event waiting time for e must be $[0, y']mG$ where $y' < y$ and mG is the minimum time granularity considered in the process. This ensures that e cannot occur at the T completion instant as required by [11]. For practicality, in case of coarser granularity the model admits $y' \leq y$ assuming that y' is always before y after the conversion to the minimum granularity.

As an example, task T_7 , having duration $[1, 5]days$, and boundary event ADR, having waiting time $[0, 5]days$: in this case, since *days* are coarser than *min-*

utes (the minimum granularity), the upper bounds coincide. If e is *non-interrupting* (e.g. TT in Fig. 5.1), BPMN requires that the duration of the associated activity includes the duration of all non-interrupting event handlers [11]. As discussed later, such specification can be satisfied by combining the described temporal properties, thus allowing designers to think about the elementary temporal characterization of activities.

Gateways and sequence flow. Gateways and sequence flow edges also have a duration range of the form $[l, u]G$, with $0 \leq l \leq u \leq \infty$. However, in this case, the process engine plans the real execution time for such elements by choosing a suitable value of the range. A range associated to a sequence flow edge connecting an element A to an element B is called *sequence flow delay* because it represents the possibility for the process engine to delay the enacting of B after A is completed.

For example, between T_1 and T_3 there is a delay of $[0, 7]$ days: considering the decision in T_0 and the completion time of T_1 , the engine could reduce this delay even to 0 to guarantee that following tasks can be executed without constraint violations. If a designer does not set a duration, it is assumed to be $[0, \infty]mG$.

Relative constraints. Relative constraints are depicted in Fig. 5.1 as dashed edges that connect any two process nodes [29]. Relative constraints limit the time distance between the starting/ending instants of two elements and have the form $I_S[u, v]I_FG$, where I_S is the starting (S)/ending (E) instant of the first element, while I_F is the starting/ending instant of the second one [29].

For example, the time distance between the end of task T_0 and the beginning of task T_4 is given by $E[4, 13]S$ days meaning that, if iaHA or iaPRP are needed, the decision of which one to prescribe must be made after at least 4 days and before 13 days from the completion of T_0 . To deal with event instantaneity, we choose to always adopt the notation I_S to denote the triggering instant of one event. For example, constraint $S[5, 50]S$ days represents the overall minimum and maximum process durations and holds between events Z and E.

5.2.2 Specification of Decisions: A Novel View on Decision Outcomes

The modeling decisions associated to processes is becoming increasingly important. In this section, we provide a novel view on decision tasks, based on where and how their outcomes are used in a process.

In our proposal, decisions are made in decision tasks and any following exclusive gateway may use the outcome of such decisions to route the process flow [11]. In this way, there is a greater flexibility compared to the assumption that decisions are made by a decision task immediately preceding the exclusive gateway [156]. Moreover, allowing a decision to be made at any place prior to an exclusive gateway, may increase temporal flexibility during process execution.

For example, T_0 determines the therapies contraindicated for a certain patient, thus affecting which process path is taken at gateway \times_{2s} . If the outcome

of T_0 is that NSAIDs are contraindicated then, to guarantee that at least one of T_5 and T_6 can be chosen, the delay $[0, 7]$ days between T_1 and T_3 must be set to 3 days at the most, for not precluding the possibility to satisfy also the overall duration constraint $S[5, 50]$ S days. Otherwise, a delay of 7 days can be allowed.

Decision tasks and corresponding gateways are given the same coloring scheme to highlight the connection between where decisions are made and where they are used. Without loss of generality, we assume that exclusive gateways are binary, i.e., they only have two alternative outgoing sequence flows. Indeed, if a decision has n alternative outcomes, these can be evaluated by setting a proper sequence of $\lceil \log n \rceil$ exclusive gateways.

Beside decision tasks and exclusive gateways, interrupting boundary events, such as ADR of Fig. 5.1, may also represent decisions as their occurrence determines the enactment of an alternative, exception flow. However, being their triggering instantaneous, interrupting boundary events always represent points in the process where a decision is made and used at the same time.

Beside their position, decisions may be distinguished at a conceptual level based on the availability of their outcomes during process run-time. In general, a process should be guaranteed to be executable for any possible combination of decision outcomes also with respect to the temporal constraints. Since such requirement can be very strict, sometimes it is reasonable to relax it by admitting that some paths are not executable at run-time if temporal constraints cannot be satisfied. In other words, it is reasonable to reduce the possible outcomes of some decisions in order to guarantee that the process can be completed successfully.

In this regard, we propose a novel perspective aimed to conceptually distinguish decisions based on how their outcomes may be chosen at run-time.

Some decisions represent the response of the process to conditions that are dictated by the context in which the process is executed, such as data-based conditions or event occurrence. At run-time the process must always be guaranteed to run **any** of such alternative outcomes. We refer to decisions of this kind as *observations*: *A decision is called observation when the number of its possible outcomes cannot be reduced at run-time.* Task T_0 makes an observation: Physicians must determine which drugs are contraindicated based on well-documented evidence. For every possible outcome (alternative flows in \times_{2s}), the process must be executable for any possible duration of other process tasks.

Conversely, for some decisions it is possible to limit their outcomes at run-time if this can help to execute the process successfully. In this case, the choice of the outcomes is still arbitrary, but the set of possible outcomes can be reduced considering the past execution of previous elements. It must be ensured that **at least one** outcome is always allowed. To denote that the decision is guided by the limitation of its outcomes we refer to decisions of this kind as *guided decisions*: *A decision is a guided decision when its possible outcomes can be reduced at run-time to comply with temporal constraints.*

In Fig. 5.1, T_4 makes a guided decision. Since iaHA and iaPRP have similar efficacy and safety, physicians may suggest one or the other without the need for both to always be available when the decision is made. At run-time, if T_4

has been enacted 13 days after T_0 (in case that TT is triggered at day 5 and T_2 lasts 7 days), then T_6 cannot be allowed as its execution could violate the upper bound of the process duration constraint $S[5, 50]S$; therefore, T_4 can only select iaHA task.

In Fig. 5.1, symbol ? next to the task type icon denotes decision tasks that make observations, and symbol ! denotes tasks that make guided decisions. Decisions based on the occurrence of boundary events are always *observations*.

5.2.3 Controllability of a Time-Aware BPMN Process

From a temporal perspective, *executing* a time-aware BPMN process P means:

- (i) to schedule the starting time of all elements,
- (ii) to set the duration of gateways and sequence flow delays, and
- (iii) to determine which are the allowed outcomes of a guided decision before enacting the corresponding decision task.

The values of observations in P are not known in advance as they are incrementally revealed over time as decision tasks are executed. Similarly, the durations of activities are only known once the activities complete.

Therefore, a *dynamic* execution of P must *react* to observations and contingent durations in real time. A *viable* execution is one that guarantees that all *relevant* constraints—those holding in the paths being executed—will be satisfied no matter which observation outcomes and durations are revealed over time. A time-aware BPMN process with a dynamic and viable strategy is called *dynamically controllable* (DC).

5.3 Dynamic Controllability Checking

In this section, we show how to determine if a time-aware BPMN schema is DC.

First, we introduce a temporal-constraint model, called Conditional Simple Temporal Network with Uncertainty and Decision (CSTNUD) [67], that results to be a well-founded model for representing and reasoning about temporal constraints; then, we show how to verify the dynamic controllability of a process model P using a corresponding CSTNUD S_P .

5.3.1 A Short Introduction to CSTNUDs

This section recalls some concepts about temporal constraint networks, the formal definition of CSTNUD and the corresponding dynamic controllability property.

In general, a temporal-constraint network can be viewed as a graph in which nodes represent real-valued variables and edges represent binary constraints on variables. The kind of binary constraint that can be attached to edges characterizes the network and its expressive power.

For example, in a *Simple Temporal Network* (STN) [133] $(\mathcal{T}, \mathcal{C})$, where \mathcal{T} is a set of real-valued variables, called *time-points*, and \mathcal{C} is a set of binary constraints, each constraint has the form $(Y - X \leq \delta)$, where $X, Y \in \mathcal{T}$ and $\delta \in \mathbb{R}$. When it is possible to assign a value to each time-point of a STN such that all constraints are satisfied, then the STN is said to be *consistent*. By *executing* a temporal-constraint network we mean that the assignment of values to time-points is made by an (executing) engine incrementally following an execution strategy. An execution strategy determines the schedule to apply.

For example, if an STN S has a solution, the *earliest execution strategy* schedules S assigning to each time-point its earliest possible execution time.

In [157, 158], Hunsberger et al. extended STNs to include time-points and temporal constraints that apply only in certain *scenarios*, where each scenario is represented by a conjunction of propositional literals. Each time-point/temporal constraint has a *label* that concisely specifies the scenarios in which that time-point/temporal constraint must be considered. At run-time, the execution of so-called *observation time-points* unveils truth values for the corresponding propositional variables. Thus, the scenario is incrementally revealed. The result is a *Conditional STN* (CSTN). A CSTN is called *dynamically consistent* if there is a strategy for executing its time-points such that all relevant constraints will be satisfied no matter which scenario is incrementally revealed.

In [159], Hunsberger et al. proposed Conditional Temporal Network with Uncertainty (CSTNU), that extends STNs including *scenarios* [158] and *contingent links* [64]. A scenario specifies which time-points and constraints to consider during an execution and it is represented by a conjunction of propositional literals. The value of each proposition is unveiled during the execution (environment decides its value), i.e., the scenario is incrementally revealed. A contingent link is a special kind of temporal constraint having the form, (A, x, y, C) , where A and C are time-points, and $0 < x < y < \infty$. A is called the activation time-point; C is called the contingent time-point. Typically, it assumed that a contingent link is activated when A is executed. Then, the value for setting C is decided by the environment not by the engine. However, C is guaranteed to execute such that the temporal difference, $C - A$, is between x and y , i.e., the contingent link is satisfied. Contingent links are used to represent actions with uncertain durations.

In [69] the authors propose CSTN with Decisions (CSTND), a generalization of STN with scenarios (CSTN) that allows some of the propositional variables to be assigned not by the environment, but by the engine executing the network.

In [67] CSTND are generalized incorporating contingent links. The resulting network is called a CSTNU with Decisions (CSTNUD).

In the remainder of this section, we combine and extend concepts from earlier work [67, 69, 159, 158].

Definition 5.1 (Label representing a scenario). Let \mathcal{P} be a set of propositional letters. A *label* ℓ over \mathcal{P} is a conjunction, $\ell = l_1 \wedge \dots \wedge l_k$, of literals $l_i \in \{p_i, \neg p_i\}$ on distinct variables $p_i \in \mathcal{P}$. The empty label is denoted by \square and

always evaluates to true. For any label ℓ , and any $p \in \mathcal{P}$, if $\ell \models p$ or $\ell \models \neg p$, we say p *appears* in ℓ . For labels, ℓ_1 and ℓ_2 , if $\ell_1 \models \ell_2$, we say ℓ_1 *entails* ℓ_2 . \mathcal{P}^* denotes the set of all labels over \mathcal{P} .

Definition 5.2 (CSTNUD). A *Conditional STN with Uncertainty and Decision* is a tuple $\langle \mathcal{T}, \mathcal{P}, \mathcal{CP}, \mathcal{DP}, \mathcal{C}, \mathcal{OT}, \mathcal{O}, \mathcal{L} \rangle$ where:

- \mathcal{T} is a finite set of *temporal variables* or *time-points*;
- \mathcal{P} is a finite set of *propositional variables*, i.e., boolean variables;
- $(\mathcal{CP}, \mathcal{DP})$ is a partition of \mathcal{P} into *contingent propositional variables* (**observations**) \mathcal{CP} and *controllable propositional variables* (**decisions**) \mathcal{DP} ;
- \mathcal{C} is a finite set of *labeled constraints*, each of the form, $(l \leq Y - X \leq u, \ell)$, where: $X, Y \in \mathcal{T}$; $l \leq u$; $l, u \in \mathbb{R}$; and $\ell \in \mathcal{P}^*$;
- $\mathcal{OT} \subseteq \mathcal{T}$ is the set of *disclosing time-points*; and
- $\mathcal{O}: \mathcal{P} \rightarrow \mathcal{OT}$ is a bijection that associates each $p \in \mathcal{P}$ to a disclosing time-point $\mathcal{O}(p) \in \mathcal{OT}$, i.e., to a time-point that, when executed, determines the disclosure of value of the associated proposition variable. If $p \in \mathcal{CP}$, then its $\mathcal{O}(p)$ is called an *observation time-point*; otherwise a *decision time-point*.
- \mathcal{L} is a set of *contingent links* each of the form (A, x, y, C, ℓ) , where: $0 < x < y < \infty$; $A, C \in \mathcal{T}$ are the *activation* and *contingent* time-points; $\ell \in \mathcal{P}^*$; and distinct contingent links have distinct contingent time-points.

When an observation time-point is executed, the environment assigns a truth value to the corresponding observation; however, when a decision time-point is executed, the decision is assigned a truth value by the engine. $P!$ represents the decision time-point associated to decision p , while $Q?$ the observation time-point associated to observation q . As shown in [160], without loss of generality in Def. 5.2 only constraints are labeled, not time-points.

Viewed as a graph, a CSTNUD edge represents either a labeled constraint or a contingent link. In particular, each edge having the form $X \xrightarrow{\langle [l, u], \ell \rangle} Y$ represents a labeled constraint, $(l \leq Y - X \leq u, \ell)$, and it is called also *standard edge*; each edge having the form $A \xrightarrow{\langle [x, y], \ell \rangle} C$ represents the labeled contingent link (A, x, y, C, ℓ) , and it is called *contingent*. The pair $\langle [l, u], \ell \rangle$ is called *labeled range/value*. If between two time-points there exist more labeled constraints, the standard edge connecting them has more labeled ranges, one for each labeled constraint.

Example 5.2. (Conditional STN with Uncertainty and Decision) Fig. 5.2 shows a CSTNUD having 7 time-points of which 2 are contingents and 3 are disclosing time-points. Contingent link $(B!, 3, 7, E!, b)$ is activated only if decision b is true, while contingent link $(E?, 3, 5, S, \neg e)$ is activated only if observation e is false. For example, if $(B!, 3, 7, E!, b)$ is executed (b true), and it lasts 7, and the observation e results true, for executing the network without violating the constraint between $B!$ and I_2 is necessary to set decision h to false.

In [67], the execution semantics of a CSTNUD is given as a two-player game in which Pl_1 models the executing agent and Pl_2 models the environment, assumed

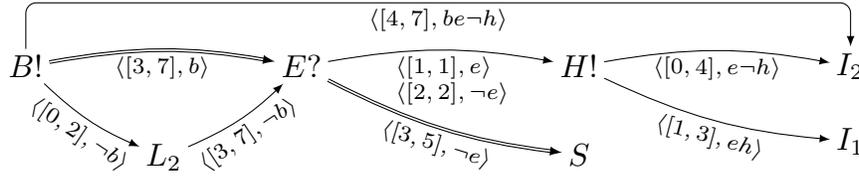


Fig. 5.2: An example of CSTN game. b, e and h are relative to $B!, E?$ and $H!$, respectively.

as the most powerful possible player. A game runs in turns: at any time instant t , there exist two turns: the P1_1 turn, $T_1(t)$, and the P1_2 one, $T_2(t)$, occurring after $T_1(t)$. At each turn, a player may decide to make k move(s), with $0 \leq k < \infty$. A P1_1 move is either the execution of a non-contingent time-point X or the assignment of a truth-value to a decision d . A P1_2 move is either the execution of a contingent time-point C or the assignment of a truth-value to an observation p . P1_2 is guaranteed to always have full information on what P1_1 has done before.

During the game, the conjunction of truth-values of propositional variables is represented by the label ℓ_{cps} . P1_1 wins the game when there are no more time-points to execute and for each constraint $(l \leq Y - X \leq u, \ell) \in \mathcal{C}$ such that $\ell_{cps} \models \ell$, then the execution times $S(X)$ of X and $S(Y)$ of Y satisfy the constraint $l \leq S(Y) - S(X) \leq u$. P1_2 wins otherwise. We denote by σ_i a winning strategy if P1_i wins the game by following σ_i .

Informally, a CSTN game has the dynamic controllability property if P1_1 has a winning strategy that is based only on the history of past moves made in the game. The history is defined in terms of *execution sequence*, the ordered sequence of executed time-points and assigned propositions [67]. Usually, $Z_1(Z_2)$ represents an execution sequence of $\text{P1}_1(\text{P1}_2)$ and $\sigma_1(Z_1, t)(\sigma_2(Z_2, t))$ represents a moved-based strategy that tells a player to make a move at time instant t only if the move is applicable at t [67].

Definition 5.3 (Dynamic Controllability [67]). A CSTN game is *dynamically controllable* (DC) if P1_1 has a winning strategy such that for any $t > 0$ and any pair of execution sequences Z_1, Z_2 , if $\sigma_2(Z_1, t') = \sigma_2(Z_2, t')$ for $0 \leq t' < t$, then $\sigma_1(Z_1, t) = \sigma_1(Z_2, t)$.

In [67] a DC checking algorithm for CSTN games based on timed game automata (TGA) [161] is proposed, while a DC checking algorithm based on constraint propagation for a sub-class of CSTN games is presented in [69].

5.3.2 Mapping Time-Aware BPMN onto CSTN game

To verify the dynamic controllability of a process model P , it is convenient to transform it into an CSTN game S_P using the transformation rules depicted in Table 5.1 and Table 5.2. Such rules are described in the proof of Theorem 5.4.

Time-aware BPMN fragment	Corresponding CSTNUD	Time-aware BPMN fragment	Corresponding CSTNUD
Start/End event 	$Z \rightarrow \quad \rightarrow E$	Task/Subprocess 	$\rightarrow A_S \xrightarrow{\langle [x, y], \ell \rangle} A_E \rightarrow$
Decision Task (observation) 	$\rightarrow A_S \xrightarrow{\langle [x, y], \ell \rangle} A_E$ $\langle [0, 0], \ell \rangle \downarrow$ $\leftarrow A_{[log\ n]}? \leftarrow \dots \leftarrow A_1?$	Decision Task (guided decision) 	$\rightarrow A_S \xrightarrow{\langle [x, y], \ell \rangle} A_E$ $\langle [0, 0], \ell \rangle \downarrow$ $\leftarrow A_{[log\ n]}! \leftarrow \dots \leftarrow A_1!$
AND Split 	$\rightarrow +s \xrightarrow{\langle [l, u], \ell \rangle} +E$	AND Join 	$\swarrow +s \xrightarrow{\langle [l, u], \ell \rangle} +E \rightarrow$
XOR Split <p>associated to p proposition.</p>	$\rightarrow +s \xrightarrow{\langle [l, u], \ell \rangle} +E$ $\langle [0, \infty], \ell p \rangle \searrow$ $\langle [0, \infty], \ell \neg p \rangle \searrow$	XOR Join <p>associated to p proposition.</p>	$\langle [0, \infty], \ell p \rangle \searrow$ $\swarrow +s \xrightarrow{\langle [l, u], \ell \rangle} +E \rightarrow$ $\langle [0, \infty], \ell \neg p \rangle \searrow$
Sequence flow edge 	$\langle [l, u], \ell \rangle$	Intermediate Event 	$\rightarrow I_S \xrightarrow{\langle [x, y], \ell \rangle} I_E? \rightarrow$
Boundary Interrupting Event 	$B_S \leftarrow \langle [0, 0], \ell \rangle A_S$ $\downarrow \langle [0, y'], \ell b \rangle \langle [x, y], \ell \rangle \downarrow A_E$ $B_E? \downarrow \langle [0, \epsilon_2], \ell b \rangle$ $B_{P_S} \downarrow \langle [0, \infty], \ell \neg b \rangle$ $B_{P_E} \downarrow \langle [w, k], \ell b \rangle$ $B_{P_E} \rightarrow \langle [0, \epsilon_3], \ell b \rangle X$	Boundary Non-Interrupting Event 	$B_S \leftarrow \langle [0, 0], \ell \rangle A_S$ $\Downarrow \langle [0, y'], \ell b \rangle \langle [x, y], \ell \rangle \Downarrow A_E$ $B_E? \downarrow \langle [0, \epsilon_2], \ell b \rangle$ $B_{P_S} \downarrow \langle [w, k], \ell b \rangle$ $B_{P_E} \downarrow \langle [0, \infty], \ell b \rangle$ $B_{P_E} \rightarrow \langle [0, \infty], \ell b \rangle$ $E_{B_{P_E}} \downarrow \langle [0, \epsilon_1], \ell \rangle$ $\langle [0, \infty], \ell \rangle \downarrow$

If not specified, the scenario is assumed to be ℓ and the temporal range of an edge is $\langle [0, \infty], \ell \rangle$.

Table 5.1: Mapping of time-aware BPMN fragments to CSTNUDs.

The obtained S_P may be checked for DC by applying one of the available algorithms for DC checking [67, 69].

The following theorem shows that the process model P results to be DC if and only if S_P is DC.

Theorem 5.4. Given a time-aware BPMN process P , there exists a CSTNUD S_P such that P is dynamically controllable if and only if S_P is DC.

Proof. Without loss of generality, we assume that all temporal ranges have the same base granularity mG . In case that P contains ranges with different time granularities, it is possible to convert them to mG .

Table 5.1 and Table 5.2 outline the mapping of the elements that can be used to transform a time-aware BPMN fragment into the corresponding CSTNUD. By

applying the proposed mappings to P , one can simply verify that the obtained S_P represents all precedence relations and temporal constraints of P .

Let us consider each mapping in detail.

– *Task/Subprocess*. Each task A is transformed into two CSTNUD time-points, A_S and A_E , representing its start and end instants. The duration range, $[x, y]$, is converted to the contingent link (A_S, x, y, A_E, ℓ) .

The label ℓ is determined considering the (possible) XORSplit/XORJoin gateways that are present in the path, Π , from the start event to A in P :

- (i) Initialize $\ell = \square$;
- (ii) For each (possible) XORSplit \times_i in Π associated to proposition p , add p or $\neg p$ to ℓ according to the branch present in Π ;
- (iii) For each (possible) XORJoin \times_i in Π associated to proposition p , remove any p literal from ℓ .

The obtained label represents the scenario in S_P where A_S and A_E have to be executed and their contingent link observed. The mapping of a subprocess onto a CSTNUD is equivalent.

– *Decision Task (observation)*. The conversion is analogous to the one of a task as for its duration attribute. As regards the observation made, it is necessary to represent all the possible outcomes by adding to S_P a suitable number of observation time-points. In particular, if an observation of a decision task A in P can assume n distinct values, then, in S_P there must be $\lceil \log n \rceil$ propositions, associated to new $A^?_1, \dots, A^?_{\lceil \log n \rceil}$ observation time-points. In this way, each A outcome is represented by a proper combination of truth-values of such $\lceil \log n \rceil$ propositions. $A^?_1, \dots, A^?_{\lceil \log n \rceil}$ are added in sequence after the CSTNUD time-point A_E . The temporal distance between $A^?_1, \dots, A^?_{\lceil \log n \rceil}$ and A_E is set to 0 to constrain that the observation values are available at the same instant in which A_E is executed.

– *Decision Task (guided decision)*. The conversion is analogous to the one of a decision task making an observation. In this case, however, the possible outcomes of a decision task are represented using decision time-points instead of observation ones to capture the semantics associated to guided decisions.

– *ANDSplit/ANDJoin gateways*. The conversion is analogous to the one of a task. In this case, however, duration attribute $[x, y]$ is converted to a standard edge as gateways are executed by the process engine.

– *XORSplit/XORJoin gateways*. The conversion is analogous to the one of an ANDSplit/ANDJoin as regards its duration attribute. As for scenario, if ℓ is the scenario in which a XORSplit is present, then all converted elements located in its *true* outgoing flow will have a label entailing ℓp , while all converted elements located in its *false* outgoing flow will have a label entailing $\ell \neg p$, where p is the proposition associated to the considered XORSplit. In case of XORJoin, the process is reverse: the scenario label is updated removing p literal.

– *Sequence Flow Delay*. A sequence flow edge having temporal delay $[l, u]$ is converted to a standard edge having $\langle [l, u], \ell \rangle$ as labeled range.

– *Intermediate Event*. Since the temporal range of an intermediate event represents the waiting time allowed for event triggering, its mapping is analogous to the one of a task. Moreover, the end time-point $I_E?$ is also an observation time-point associated to a proper proposition i for representing if the event occurred (true value). In this way, the semantics of the CSTNUD fragment is the following: the execution of $I_E?$ reveals if the event occurred or not and, in case of occurrence, the execution time of $I_E?$ is the exact instant in which the event occurred.

– *Boundary Interrupting Event*. This conversion can be split in two parts that work in parallel, one for task A and one for interrupting event B . $y' < y$ as required by BPMN [11]. Task A and event B are mapped using the previous mappings for tasks and intermediate events, respectively. The contingent links associated to A and B must start at the same time: in Table 5.1, B_S is at distance 0 from A_S . In S_P , after $B_E?$, there are time-points and constraints related to the temporal characterization of subprocess BP (representing exception handling) that are labeled by ℓb to represent the fact that they must be considered only in case B occurs. Both A_E and BP_E are connected to time-point \times that represents the original exclusive gateway \times assumed instantaneous for simplicity.

Since the value of an observation cannot be constrained in any way, the obtained CSTNUD fragment allows also the representation (and reasoning) of cases that are not possible in a real process run.

For example, in the CSTNUD it is possible that (i) the contingent link associated to B lasts more than the contingent link associated to A , (ii) the proposition b is set true and, therefore, (iii) all temporal constraints associated to the interruption branch must be observed. But this case cannot occur in a real run of P because all interrupting events must occur prior to task completion [11]. Therefore, the dynamic controllability of this CSTNUD fragment guarantees the dynamic controllability of the original fragment containing Boundary Interrupting Event even for execution cases that can never occur.

On the other hand, since it is necessary to guarantee the controllability for any possible combination of task duration and event occurrence, all real cases are simpler cases of the real worst case in which A completes at its maximum and B occurs at the last possible instant and BP completes at its maximum, that it is captured by the this conversion. In case that there exist some relative constraints (explained below) involving the start/end of task BP or the end of task A , in their corresponding CSTNUD edges, labels have to be adjusted considering literals $b/\neg b$ to guarantee that the edges are considered in the right scenarios. For example, in the edge associated to a relative constraint involving the end instant of A , the label must contain also $\neg b$ because the constraint has to be considered only when A is not interrupted.

– *Boundary Non-Interrupting Event*. The conversion is analogous to the one of a boundary interrupting event. In this case, however, there is a small complication given by the fact that the BPMN semantics dictates that, in case of event

Time-aware BPMN fragment	Equivalent CSTNUD	Time-aware BPMN fragment	Equivalent CSTNUD
Start(A)–Start(B) 		End(A)–End(B) 	
Start(A)–End(B) 		End(A)–Start(B) 	

Tasks A and B can be in different paths but not in not-compatible branches.

Table 5.2: Mapping of relative constraints between BPMN elements to CSTNUDs. For brevity, only tasks are exemplified in the table, but it is possible to consider any pair of elements.

occurrence, task A (cf. Table 5.1) must wait the completion of event handler represented by BP for reaching state “complete” [11].

Thus, to properly represent such temporal constraint in the CSTNUD, it is necessary to add a join time-point \dagger after A_E . \dagger has to be executed after A_E and BP_E in case the event occurred (proposition b is true) and immediately after A_E in case the event did not occur. Such behavior is ensured by associating two temporal ranges to the edge $A_E \rightarrow \dagger$: $\langle [0, \infty], \ell b \rangle$ represents the delay to observe in case the event occurred (b is true), while $\langle [0, 0], \ell \rangle$ represents the 0 delay in case the event did not occur. After \dagger , there is the delay $\langle [0, \epsilon_1], \ell \rangle$ corresponding to the sequence flow delay associated to the flow outgoing of A in the BPMN fragment.

– *Relative Constraints.* Let us consider a relative constraint $\langle I_F \rangle [l, u] \langle I_S \rangle$ between two elements A and B , where I_F and I_S represent the kind of instants to be considered, i.e., S or E. In Table 5.2, A and B are tasks for space reasons, but they can be any combination of tasks, subprocesses, gateways and events. A relative constraint is converted to a CSTNUD edge between the time-points associated to instants A_{I_F} and B_{I_S} with labeled range $\langle [l, u], \ell \rangle$. If ℓ_A/ℓ_B is the scenario where A/B are mapped, then ℓ must satisfy $\ell \models \ell_A \ell_B$, i.e., relative constraints can be defined only in consistent scenarios.

As introduced above, a time-aware BPMN process is dynamically controllable if it is possible to execute it by satisfying all relevant constraints while reacting in real time to (i) the observation values that occur, (ii) tasks/subprocess durations, and (iii) event occurrences.

According to the two-player semantics of CSTNUDs, a CSTNUD is dynamically controllable if it is possible to execute it in a way such that, no matter how the execution of any contingent link turns out and any observations turns out (PL₂ execution strategy), it is possible to set a sequence of decisions and to

schedule all non-contingent time-points in real time (Pl_1 strategy) satisfying all relevant constraints.

Considering the provided mapping and, in particular, the mapping of process fragments containing boundary events, it is a matter of definitions to verify that, given a process diagram P and its corresponding CSTNUD S_P , the dynamic controllability in S_P implies the dynamic controllability in P and vice-versa. \square

5.4 Conclusion

In this chapter, we presented a novel extension of time-aware BPMN where events can be temporally characterized and decisions are distinguished into two types, based on how the range of possible decision outputs may be restricted at run-time to satisfy temporal constraints.

As for temporal characterization, we proposed to distinguish between durations that can be limited by a process engine and durations that become known only at run-time. Regarding decisions, we proposed that they can be made and used in different points in the process to allow a greater flexibility with respect to temporal constraints. Moreover, they can be of two kinds: observations, for which the process has to be guaranteed to run for all possible outcomes, and guided decisions, for which the number of possible outcomes can be reduced at run-time to ensure a successful execution.

At application level, it is important to have processes that can be executed reacting to already executed activities, event occurrences and past decisions; we formalized this property as dynamic controllability for time-aware BPMN processes. For verifying whether a BPMN process is dynamically controllable, we propose to map it onto a corresponding CSTNUD instance, whose dynamic controllability is likely to be checked by algorithms that go beyond the commonly adopted model-checking approach based on TGA.

Related Work

The specification and management of temporal constraints has become a key aspect in the context of business process modeling and management, as constraining time is vital for reliable process actualization and execution [16, 17, 124].

Various proposals have addressed temporal constraint modeling within the research communities of workflows [16, 17, 23, 29, 33, 124, 125, 162, 130, 163] and BPM [34, 35, 36, 52, 53, 119, 164], drawing inspiration from approaches grounded in the formal principles of timed automata [31, 32], time Petri nets [65, 165], and temporal constraint networks [133, 157, 158, 159, 134] both used for the verification of timed (process) models. This chapter describes relevant contributions to the field of temporal constraint modeling and checking in business processes.

Sect. 6.1 discusses selected relevant approaches belonging to the field of BPM, specifically focusing on proposals based on the BPMN standard [11] and related extensions. Sect. 6.2 compares the contributions introduced in Chapter 3, Chapter 4, and Chapter 5 with relevant state-of-the art approaches.

6.1 Temporal Constraints: Modeling and Management

Early contributions to the representation of temporal information and temporal constraints are summarized in [16, 124]. The authors identify issues related to the poor consideration of temporal aspects in workflows. They reason on the effects that time violations have on business process costs and explain how proper time management may increase organizational competitiveness and improve timely reaction to external events and changes [127].

Focusing on handling globally distributed business processes, in [163] the authors underline the need of incorporating time-dependent factors, such as temporal order and time differences, into the logic of process activities at build-time. In detail, they consider activity duration and multiple time axes to represent different time zones, and define time constraints and process routing in terms of restrictions on starting and ending times of activities. Last but not least, a

method for checking both the build-time and run-time consistency of the proposed time workflow model is presented.

In [125], the authors introduce a temporal model for conceptually designing clinical workflows, by addressing the representation of activity duration, delays, periodic and absolute constraints, and inter-activity constraints such as relative constraints and absolute delays. The authors discuss the verification of the modeled constraints and propose a prototype based on YAWL [84], called Temporal Workflow Analyzer, for supporting workflow modeling and time management at design time. TNest, a new structured workflow language providing full support of temporal constraint specification during process design, is presented in [162]. The authors consider two main kinds of temporal constraints, namely activity durations and relative constraints, and tackle different nuances of the temporal workflow controllability concept, which was firstly introduced in [29, 130].

A major contribution to the formal specification and operational support of the temporal perspective in business processes is described in [17, 23, 33].

In [17], the authors identify a set of ten time patterns to ease the comparison of process-aware information systems (i.e., information systems that provide process support functions and separate the process logic from applications) and foster the choice of appropriate time constraints at design time. Furthermore, in [23] the authors provide an evaluation on the presented time patterns of selected process modeling approaches coming from both industry and academia, such as, among others, the BPMN [11] and BPEL standards, and those presented in [16, 125, 163].

In [33], a formal temporal semantics is defined in order to avoid ambiguities and ease the practical use of time patterns proposed in [17, 23]. The authors exhaustively explore various aspects of the temporal perspective in business processes, by analyzing the identified patterns with respect to multiple design features and considering both process and time granularity. The studied temporal patterns have been extracted from a rich benchmark of business processes, mostly retrieved from the healthcare domain. BPMN is found among the approaches that have systematically been reviewed with respect to temporal pattern support and expressiveness, as it will be later discussed. The authors summarize their previous efforts by providing a complete picture of their work: a formal semantics is discussed extensively for each time pattern and the ATAPIS Toolset is used for supporting the design, implementation, and verification of those temporal patterns mostly used in practice. ATAPIS is also used in [127] to implement change operations that allow modifying the temporal constraints of a time-aware process and to check the soundness of the changed processes.

Management of time constraints on activities that are contained in loops is addressed in [141]. The author highlights the need for improving the formulation of time constraints in workflows with loops and chooses timed workflow graphs (TWfG) to check constraint satisfiability.

The definition, modeling, and management of temporal constraints encompasses the concept of *temporal constraints violation*. In [27], the authors propose an approach for managing controlled violations of time constraints in temporal

workflows. Temporal constraints are expressed by means of formal expressions. However, not all constraints need to be strictly observed as the relaxation of some of them is allowed, provided that an associated penalty is applied.

Another aspect that is closely related to the one of temporal constraint violation is that of *predictable exception*, that is, a deviation that is known to possibly appear between what is planned and what is actually happening. In [28], the authors consider a particular kind of predictable exception, namely deadline escalation, which arises when an organization is not able to meet the deadlines for one or more instances of a process model. To improve the reaction to such predictable (temporal) exceptions, the authors propose a set of deadline escalation strategies to support decision-making and minimize tardiness.

6.1.1 Expressing Temporal Constraints in BPMN

The expressiveness of the Business Process Model and Notation [11] with respect to the modeling of temporal constraints has been addressed by some research proposals [31, 32, 34, 35, 119, 153, 164, 166, 167], most of which aimed to extend the standard in order to improve the specification and formal verification of temporal aspects.

Time-BPMN [34] is an extension proposed to capture the temporal perspective of business processes modeled with BPMN 1.2 [168], the previous version of the standard. The aim of the introduced extension is to simplify the representation of temporal constraints and dependencies that are vital for real business process enactment. Graphical markers are specified to control the start and end of activities and temporal constraint attributes are used to detail the targeted activity, the constraint kind, or additional useful documentation. Although activity start and finish times can be specified as inflexible constraints, the modeling of activity duration is not explicitly addressed. The weakness of BPMN to represent the temporal dimension of a process is compared to the expressiveness of project planning tools in [166]. In particular, the inability to visually represent the temporal execution order is highlighted, and the need for representing task duration is observed. BPMN process orchestration is analyzed and a preliminary representation of tasks with fixed duration is presented.

In [35], the authors extend BPMN 2.0 in order to enable the specification of temporal constraints and to verify potential violations by means of model checking approaches. In detail, the authors propose a graphical decorator depicting the minimum and maximum desired duration limits for an activity. Similarly to [34], constraint attributes are introduced to regulate the behavior and strength of the expressed duration constraints.

A BPMN extension designed to handle temporal constraints besides resource and concurrency ones is presented in [32]. The authors propose a mapping from BPMN elements to timed automata to verify business processes and avoid design time and exceptions related to temporal and resource aspects. Flow objects are mapped onto timed automata, relations among them correspond to synchronization patterns between such automata, and process constraints are expressed

as invariants, guards, and assignments on the timed automata. Regarding task duration modeling, attributes are added to BPMN tasks in order to explicitly specify their minimum and maximum execution times. A similar verification approach based on timed automata is adopted also in [31] and applied to the set of temporal constraints introduced in [35].

Overall, the approaches introduced in [31, 32, 35] focus on translating BPMN into timed automata for constraint verification. As a result, the modeling of temporal constraints in BPMN is mostly based on [34] and temporal aspects are expressed in terms of non-standard attributes and graphical decorators.

In [153] the authors propose an encoding of timed business processes into the Maude language, for automatically verifying some properties, yet considering a simpler extension of BPMN where only task timeouts and sequence flows delays can be expressed.

In [167], BPMN is extended for supporting Business Activity Monitoring, that is, the real-time monitoring and control capabilities during process runtime. Specifically, with respect to activity duration, a meta-model is proposed to measure the time span that goes from the moment the activity is assigned to a resource to the moment it is concluded.

Finally, a recent extension of BPMN considers raising the modeling of temporal control structures, such as temporal loops and temporal XOR-splits, at a conceptual level [119]. Temporal loops are associated to conditions that specify an upper temporal bound in addition to a regular loop-condition. That is, a loop iteration is executed if (i) the regular loop-condition holds and (ii) the time elapsed from the start of the process and a given time-point is less or equal than the specified upper temporal bound. Similarly, temporal XOR-splits compare the time elapsed from the start of the process and a given time-point with a constant and consequently route the process flow.

A novel relevant meta-model for the integration of temporal aspects in BPMN processes is presented in [36]. The authors provide a decorator-based extension to the standard, where each constraint is expressed by using BPMN constructs. However, models proposed for activity duration are utterly different from those presented in Chapter 3, mostly due to the assumed semantics of event-based gateways [169]. Moreover, whereas the authors in [36] constrain activities to be executed within a fixed or flexible duration, we separate constraint specification from violation management and consider different levels of flexibility in both these aspects.

In [164], the authors propose a method to verify the controllability of time-aware business processes that consider constraints over activity duration. The approach suggests to specify both the structure and operational semantics of a process in terms of Constrained Horn Clauses (CHC). Then, also the notions of weak and strong controllability are encoded in CHC. Finally, two novel algorithms to solve the related strong and weak controllability problems are designed in order to deal with the computational cost given by nested universal and existential quantifiers.

6.2 Discussion

In this section, we frame the contributions introduced in Chapter 3, Chapter 4, and Chapter 5 with respect to related literature.

In detail, we start by reviewing the duration constraints described in Chapter 3 and, then, consider more formal approaches relying on timed automata and temporal networks for process verification and review the contribution of Chapter 4 and Chapter 5.

6.2.1 Modeling Duration constraints in BPMN

Compared to the described approaches, in Chapter 3 we propose structured BPMN process models to be (re-)used as a base building block for specifying different alternatives of activity duration. Table 6.1, compares the contribution of Chapter 3 with selected approaches, considering both the kinds of addressed temporal constraints and the final research goal (e.g., run-time evaluation of temporal constraints, formal verification).

Following the results presented in [33], we focused on evaluating the suitability of elements defined in the BPMN standard to represent duration constraints, by defining different processes called *duration patterns* and showing how to suitably combine them. Motivated by real case studies, we also considered the effects of external events on activity execution. We distinguished three main kinds of duration constraints introduced in Sect. 3.3, 3.4, and 3.5 respectively, classified according to how the constraint is applied and measured. In detail, beside simple duration for activities, activities with boundary events, and process regions, we proposed a design solution that allows one to dynamically choose an activity duration range out of more. The choice may depend on a condition that is either evaluated at a moment in time preceding the activity starting instant or it can be taken after initiation (deferred duration). Finally, modeled shifted duration ranges apply only to part of a running activity under certain conditions.

The core duration pattern ϕ_{simple} is able to capture simple duration of activities, SESE regions, and activities with attached boundary events, the latter ones with slight amendments in the way ϕ_{simple} is connected to the task and the exception flows. ϕ_{simple} is also the starting point for the processes for managing the duration of non-SESE regions (i.e., ϕ_{mSucc}) and shifted durations (i.e., $\phi_{shifted}$ and $\phi_{shiftRes}$), which require ϕ_{simple} to be adapted and enriched to deal with these more complex case. Instead, deferred durations are specified through a more complex duration pattern $\phi_{deferred}$ in order to “synchronize” the specification of more duration ranges. However, connecting kits Ck_1 connecting ϕ_{simple} to an activity, Ck_3 connecting ϕ_{simple} to a SESE region, Ck_5 connecting $\phi_{deferred}$ to an activity, and Ck_6 connecting either $\phi_{shifted}$ or $\phi_{shiftRes}$ to an activity contain the same number and kinds of elements.

As for simple duration specification, the main difference with the proposals presented in [32, 35, 36] is the way we deal with duration specification. Designing dedicated processes for the management of duration violations allows us to

	The approach presented in Chapter 3	Modelling Processes with Time-Dependent Control Structures H. Pichler, J. Eder, M. Ciglic	Toward a Time-centric modeling of Business Processes in BPMN 2.0 S. Cheikhrouhou, S. Kallel, N. Guernouch, M. Jmaiel	Time-BPMN D. Gagné, A. Trudel	Formal Verification of Business Processes with Temporal and Resource Constraints K. Watabiki, F. Ishikawa, K. Hiraishi	A metamodel to integrate business processes time perspective in BPMN 2.0 C. Arevalo, M.J. Escalona, I. Ramos, M. Dominguez-Muñoz	Temporal Specification of Business Processes through Project Planning Tools C. Flores, M. Sepúlveda
	[52, 53]	[119]	[35]	[34]	[32]	[36]	[166]
Simple Duration Constraint of Activity	✓	✓	✓	✓	✓	✓	✓
Simple Duration Constraint of Process Region	✓	-	-	-	-	-	-
Deferred Duration Constraint	✓	-	-	-	-	-	-
Shifted Duration Constraint	✓	-	-	-	-	-	-
Inter-Activity Temporal Constraint	-	-	✓	✓	-	✓	✓
Temporal loops	-	✓	✓	-	-	✓	-
Temporal Constraint correlated with resource/data constraints	-	-	✓	-	✓	-	-
Definition of Semantics	✓	-	-	-	✓	✓	-
Run-time Constraint Evaluation	✓	-	-	-	-	-	-
Extended BPMN Notation	✓	✓	✓	✓	✓	✓	✓
BPMN-compliant extension	✓	-	-	-	-	✓	✓
Formal Verification of time constraint	-	✓	✓	-	✓	✓	-

Table 6.1: Comparison between different approaches (✓: The considered approach supports the feature).

specify the constraints in a more flexible way, that is, by allowing the activity to execute regardless of the defined constraints. Instead, in the approach introduced in [36], graphical decorators are used to encode a “strong” duration semantics that enforces the activity to observe the constraints (both fixed and flexible durations), as interrupting boundary timer events are used for expressing duration. Moreover, it is not clear if and how duration of process regions may be expressed with the approaches presented in [32, 35, 36], especially for non-SESE regions. Only the concept of “combined duration” introduced in [27] may be close enough to express the duration of a process region. The choice of using graphical decorators is a design choice: we enclosed duration patterns within subprocesses to

avoid using non-standard icons, but we could have as well employed different kinds of icons to denote activities having constrained duration.

Deferred duration and shifted duration are novel concepts, since in both cases duration depends on the occurrence of some event related to activity execution. In real-world processes, they are more typical of activities that unfold in time and are described at a quite high level of abstraction.

In [35] temporal constraints correlated with data constraints are exemplified by associating two different duration ranges to one activity, one of which is chosen depending on some data-based condition. However, the authors do not detail when the choice is made and what happens if conditions change while activity is being executed.

Drawing inspiration from [23], when dealing with activity duration we do not consider the time span between the beginning of the activity and its real activation, that is we measure the time span between its start and end events. We also assume that sequence flows, gateways, and events consume a fixed or null amount of time and that any eventual data required to start the activity is available and its evaluation does not delay the execution. According to the evaluation results presented in [23], BPMN does not *directly* support minimum and interval duration specification, whereas maximum duration can be partially supported by non-interrupting timer events attached to the activity's boundary. Additionally, in [23] the authors highlight that there is no means to model the start time of a BPMN activity.

In this setting, while acknowledging the importance of improving the temporal perspective of BPMN we show how elements already in the notation can be combined to capture interesting duration properties. Despite the obtained models require some design effort, being able to express duration entirely in BPMN brings several advantages. First of all, the semantics of the constructs is already defined in the notation and, therefore, existing BPMN software can be used directly to analyze and execute temporal process models. Then, the proposed process models are meant to be transparent to process designers, that is, expert modelers can see how constraints are specified and handled, and they have the freedom to refine and tune them according to their needs. Last but not least, this modular approach enhances readability and decomposition of complex process models and provides multiple design levels that are suitable for different stakeholders. Moreover, by avoiding adding artifacts and text annotations, we reduce the probability of adding errors to the process.

The duration patterns proposed in Chapter 3 do not aim to cover the whole span of existing temporal relations that hold between any two process activities, such as those captured by Allen's interval-based temporal logic [131]. Indeed, by focusing on the modeling of duration constraints Chapter 3 does not consider other important temporal constraint categories, such as time-lags between activities, restricting execution times, variability, and recurrent process elements [17]. For instance, we are not able to capture time-lags such as "patients reaching the operating room for appendectomy directly from ER should receive prophylactic antibiotics within a 60-minute window before the initial incision". However, the

specification of duration constraints of process regions (discussed in Sect. 3.3.2) partially considers time-lags between activities, as it focuses on the temporal dependencies between the starting and ending nodes of the region. Time-lags are explicitly considered in [34, 35, 36]. As for restricting execution times, flexible and inflexible time constraints are considered in [34, 35], which partially express the concept of schedule restricted element defined in [17]. Recurrent process elements are still poorly supported by existing approaches. Temporal loops and temporal constraint over cardinality are addressed. In [35, 36] the authors define temporal constraint over cardinality to limit the number of iterations of a loop activity within a certain time frame. Instead, the temporal loops proposed in [119] incorporate temporal aspects within the loop condition, to limit the number of loop iterations depending on certain temporal conditions. Duration pattern ϕ_{simple} , introduced in Sect. 3.3, can be used to specify the duration of complete loops forming SESE regions.

Although relying on time Petri nets [65] to check the soundness of the duration patterns proposed in Chapter 3, we abstain from dealing with the formal verification of the specified duration constraints. Indeed, temporal constraint verification requires using other approaches, such as those proposed in [31, 32, 77, 153, 164] and discussed in the following section.

6.2.2 Time Petri Nets, Timed Automata, Timed Game Automata and Temporal-Constraints Networks

The formal verification of business process models encompasses correctness checking, and process compliance and variability [170]. In general, processes are verified against various properties, such as structural properties (e.g., bottlenecks and deadlocks), reachability properties, user-defined temporal constraints, and controllability [18].

Several techniques exist to verify and reason on timed systems. In the fields of workflows and BPM, time Petri nets [65] and timed automata [66] appear to be the most used [31, 32, 77, 78, 79], but temporal constraint networks are also gaining notable attention [30, 69, 158, 161].

Timed automata and bounded time Petri nets enlarged with strict constraints are equivalent with respect to timed language equivalence [171]. Therefore, choosing one formal model over another depends on the final goal and on the characteristics of the timed system being modeled.

Traditionally, time Petri nets are more suitable for modeling workflow management system [172] and important soundness verification results have been formulated on workflow nets [77]. Moreover, time Petri nets are exponentially more concise than timed automata, which lack of high-level composable graphical patterns to support systematic designs for complex timed systems [172]. However, current TPN tools (e.g., Romeo [121] and TINA [122] are relatively new, have a limited number of constructs and lack hierarchical modelling features.

In Chapter 3 we chose time Petri nets since our goal was not to verify temporal constraints, but rather check the soundness of the nets obtained from the

proposed BPMN models. Choosing Petri nets allowed us to exploit existing mappings from BPMN to Petri nets [73, 79] and results about workflow nets verification [77]. Also, the simulation features available in Romeo [121] and TINA [122] were sufficient for checking the liveness and boundedness properties of the time Petri nets we modeled.

Instead, since in Chapter 4 our main goal was to pave the path towards the verification of temporal constraints in BPMN processes, we chose timed automata as more tools exist to support this task [172].

Networks of timed automata are often used for checking quality and properties of process models [31, 32, 153]. Roughly speaking, a timed automaton is made of a (i) finite automaton-based structure, (ii) a set of clocks, and (iii) timing constraints and clock resets on transitions.

As mentioned in Sect. 6.1.1, the approaches introduced in [31, 32, 153] propose to enhance BPMN with temporal constraints and to map the obtained process models to timed automata for model checking purposes. Both the approaches presented in [31, 32] use the UPPAAL model checker [123].

Compared to the latter approaches, the work presented in Chapter 4 stands out for the use of BPMN for modeling temporal constraints instead of extending BPMN. Indeed, (i) in Chapter 4 temporal constraints are represented directly in BPMN and, more specifically, by making use of signal events exchanged between process branches to achieve synchronization. Despite giving rise to complex and detailed process diagrams, this approach lays the foundations for the channel-based communication realized by the automata of the network.

Timed Game Automata have been introduced in [173] as an extension of timed automata where the set of transitions is partitioned into controllable and uncontrollable ones. An extension of UPPAAL able to solve games based on timed game automata with respect to reachability and safety properties is presented in [174].

The application of TGA for process model verification has been considered by research approaches in the field of temporal constraints networks [67, 161].

As introduced in Chapter 5, temporal constraint networks [133] are directed graphs whose nodes represent time points and whose arcs represent distance constraints between time points.

A concept that bridges the world of temporal constraint networks with the one of time-aware business processes is that of *dynamic controllability*. In Chapter 5, we recalled the notion of dynamic controllability for Conditional Temporal Networks with Uncertainty and Decisions (CSTNUDs) [67, 69] and proposed a mapping from BPMN enhanced with time constraints to CSTNUDs.

In general, the adoption of the CSTNUD model seems to be the more promising for checking a time-aware BPMN process considering techniques different from timed automata. Indeed, while in [67] the author proposed a first CSTNUD DC checking algorithm based on TGA (The decision problem of reachability-time games in TGA with at least two clock is in EXP [175]), in [69] the problem of checking the DC of a CSTNUD was proven to be PSPACE-complete. Moreover, the authors proposed more efficient algorithms for checking the DC of some sub-

classes of CSTNUDs and this efficiency seems to be preserved for more general CSTNUD instances [69]. In addition, timed automata lack of composable graphical patterns supporting the specification of high-level requirements for complex systems design [172], whereas temporal constraint networks have been previously used to model temporal [29, 157, 158, 159], decisional [67, 69], and organizational aspects [176].

The approach presented in Chapter 5 is general and can be extended to other BPMN elements. For example, delays can be easily applied also to message flows while the concepts of duration and relative constraints can be applied to loop activities with some conditions. Moreover, we could easily include absolute temporal constraints. It would simply require to extend the mapping shown in Table 5.1.

The Informational Perspective

*Processes and data are two sides of the same coin
that must be tightly integrated [38].*

The crucial role played by data in business process design, implementation, execution, and analysis is gaining considerable attention within the BPM and database communities [74, 177].

Both the connection between processes and persistent data managed by organizational database systems and the notion of *data-aware* process modeling have been investigated by recent studies in BPM considering both data-centric [39, 42] and activity-centric modeling paradigms [45, 19, 56, 83].

As previously mentioned, process models encompass a wide range of related perspectives, each one contributing to build the information value of an organization. A crucial role is played by process-relevant (or process-related) data, which, are continuously generated, consumed, and manipulated by process activities and evolve during execution, often affecting the flow of the process and the quality of its outcomes.

The connection between business processes and data is not a new concept in BPM [42, 19, 74, 83, 178]. In traditional process models, the information perspective is captured by data objects that are read and written by activities [14].

However, data management aspects remain “hidden” within process models. Indeed, data operations, their relationships, and their effects on underlying information systems are hardly captured by (conceptual) activity-centric process models [43, 45].

Persistent data needed for process execution are usually managed by enterprise databases, which are likely to serve many applications and processes, having different scopes and goals. Indeed, the shift towards process-aware information systems is quite recent and, in the past, processes had to adapt to existing information technology [179].

In this context, linking data to business processes is a twofold challenge. On the one hand, it is important to provide a conceptual model that captures the connection between process and data models. Indeed, visualizing the interplay among processes and data reduces error-proneness and overthrows entrance barriers for non-IT users, thus allowing process experts and stakeholders to collectively create the process models. On the other hand, such model must guarantee the data consistency of the process with the database, at different levels of abstraction [42].

In this part, we tackle the informational perspective of business processes by proposing an approach to support the conceptual integration of processes and data. Our proposal relies on the novel concept of *Activity View* and aims to

assist expert process designers in representing informational aspects of business processes and in easing the exploration of intertwined process and data models.

Chapter 7 presents the Activity View as a conceptual approach to link BPMN process models to database schemata represented through UML class diagrams, and introduces a framework based on the principles of relational data modeling aimed to explore and query linked process and data models. Chapter 8 describes how the Activity View may be used to detect possible inconsistencies that may arise between linked process and data models. Chapter 9 describes in detail the experimental evaluation of the Activity View, which was conducted to have an estimate of its usability in practice. Finally, Chapter 10 introduces related work and compares our contribution with relevant state-of-the-art approaches in the field of BPM.

Conceptual Modeling of Processes and Data: Connecting Different Perspectives

This chapter is based on results published in [57].

Despite being widely accepted that processes and data are “two sides of the same coin” [38], these two assets are still conceived separately in most organizational realities [74].

On the one hand, traditional activity-centric process modeling languages such as BPMN [11] focus on modeling the control flow of a process by emphasizing the role of activities and their dynamic behavior. BPMN embraces all the phases of the business process life-cycle, being suitable for creating high-level process models needed during process design and analysis as well as for defining very detailed process models required during enactment and monitoring [1].

On the other hand, database design consists of three consolidated and distinct phases, namely conceptual, logical, and physical design [180]. At each design phase, modelers make use of different data models and related schemata to capture the aspects of interest at a particular level of abstraction. At the highest level, conceptual data models are used to create conceptual schemata that concisely represent how the information of interest for a specific domain is organized.

In this scenario, the connection between processes and data is often handled programmatically by process developers, thus contributing to increase the conceptual gap between processes and data [45, 177]. Indeed, whereas in the field of database design conceptual modeling covers a crucial role, activity-centric process diagrams provide little detail about the structure and semantics of conceptual data related to a process.

However, especially during high level conceptual modeling, designers may not disregard the crucial role of data. Indeed, at every stage of the business process life-cycle, considering the data perspective of a process enriches its representation and improves understanding of the way operational work is conducted [19].

Being close to the human perception of the represented domain, conceptual approaches bring many advantages to process designers: they foster the represen-

tation of processes and related data, support conceptual reasoning, and improve system flexibility in terms of preventively detecting issues and data inconsistencies that may arise during implementation [56].

The contribution of this chapter is twofold. In Sect. 7.1, we address the problem of connecting processes and data at the conceptual level by using BPMN [11] to represent processes and UML Class Diagrams [63] to model conceptual database schemata. BPMN is used at a level of abstraction tailored to meet the one of conceptual data modeling, that is, we consider dealing with detailed process models showing all the steps and complexities understandable by designers, but not having implementation details.

More specifically, we propose a novel *Activity View* aimed to capture the connection between a BPMN process model [11] and a UML class diagram [63] representing the conceptual schema of a database storing information related to the application domain in which the process is executed.

Activity Views are meant to support process designers in conceptually modeling the operations performed by process activities on persistent data stored in a database and to enable basic reasoning on the interplay between a process and a related database. Our approach is based on existing modeling standards to avoid defining yet another conceptual model and to ease the mapping of devised concepts to known (logical) frameworks [180]. Indeed, sitting in-between process models and database schemata, the Activity View provides a novel connected perspective, while leaving the original models untouched.

Then, in Sect. 7.2, we propose an approach based on the Activity View to define, explore, and query the connection between process and data models. Our aim is to provide a uniform representation of (i) the structure of a process model, (ii) the structure and content of a related conceptual database schema, and (iii) the data operations connecting them, and to explore properties of process and data models that stem from this connection.

The core of the proposal presented in Sect. 7.2 relies on a relational database approach [181]: we use a suitably extended relational data model to formally represent and reason on the information described by (i)-(iii), propose an algorithm to manage some challenging aspects related to representing information about the structure of a process model, and formalize queries to unravel the connection between process and data models. Eventually, by allowing designers to reason about the operational work performed by process activities under a data perspective, our approach may foster process understanding, improvement, and re-engineering.

7.1 Bridging the Gap between Processes and Data

The main novelty of this section is the formalization and experimental evaluation of the *Activity View* that can be used (i) to support the specification of data operations during process modeling and re-engineering, and (ii) to provide inter-

esting insights on the interplay between process models and conceptual database schemata.

The remainder of the section is organized as follows. Sect. 7.1.1 provides the motivation for our approach. Sect. 7.1.2 introduces and formalizes the Activity View, while Sect. 7.1.3 describes how our proposal fosters new conceptual insights.

7.1.1 Motivations and Open Research Questions

In this section we introduce a sample scenario to motivate our approach, starting from the BPMN [11] and UML [63] standards, both currently managed and maintained by the Object Management Group.

Let us consider the procedure conducted by a professor to examine and grade students, as the one represented by the simple BPMN process of Fig. 7.1(a) and described in the following example.

Example 7.1 (Student Examination and Grading). The process begins with a start event s , followed by activity *Check attendance*. To check student attendance, the professor must compare the matriculation numbers of the attending students with the data retrieved from the exam registration repository. The latter one is a source of persistent data and is represented by a *data store* named *DB*. Data stores are connected to one or more activities through directed

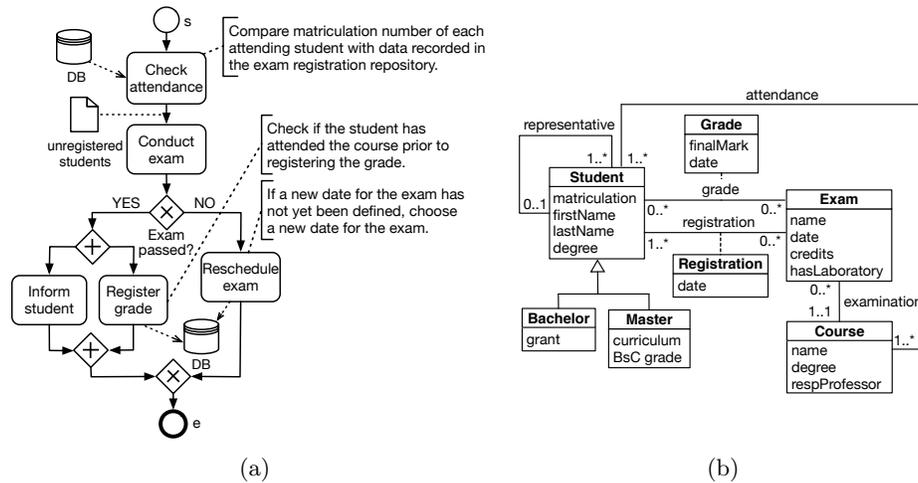


Fig. 7.1: (a) BPMN process diagram showing the actions performed by a professor to examine students. (b) UML class diagram of the exam registration repository. This figure shows that currently there is no formal relationship between a process model and the data model of the accessed database, hence providing a motivation for our work.

data associations. The attendance of **unregistered students** is also recorded and this volatile information is passed along to activity **Conduct exam** as a *data object*. Then, each student is examined individually and, once the exam is concluded, the professor decides how to proceed: This decision is represented by exclusive gateway **Exam passed?** that splits the flow into two branches. For those students who failed, the exam is rescheduled, whereas, only for those who passed the grade is registered in the repository. While registering the grade, the professor informs the student about the result. Hence, activities **Inform student** and **Register grade** are executed in parallel, as shown by the enclosing *parallel gateways*. Finally, end event **e** concludes the process.

UML is a standard for software modeling and design that embodies a set of formalisms for capturing important aspects of software systems [63]. UML class diagrams are used to design conceptual data models and are primarily composed of classes and associations among them.

Classes are at the heart of object-oriented systems and describe the sets of objects that exist in a system and share common properties. Each class is characterized by a unique name and a set of properties, namely *attributes* and *operations*. *Objects* that belong to a class constitute the so-called instantiation of the class. Unique object identifiers (OIDs) are assigned to objects for identification and for information sharing purposes.

Class diagrams allow one to define many kinds of relationships between classes, called *associations*. Binary association are depicted as a solid lines that connect two participating classes. N-ary associations are represented by a diamond connected to all the participating classes by a solid line.

Properties related to associations are represented by *association classes*, which are declarations of associations that have a proper set of features. An association class is both an association and a class, and preserves the static and dynamic semantics of both [63].

Finally, *multiplicity* specifies the maximum participation cardinality of each class in the association. Each class is associated with a multiplicity ($m_{min}..m_{max}$) where m_{min} (m_{max}) $\in \{0, 1, *\}$ that defines the minimum and maximum number of class instances that can participate in the association.

The UML class diagram of the exam registration repository is shown in Fig. 7.1(b). Classes **Student**, **Exam**, and **Course** represent the main concepts of interest, related by associations **grade**, with multiplicity (0..*, 0..*), **registration**, **examination**, and **attendance**. Associations **grade** and **registration** have a related association class. Reflexive association **representative** links students with their student representative. Finally, classes **Bachelor** and **Master** specialize students.

Despite capturing informational aspects through data objects and data stores, BPMN process models provide little or no detail about the operation performed by process activities on a database. This lack of knowledge complicates the modeling of data in BPMN processes from different standpoints.

(i) In BPMN data stores represent persistent data sources [11] and, most likely, conceptualize where (i.e., by which activities) a database is accessed in the process. However, BPMN defines them at a very high level, and there is currently no standard-compliant way of specifying the schema of the database represented by the data store. For example, the conceptual schema of the database represented by data store DB of Fig. 7.1(a) cannot be specified.

(ii) BPMN data objects can be used to represent volatile data at different granularities, spanning from single data entries to structured documents [182]. As a result, the correspondence between data objects or conceptual information entities related to a process and persistent data is not specified.

For instance, it is not clear whether data object `unregistered students` is related to data class `Student` and, if so, how. More in general, let us suppose that a professor wants to read a student’s grade transcript, containing the whole academic record. In the data schema of Fig. 7.1(b), there is no class “transcript” that captures such information directly, since the concept of grade transcript is realized by data classes `Student` and `Exam`, related through association `grade`, and by association class `Grade`. Therefore, the data object representing the transcript must correspond to a more complex conceptual object, which is identified by a view on the accessed database.

(iii) Last but not least, too little detail about data operations is provided as these are often encoded within activities or their labels [177, 182]. Despite directed data associations allow one to visualize when data are read or written by an activity, it is not possible to distinguish the granularity of the conceptual object(s) or of the sets of objects needed by the process.

As process models and database schemata are conceived separately, to foster data-aware process modeling it is necessary to support designers in understanding and capturing the connection between processes and databases [74, 83].

7.1.2 The Activity View to connect Processes and Data

In this section, we propose a novel solution aimed to capture the connection between BPMN process models and UML class diagrams at a conceptual level.

To this end, we devise the *Activity View*, a novel approach linking the conceptual representations of process models and data schemata by detailing which operations are performed by a process activity on a database and how.

We chose activities as a starting point, as data modeling in BPMN is often related to activities or whole processes. The final goal of the Activity View is to show which is the portion of a database schema (i.e., the view) that is accessed by a given process activity and to detail interesting aspects of the performed data operations.

Definition 7.1 (Activity View). Let us consider dealing with a process model $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$ (cf. Def. 2.1).

Given an activity $ac \in A \subseteq N$, its Activity View $av_{ac} = \{t_1, \dots, t_n\}$ is a set of tuples, where each tuple t_i denotes a particular data access operation performed

by activity ac on classes of a given database schema. The latter is composed of a set of classes Cl , a set of associations As , and a set of association classes AsC . Each tuple t_i is defined as follows:

$$t_i = \langle C_{set_i}, A_{set_i}, AccessType_i, AccessTime_i, NumInstances_i \rangle$$

where:

- $C_{set_i} = \{c_1, \dots, c_j\} \subseteq (Cl \cup AsC)$, is the set of connected classes accessed by process activity ac . By “connected” we mean that each class $c_j \in C_{set_i}$ is reachable from at least another class $c_h \in C_{set_i}$ by navigating an association $a_f \in As$ that directly links c_h and c_j (i.e., c_h and c_j are at the ends of association a_f). Moreover, if a_f is associated to association class $c_f \in AsC$, then c_l must also belong to C_{set_i} . However, $c_f \in AsC$ may also be accessed individually, as other classes. If a class $c_j \in C_{set_i}$ specializes a more general class c_l , then it is sufficient that c_l is one end of association a_f for c_j to be considered connected to other classes of C_{set_i} . Instead, the opposite does not hold.

Each class $c_j(attr_1, \dots, attr_n) \in C_{set_i}$ is characterized by a unique name c_j and a set of attributes $\{attr_1, \dots, attr_n\}$. If all the attributes of c_j are involved in the data operation, we write $c_j(*)$. Instead, if only a subset of attributes of c_j is accessed, we explicitly specify it by $c_j(attr_g, \dots, attr_m)$ with $1 \leq g < m \leq n$.

- $A_{set_i} = \{a_1, \dots, a_r\} \subseteq C_{set_i} \times C_{set_i} \subseteq As$ is a set of binary associations that directly link any two classes of C_{set_i} (i.e., the ends c_h and c_j of association a_i with $1 \leq i \leq r$ must belong to C_{set_i}). $A_{set_i} = \{*\}$ is the set of all associations that directly link any two classes of C_{set_i} .
- $AccessType_i \in \{R, I, D, U\}$ defines the type of access to the related information. R denotes a *read* of elements of C_{set} , whereas I , D , and U denote different kinds of write operations, namely I indicates an *insertion*, D stands for a *deletion*, while U denotes an *update*.
- $AccessTime_i \in \{start, end, during\}$ denotes when a data operation is performed with respect to activity execution. This qualitative information refines the description of data operations by specifying the moment they are performed. Access time defines a partial order among Activity View tuples.
- $NumInstances_i = (min, max)$, where $min \in \{0, 1, *\}$ and $max \in \{1, *\}$, denotes the number of objects involved in the considered operation. Indeed, activities must be able to access collections of objects and cardinalities should be properly managed [183]. Values 0, 1, * have the same meaning as in UML multiplicity specification, where * means that multiple objects are involved in the considered operation.

An Activity View is consistent if the following constraints hold:

- (i) $\forall i C_{set_i} \neq \emptyset \wedge AccessType_i \neq \emptyset \wedge AccessTime_i \neq \emptyset \wedge NumInstances_i \neq \emptyset$;
- (ii) If $|C_{set_i}| = 1$ and no (reflexive) association is accessed by the process, then $A_{set_i} = \emptyset$;

- (iii) If $|C_{set_i}| > 1$, then $A_{set_i} \neq \emptyset$ and all data classes that are ends of $a_f \in A_{set_i}$ and the association class $c_f \in AsC$ related to a_f must belong to C_{set_i} ;
- (iv) An association class $c_f \in AsC$ may be accessed individually, as other classes.

Sitting in-between two well-established standards, the Activity View blends the concepts of activity, borrowed from BPMN, with those of class, attribute, and association taken from UML. Moreover, being defined independently from data stores and data objects the Activity View provides a clear representation of the area of a data schema used by an activity as well as of the data operations performed on it, thus addressing the open issues (i) – (iii) discussed in Sect. 7.1.1.

As an example, consider the previously described process of Fig. 7.1(a). To check students attendance, the professor must retrieve data regarding student enrolment from the exam registration repository, depicted as data store DB.

Given the data schema of the exam registration repository shown in Fig. 7.1(b), the described data access operations can be formalized as follows:

$$\begin{aligned}
 av_{CheckAttendance} &= \{\{\{Student(matriculation), Exam(name, date), \\
 &\quad Registration(*)\}, \{registration\}, R, during, (1, *)\}\}. \\
 av_{RegisterGrade} &= \{\{\{Student(matriculation), Course(name)\}, \{attendance\}, \\
 &\quad R, start, (1, 1)\}, \{\{Grade(*)\}, \emptyset, I, during, (1, 1)\}\}. \\
 av_{RescheduleExam} &= \{\{\{Exam(*)\}, \emptyset, I, during, (0, 1)\}\}.
 \end{aligned}$$

Activities **Conduct Exam** and **Inform Student** do not have a related Activity View as they do not require access to persistent data.

For improved readability, the tuples of one Activity View can be represented in a tabular form, as exemplified in Fig. 7.2 for activity **Register grade**.

<i>avRegisterGrade</i>					
<i>Tuple</i>	<i>C_{set}</i>	<i>A_{set}</i>	<i>AccessType</i>	<i>AccessTime</i>	<i>NumInstances</i>
<i>t₁</i>	{ <i>Student(matriculation), Course(name)</i> }	{ <i>attendance</i> }	<i>R</i>	<i>start</i>	(1, 1)
<i>t₂</i>	{ <i>Grade(*)</i> }	\emptyset	<i>I</i>	<i>during</i>	(1, 1)

Fig. 7.2: Tabular representation of the Activity View for activity **Register Grade**.

Graphically, the link captured by the three described Activity Views can be visualized over a process diagram and a database schema as shown in Fig. 7.3. Dashed arrows connect activities to the portion of the data schema specified in the Activity View. The same colors are used for the activity border, the dashed lines that frame the accessed data classes, and the full lines that highlight associations. Connecting arrows are labeled with the information related to access type, access time, and number of objects involved in the operation. Whenever multiple tuples of one Activity View represent different data operations on the same area of the data schema, the dashed arrow connecting the activity with that area of the data schema may be associated to multiple labels.

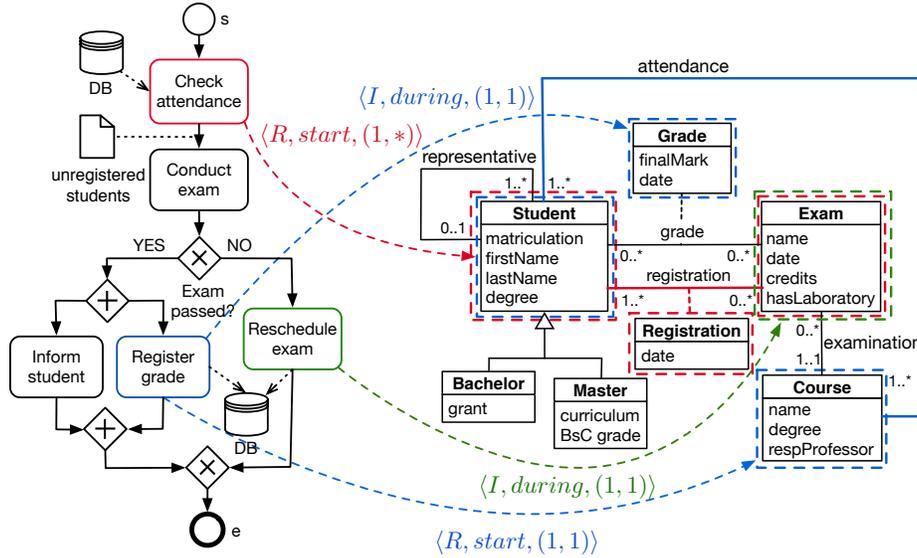


Fig. 7.3: Graphical representation of Activity Views for the introduced example.

To come up with the definition of Activity View provided above, we considered the following aspects related to linked processes and data.

Data classes and attributes. Data classes define the conceptual objects of interest that are needed by a process activity. An Activity View allows designers to specify that only certain attributes of a class are read or written. This situation is quite common whenever the data schema represents a database that has not been specifically designed for process support. Moreover, the creation/update of a certain object may be realized by multiple activities, each one acting on a specific attribute [183]. Finally, when considering process roles and data access privileges, it is plausible that certain attributes may have restricted access and, thus, a data class may not necessarily be accessed as a whole.

Data associations and association classes. Adding data associations to the specification of an Activity View changes the level of detail provided by the Activity View itself, especially for those data schemata having reflexive or multiple associations between any two classes.

Specifically, if associations are not specified, the Activity View has a higher level of abstraction but it is not clear how any two classes of C_{set} are connected. Instead, by specifying class associations, we provide a more precise description of how the classes of C_{set} are related.

As an example, let us consider the setting shown in Fig. 7.3 and let us assume that the department secretary needs to have access to the information related to exam registration, that is, she will need to read objects of classes

Student, Registration, and Exam, respectively. However, for privacy reasons, secretaries are not allowed to see the grades of students, stored in objects of association class *Grade*. Whereas the described scenario cannot be managed if associations are not specified, as shown in Fig. 7.4.(a), considering data associations one can distinguish between *grade* and *registration*, as shown in Fig. 7.4.(b). Thus, Def. 7.1 includes associations.

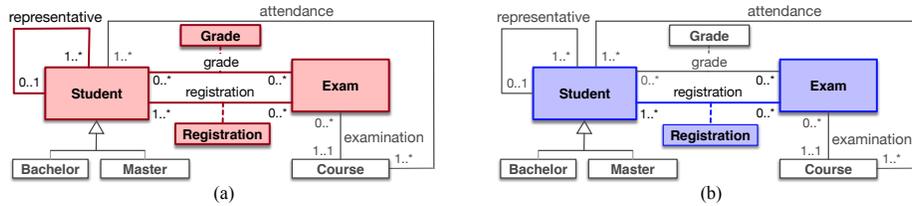


Fig. 7.4: Example of Activity View when (a) excluding or (b) including associations between classes.

7.1.3 Novel Conceptual Insights

Besides supporting the conceptual modeling of process and related data, the Activity View provides other interesting perspectives that enable basic reasoning on the interplay between process and data models, useful for both analysis and communication purposes. In this section, we discuss how Activity Views can be exploited to capture interesting aspects of the connection between processes and data that can be useful during process modeling and analysis.

Identifying the portion of a data schema accessed by a process activity. As described in Sect. 7.1.2, the Activity View allows one to identify which are the classes and associations of a data schema that are accessed by a certain activity, thus providing a better specification than BPMN data stores. To identify the portion of the data schema accessed by an activity ac_k , all the tuples $t_{1,k}, \dots, t_{n,k}$ of av_{ac_k} must be properly combined: the comprehensive set of classes and association classes of a data schema accessed by ac_k is $\bigcup_{j=1}^n C_{set_{j,k}}$, where j denotes the tuple and k the activity, while the set of all associations accessed by ac_k is $\bigcup_{j=1}^n A_{set_{j,k}}$. In Fig. 7.3 and Fig. 7.5, the area of interest of the data schema is graphically rendered by framing classes with dashed lines colored as the border of the activity that access them.

Detecting which activities operate on a certain data class. Under another viewpoint, Activity Views allow one to understand which process activities have access to objects of a certain data class. This is useful for several reasons, starting from easing the communication with domain experts during process modeling. Stakeholders are often interested in seeing where certain data

are used in the process to understand which is the information that drives certain activities and used to make decisions. This holds also for data compliance. Indeed, in some circumstances, the quality of activity execution may drastically improve if proper information is available. Under an engineering perspective, understanding how data are used during process execution provides hints for data management support and re-engineering.

For example, class `Exam` of Fig. 7.5 is accessed by tasks `Check attendance` and `Reschedule exam`, as highlighted by the filled background. By taking a look at the structure of the process, one can see that if the student succeeds, class `Exam` is only accessed at the beginning of the process.

The set of process activities ac_g, \dots, ac_l that access a certain class c_i is $\{ac_k \mid \exists t_{j,k} (c_i \in C_{set_{j,k}})\}$ and it is obtained by going through all the Activity Views of a process and checking whether c_i belongs to at least one class set $C_{set_{j,k}}$ of a tuple $t_{j,k} \in av_{ac_k}$.

Understanding which classes are either read or written by a process. The type of access to data allows designers to easily visualize whether data classes have associated read/write operations and how these are distributed in the process. Besides, one can also retrieve which classes of the data schema are associated only to read or write accesses. This is particularly useful when speaking about data integrity, as several activities of one or more processes may operate on the same data class concurrently and, thus, transactional properties must be discussed [42, 83]. Last but not least, certain sequences of read and write operations performed on the same data classes may lead to inconsistencies in [56]. For example, in order to understand whether objects of a class c_k are only written by activities of a process, we shall go through all the Activity Views and ensure that there exists no tuple having $c_k \in C_{set}$ and access type of kind R : $\{c_k \mid \nexists j, i ((C_{set_{j,i}} \ni c_k) \wedge AccessType_{j,i} = "R")\}$. In Fig. 7.5, for each data class related to the process, \textcircled{R} and \textcircled{W} denote if the class is read or written by process activities.

By combining the described insights provided by the Activity View, designers can understand and visualize, with the help of stakeholders, which is the key information needed to support process execution. This can be represented by one or more data classes, which we refer to as *core classes* for a given process.

Informally, given a data schema, a core class is a class of the data schema that represents valuable process-related data and

- (i) it appears in a considerable number of Activity Views related to the process (i.e., it is shared by multiple process activities);
- (ii) its objects are frequently accessed by the process, that is, it appears in a considerable number of Activity View tuples;
- (iii) its objects are used by the most important activities of the process (i.e., activities that are crucial for the chosen application domain or are executed in (almost) all the instances of a process, if any);
- (iv) it is mostly subjected to mandatory access [183], that is, Activity View *min* of *NumInstances* is never equal to 0.

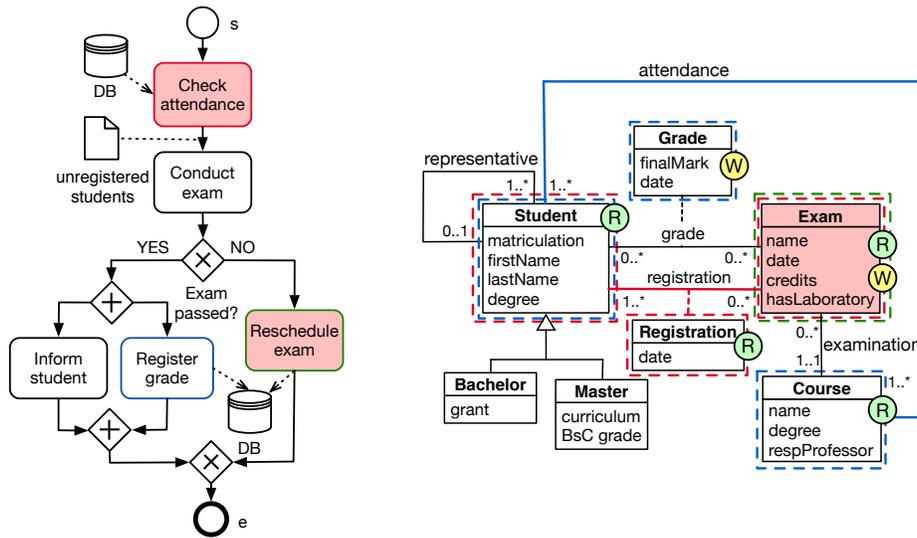


Fig. 7.5: Visualization of the conceptual insights provided by the Activity View.

With respect to the process of Fig. 7.5, classes Exam and Student are core classes, as they are the most accessed in the process. As for their use, they are accessed by exactly the same activities, but class Student is only read by process activities. Of course, to determine whether a read-only access is less important than a write access, domain experts should be consulted, as the idea is to exploit Activity Views in any way, to retrieve information useful for conceptual design. Indeed, whereas in such a simple example the identification of important data is quite straightforward, the concept of core classes becomes useful in complex and highly branched processes, where identifying the key information to support process execution is not straightforward.

This latter issue is similar to an open issue in the field of data-centric process modeling, since the same questions need to be answered to identify the data artifacts on which the processes are based [39, 42].

7.2 A Relational Approach based on the Activity View

In this section, assuming that the reader got an idea of the linkage realized by the Activity View, we start from a scenario coming from surgical medicine to show a possible way to analyze of linked process and data models.

The process model of Fig. 11.1 shows a BPMN process model describing the main steps related to laparoscopic appendectomy [184], i.e., the surgical removal of an infected appendicitis.

Example 7.2 (Laparoscopic Appendectomy). Laparoscopic appendectomy is a safe and effective method for treating appendicitis, the inflammation and infection of the appendix, which represents the most common general surgical disease affecting the abdomen. For simplicity, we consider dealing with patients that are already hospitalized for suspected appendicitis and require evaluation and treatment.

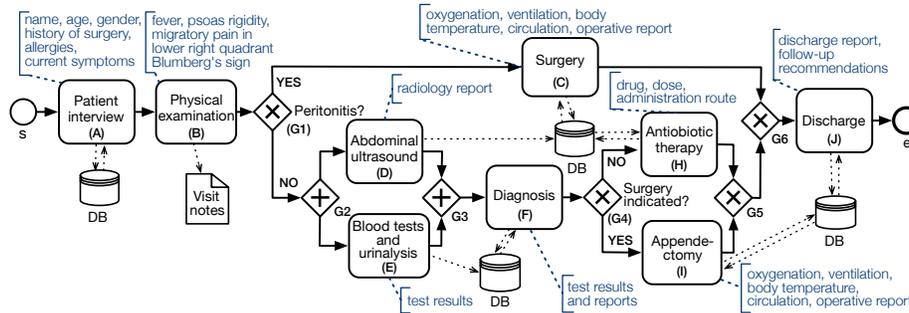


Fig. 7.6: BPMN process diagram modeling the main steps related to laparoscopic appendectomy. (Blue) text annotations describe the data needed by each activity.

The diagnosis of appendicitis is mainly clinical, and begins with a **Patient interview (A)**, during which information about the patient such as *generalities*, *medical history*, and *current symptoms* is gathered and recorded in the clinical database. The latter is represented by data store DB, which is referenced multiple times in the process. Common symptoms of appendicitis include *abdominal pain*, *loss of appetite*, *nausea*, and *constipation*. Then the patient undergoes a **Physical examination (B)**, during which signs indicative of appendicitis, such as the presence of *fever*, *psoas rigidity* or *migratory pain* to the right lower quadrant of the abdomen are investigated and summarized in *Visit notes*, represented by thy homonym data object.

If during physical examination the physician suspects a peritonitis, a more severe and fatal infection, the patient must immediately undergo **Surgery (C)**. Otherwise, further tests are conducted in order to rule out other conditions. An **Abdominal ultrasound (D)** exam is used to obtain images of the appendix, while **Blood tests and urinalysis (E)** are needed to check for infection. If the *test results* and *radiology report* stored in the DB suggest a **Diagnosis (F)** of uncomplicated appendicitis, an **Appendectomy (I)** is performed under anesthesia. Any surgical intervention under anesthesia requires that data about the administered *treatment* are recorded in the DB, together with constant monitoring information related to the patient's *oxygenation*, *ventilation*, *circulation*, and *body temperature*. As soon as surgery is completed, an *operative report* is compiled to document the procedure and to provide information useful for post-operative follow-up. In some cases, **Antibiotic therapy (H)** may be prescribed to treat uncomplicated

appendicitis non-operatively. Usually, Discharge (J) from the hospital is planned a few days after surgery/treatment and self-care continues at home.

In this section, we consider dealing with process models complying with Def. 2.1, but we omit the resource perspective and boundary events for practicality. That is, given a process model $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$, we consider $R = \emptyset$ and $E_B = \emptyset$, where $E_B \subseteq E_{int} \subseteq E \subseteq N$. Being $E_B = \emptyset$, according to structural criterion (v) on page 29, m has a unique end event.

Without loss of generality, we assume to deal with process models that follow the principles of structural soundness and well-structuredness introduced in Sect. 2.1.2, and are compliant with the structural criteria defined on page 29. Moreover, we consider processes having exactly two flows outgoing/incoming of split/merge gateways to increase model readability. Indeed, a gateway having x outgoing/incoming flows, may be expressed as a cascaded sequence of $x - 1$ binary gateways. In other words, we restrict criteria (ii) and (iii) on page 29 such that $|g \bullet| = 2$ for split gateways and $|\bullet g| = 2$ for merge gateways.

In Fig. 7.6, we make use of text annotations (sometimes combined with data stores) to give an idea of the most important data needed by each process activity to be executed. Persistent data needed to support activity execution are usually stored in databases and managed by suitable database systems, that are likely to serve several applications and processes. Thus, we may assume that data about patient admissions, diagnoses, interventions, and treatments are stored in a database, such as the one represented Fig. 7.7 by a UML Class Diagram [63].

Despite giving an idea of the needed information, text annotations do not necessarily refer to a database. Similarly, data store DB identifies where persistent data are needed in the process, but does not suffice to detail the data operations performed by process activities on such data.

Therefore, we resort again to the Activity View, introduced in Sect. 7.1.

In Fig. 7.7, we provide a graphical account of the connection realized by the Activity View, and show the process of Fig. 7.6 (at the top) together with the data operations performed by activities E and J on the clinical database (whose schema is depicted at the bottom).

The complete list of Activity Views for Example 7.2 is provided below.

$$\begin{aligned}
av_A &= \{\langle\{Patient(SSN), Admission(*)\}, \{hospitalization\}, R, start, (1, *)\}, \\
&\quad \langle\{Interview(*)\}, \emptyset, I, during, (1, 1)\rangle, \langle\{Surgery(*)\}, \emptyset, R, during, (1, *)\rangle\}; \\
av_C &= \{\langle\{Admission(*)\}, Interview(*)\}, \{anamnesis\}, R, start, (1, *)\}, \\
&\quad \langle\{Surgery(*)\}, \emptyset, I, start, (1, 1)\rangle, \langle\{Treatment(*)\}, \emptyset, I, start, (1, *)\}, \\
&\quad \langle\{Monitoring(*)\}, \emptyset, I, during, (1, *)\rangle, \langle\{Surgery(opRep)\}, \emptyset, U, end, (1, 1)\rangle\}; \\
av_D &= \{\langle\{DiagExam(*)\}, \emptyset, I, start, (1, *)\rangle, \langle\{Report(*)\}, \emptyset, I, end, (1, *)\rangle\}; \\
&\quad \langle\{DiagExam(status), Report(*)\}, \{result\}, U, end, (1, *)\rangle\}; \\
av_E &= \{\langle\{DiagExam(*)\}, \emptyset, I, start, (1, *)\rangle, \langle\{Report(*)\}, \emptyset, I, end, (1, *)\rangle\}; \\
&\quad \langle\{DiagExam(status), Report(*)\}, \{result\}, U, end, (1, *)\rangle\};
\end{aligned}$$

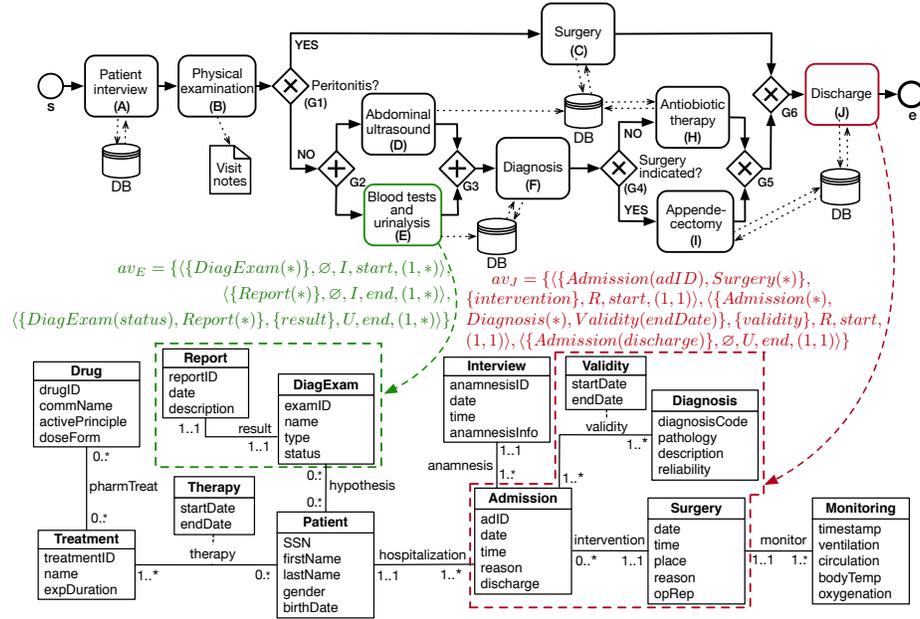


Fig. 7.7: Graphical representation of Activity Views connecting process activities Blood tests and urinalysis and Discharge to the accessed areas of the UML Class Diagram.

$$\begin{aligned}
 av_F &= \{\{\{DiagExam(examID), Report(*)\}, \{result\} R, start, (1, *)\}, \\
 &\quad \{\{Diagnosis(*)\}, \emptyset, U, during, (0, 1)\}\}; \\
 av_H &= \{\{\{Patient(SSN), Treatment(*)\}, \{Drug(drugID), Therapy(*)\}, \{therapy, \\
 &\quad pharmTreat\}, R, start, (1, *)\}, \{\{Treatment(*)\}, \emptyset, I, start, (1, *)\}\}; \\
 av_I &= \{\{\{Admission(*)\}, \{Interview(*)\}, \{anamnesis\}, R, start, (1, *)\}, \\
 &\quad \{\{Surgery(*)\}, \emptyset, I, start, (1, 1)\}, \{\{Treatment(*)\}, \emptyset, I, start, (1, *)\}, \\
 &\quad \{\{Monitoring(*)\}, \emptyset, I, during, (1, *)\}, \{\{Surgery(opRep)\}, \emptyset, U, end, (1, 1)\}\}; \\
 av_J &= \{\{\{Admission(adID), Surgery(*)\}, \{intervention\}, R, start, (1, 1)\}, \\
 &\quad \{\{Admission(*)\}, \{Diagnosis(*)\}, \{Validity(endDate)\}, \{validity\}, R, start, (1, 1)\}, \\
 &\quad \{\{Admission(discharge)\}, \emptyset, U, end, (1, 1)\}\};
 \end{aligned}$$

Activity B does not require access to the clinical database and, thus, it does not have an associated Activity View.

By blending basic data representation and manipulation aspects, the Activity View offers a versatile conceptual bridge, which can be exploited for retrieving interesting information about the link between process and data models, while leaving both models untouched.

In this section, we argue that the informational perspective of process models brings interesting insights about the operational semantics of activities and show an approach addressed to process and database designers to explore the connec-

tion between process and database models. In particular, we focus on addressing the following information requirements *R1–R4*, exemplified with respect to the process of Fig. 7.6, that we elicited by considering the both the structure of process models and the kinds of data access operations typically performed by the activities on a database.

R1 Identical data operations. It captures activities in a process that perform the same operations on exactly the same portion of the database.

Understanding which activities are characterized by the same data operations is particularly interesting for enhancing re-use. On the one hand, re-use can be considered in terms of reusable element definition during process (re-)design (e.g., call activities may be specified). On the other hand, re-use can be seen in terms of defining data access permissions, as resources may be associated to roles considering their permissions to read or write certain data [183]. Let us consider tasks C and I of Fig. 7.6. Despite being two different interventions, their data perspective is the same. Having this kind of information about process activities is useful for several reasons, such as (i) providing compliance with clinical data collection guidelines or (ii) managing data access permissions of clinical/technical staff.

R2 Use of data across process paths. It evaluates whether activities located on alternative paths require the same data to be executed.

Alternative process paths often capture different process settings in terms of performed actions. However, activities located on alternative paths may require the same data. During re-engineering, this information may help to re-arrange activities and improve data collection (e.g., a data collection activity may be moved before the exclusive gateway). Again, C and I of Fig. 7.6 have the same activity view despite being alternative interventions. Under a database viewpoint, knowing which data are needed by activities along each path is useful to identify information that is central to the whole process. For example, class *Treatment* of Fig. 7.7 is used by tasks C, H, and I, that together “cover” all alternative process paths.

R3 Use of data along process paths. It explores how data is added/modified by activities located along a certain process path.

Indeed, whereas some data created by an activity are used later in the process, some data operations may be superfluous [185]. Thus, knowing data reading and writing patterns may help designers getting rid of redundant data operations or adding necessary data access. For example, designers may be interested in knowing which is the last activity of the process that *reads* data of class *Admission* (that is, task J).

R4 Concurrent data access. It checks whether activities located along parallel paths require access to the same data.

Under a database standpoint, this gives an overview of potentially concurrent operations requiring transactional control. For example, tasks E and D of Fig. 7.6 may have concurrent access to the same portion of the database, as they lie on parallel paths and use the same data.

The list of introduced information requirements is not complete, in the sense that it is possible to think of other interesting aspects related to process and persistent data integration (e.g., finding data that are included in a database, but never used by a process, or data authorizations [183]).

However information requirements *R1–R4* exemplify significant properties of integrated process and database models, thus providing an overview of the exploration that can be conducted on the overall connected “system” and laying the bases for process re-design and improvement [186].

In the remainder of this chapter, we describe an approach based on relational data modeling techniques to address the introduced requirements.

7.2.1 A Relational Approach for Modeling the Data Perspective

In order to encode the information related to the process model, the database model, and their connection, we rely on the well-known relational data model [181, 187] and design a database that collects and integrates the needed information.

To briefly introduce relational data modeling, let \mathcal{A} be a countably infinite set of attributes having a total order relation \mathcal{A}_{\leq} defined on it. We associate to each attribute $att \in \mathcal{A}$ a set of atomic values $dom(att)$ representing the domain of att . A relational database schema is a non-empty finite set $D = R_1, \dots, R_n$ of relation schemata, where each relation schema R_i is composed of a relation name *relname* and a subset of elements of \mathcal{A} [181, 187].

In order to handle non-atomic or hierarchically structured data, a few extensions of the relational model have been proposed [188]. Such data have complex values that are derived from atomic ones using dedicated constructors.

The relational schema of the proposed database is shown in Fig. 7.8. Attributes that denote primary keys are underlined. As for referential integrity constraints, referencing and referenced attributes are given the same name.

Relations 1–10 represent the elements of a BPMN process model that complies with Def. 2.1 and observes the previously mentioned structural restrictions. Relations **Activity**, **Gateway**, **Event** specialize **FlowNode**, while **DataObject** and **DataStore** specialize **DataNode**. Therefore the same primary keys are used.

For practicality and future work, we deliberately allow the possibility to store information about boundary events in the data schema of Fig. 7.8. However, we underline that in this section we consider dealing with process models that do not include boundary events.

Relation **Pred** and attribute **label** of relation **Activity** store information related to the structure of the process. Relations **ActivityView**, **ComposedOf**, and **DataOp** denote the data operations (i.e., the Activity Views) performed by the process on a database (represented by relation **DomainDB**), whose classes and associations are represented by relations 16–18.

Since the Activity View includes attributes such as C_{set} and A_{set} , which are sets of classes (respectively associations), we consider the relational data model extended with *sets* and other complex values [188]. That is, given an instance of relation **Class**, a set of classes is denoted by $\{\mathbf{Class}\}$.

1. Process(processID, name, documentation)
2. DataNode(dnID, processID)
3. DataStore(dnID, dsName, capacity)
4. DataObject(dnID, name, isCollection)
5. DataFlow(fnID, dnID, direction)
6. FlowNode(fnID, processID)
7. SequenceFlow(from, to)
8. Activity(fnID, activityName, isReusable, isAtomic, type, hasBoundary, boundaryEventRefs*, avID*, label)
9. Gateway(fnID, gatewayName, routingType, splitType, hasQuestion, question*)
10. Event(fnID, eventName, type, position, triggerType, isBoundary, boundaryType*)
11. Pred(fnID, preds:{Activity})
12. ActivityView(avID)
13. ComposedOf(avID, opID)
14. DataOp(opID, cSet:{ClassDB}, aSet:{AssociationDB}*, accessType, accessTime, minInstances, maxInstances)
15. DomainDB(dbID, dbName)
16. AssociationDB(assocName, dbID, leftCardinality, rightCardinality)
17. ClassDB(className, dbID, assocClass, assocRef, attributes:{AttributeDB})
18. AttributeDB(attributeName, className, dbID, dataType)

Fig. 7.8: Relational schema of the database storing information about a process model, a database model, and data operations performed by process activities. Primary keys are underlined. Some attributes, such as avID can be NULL as denoted by the * symbol.

Business processes have a graph-like structure and, thus, storing and manipulating information related to flow nodes and edges in a relational database requires special care. Indeed, due to alternative flows, not all process executions perform exactly the same set of activities.

To encode all the paths possibly taken by a process and keep track of how they are chosen during execution we associate labels to flow nodes. Since we are considering process models having exactly two flows outgoing of a split gateway, our labeling approach draws inspiration from the one proposed in [134]. However, being our focus on the data perspective, we are interested in labeling only process activities. The definition of label used in the remainder of this chapter is given below.

Definition 7.2 (Label). Given a set \mathcal{P} of propositional letters, a label $\ell = l_1 \wedge \dots \wedge l_n$ is a (possibly empty) conjunction of literals, where each l_i can be assigned distinct values $\{p_i, \neg p_i\}$ of $p_i \in \mathcal{P}$. The empty label is denoted \square and is an identity for label conjunction, i.e., $\square \wedge \square = \square$, $\square \wedge \ell = \ell$.

The notion of label provided in Def. 7.2 must not be confused with the one provided in Def. 2.1, i.e., the string σ assigned to flow nodes by function \mathcal{L} . Indeed, σ is a string defined in natural language that provides a name to flow nodes, as specified the BPMN standard [11].

To capture information about all the possible paths of a process, their execution, and the ordering of activities along each path, we designed Algorithm 1 (`processPath`), a recursive procedure that populates field `label` of relation `Activity` and relation `Pred` of the database in Fig. 7.8.

Algorithm 1 takes in input a process model $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \rho, \mathcal{L})$, a node $n \in N$, a label ℓ managed as a LIFO queue of literals, a set of activities P , and an ordered list of activities T .

Given a process model m , the procedure traverses and enumerates all the alternative paths going from its start event s to its end event e . Each recursive call focuses on flow node n . The label ℓ (i.e., the conjunction of literals) encodes the path taken to reach n from s . P is the set of activities preceding n along a particular path, which does not consider how many times an activity is traversed. However, since a process may have loops, ordered list T keeps track of how many times the activities preceding n along a particular path are traversed.

According to the kind of flow node n encountered, Algorithm 1 takes different actions before moving to subsequent nodes, which are at most two, in case n is a split gateway. At each recursive call, while the traversed path becomes longer, parameters ℓ , P , and T grow collecting information about the path.

The initial call of Algorithm 1 is `processPath(m, s, { \square }, {}, {})`. Each conditional statement evaluates n and proceeds as follows.

Activities. Algorithm 1 checks how many times each activity has been accessed in the current path (line 2), to prevent remaining indefinitely in a process loop.

The procedure iterates either zero or one time, not more. Thus, recursive calls stop at the beginning of the second loop. Then, n is labeled with ℓ (line 3). Finally, n and the list of the activities preceding it on that process path are added to relation `Pred` (line 4). Prior to moving on to the following node, n is appended to the list of visited activities (line 5).

Exclusive gateways. Exclusive gateways are crucial for determining labels (lines 6–12), since a new literal is added to the current label whenever a split exclusive gateway is encountered. In detail, each one of the two outgoing branches is associated with a literal p , respectively $\neg p$, that is appended to ℓ . Whenever a merge gateway is encountered (lines 19–20), the latest added literal is removed from ℓ . When the same exclusive split gateway is traversed multiple times (i.e., by different recursive calls), the same literals are used. In this way, paths are consistently labeled.

Algorithm 1: processPath

Input: process model $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \rho, \mathcal{L})$, $n \in N$, label ℓ , P set of activities, T list of activities traversed before n .

Result: populates field `label` of relation `Activity` and relation `Pred`.

```

1 processPath( $m, n, \ell, P, T$ )
2 if ( $n \in A \wedge \text{occurrences}(T, n) < 2$ ) then
3   Activity(fnID, label)  $\leftarrow (n, \ell)$ ; /*updates Activity setting the value of label
   for  $n$  */
4   Pred(fnID, preds)  $\leftarrow (n, P)$ ; /*inserts values in relation Pred */
5   processPath( $m, n \bullet, \ell, P \cup n, \text{append}(T, n)$ );
6 else if ( $n \in G \wedge (\gamma_{ty}(n) = xor) \wedge \gamma_r(n) = split$ ) then
7   if (gatewayProposition( $n$ ) = null) then
8     gatewayProposition( $n$ )  $\leftarrow$  NewProposition /*creates new literal*/
9      $s' \leftarrow$  first( $n \bullet$ ); /*first node on first branch outgoing of  $n$ */
10     $s'' \leftarrow$  second( $n \bullet$ ); /*first node on second branch outgoing of  $n$ */
11    processPath( $m, s', \text{push}(\ell, \text{gatewayProposition}(n)), P, T$ );
12    processPath( $m, s'', \text{push}(\ell, \neg \text{gatewayProposition}(n)), P, T$ );
13 else if ( $n \in G \wedge \gamma_{ty}(n) = and \wedge \gamma_r(n) = split$ ) then
14    $s' \leftarrow$  first( $n \bullet$ ); /*first node on first branch outgoing of  $n$ */
15    $s'' \leftarrow$  second( $n \bullet$ ); /*first node on second branch outgoing of  $n$ */
16   processPath( $m, s', \ell, P, T$ );
17   processPath( $m, s'', \ell, P, T$ );
18 else if ( $n \in G \wedge \gamma_r(n) = merge$ ) then
19   if ( $\gamma_{ty}(n) = xor$ ) then
20     pull( $\ell$ ) /* pull( $\{\square\}$ ) =  $\{\square\}$  */
21     processPath( $m, n \bullet, \ell, P, T$ );
22 else
23   if ( $n \in E_{start} \vee n \in E_{int}$ ) then
24     processPath( $m, n \bullet, \ell, P, T$ );
25 return;
```

Parallel gateways. Parallel split gateways (lines 13–17) do not affect path labeling, since both paths outgoing of such gateways are always executed. Similarly, when a parallel merge gateway is met (lines 18 and 21) a new recursive call is made, without influencing the current labeling.

Events. Events do not affect path labeling (lines 22–24). When the end event is reached, all recursive calls end.

A sample output of Algorithm 1 for the process of Fig. 7.6 is shown in Fig. 7.9.

Labels are shown on the top of process activities for understandability purposes, but the process model is not altered by the algorithm. Activities that are labeled with \square belong to paths that are traversed by every process execution. Attribute `preds` of relation `Pred` is the set of all predecessors of one activity

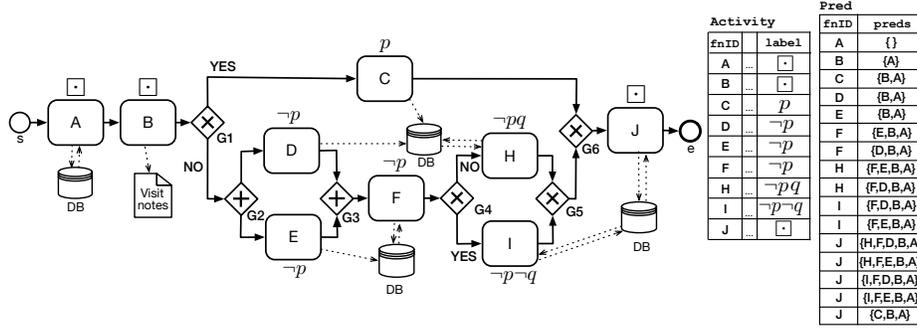


Fig. 7.9: BPMN process of Fig. 7.6 labeled by applying algorithm processPath. For the purpose of this example activity names and identifiers are assumed to coincide.

along a certain path. To obtain all the predecessors of one activity it is sufficient to merge all the sets **preds** related to that activity. Moreover, when this information is combined with that of **label**, the way paths and activities are executed becomes clearer.

In detail, let us consider two activities a_1 and a_2 , associated to labels ℓ_1 and ℓ_2 , respectively.

- (i) If $a_1 \in \mathbf{preds}$ and a_2 or $a_2 \in \mathbf{preds}$ of a_1 , then a_1 and a_2 are in sequential order;
- (ii) if $a_1 \notin \mathbf{preds}$ of a_2 or $a_2 \notin \mathbf{preds}$ of a_1 :
 - if there exists at least one propositional letter p such that $p \in \ell_1$ and $\neg p \in \ell_2$, then a_1 and a_2 belong to alternative paths;
 - if such propositional letter does not exist (i.e., for each propositional letter p it is never true that $p \in \ell_1$ and $\neg p \in \ell_2$), then a_1 and a_2 belong to parallel paths.

As for the computational complexity of Algorithm 1, it is clear that any recursive call requires a constant number of steps to be executed. Similarly, $\mathbf{occurrences}(T, n)$ can be implemented in constant time.

Thus, the complexity of the algorithm is given by the number of recursive calls needed to enumerate all the possible paths of m . Since the paths going from the start event to the end event are exponential with respect to the number of split gateways in m , the procedure may require an exponential number of recursive calls. As Algorithm 1 visits all the nodes in m at least once, the number of recursive calls needed to traverse the process is at least linear in the number of nodes in m (i.e., its computational complexity is $\Omega(|N|)$). Since all the nodes following a merge gateway lie on diverse paths and, thus, they are visited multiple times, in the worst case Algorithm 1 takes $O(|N| \cdot 2^{|G|}) = O(|N| \cdot 2^{|N|})$ steps, where $|G|$ is the number of gateways in m .

Algorithm 1 is complete since all kinds of flow nodes described in Def. 2.1 are considered and the procedure always moves from one node to all its immediate successors, thus traversing all nodes and paths. Traversing process loops at most once is enough for Algorithm 1 to retrieve all the distinct predecessors of a node in any possible path (as traversing a process loop more than once would lead to a longer path composed of the same activities).

As a result, relation **Pred** and attribute **label** are populated with all possible and available data. Correctness of the algorithm may be deduced by observing that all recursive calls stop when the end event is reached or when a loop is traversed for the second time. Thus, the algorithm always ends. Moreover, only activity labels are stored in the database, as required.

Finally, taking advantage of structured process models, labels apply only between corresponding exclusive gateways.

7.2.2 Querying Connected Process and Data Models

An important aspect of conceptual modeling approaches is to provide some form of reasoning about the designed models. In this section, we show how to discover the data perspective of a process by answering requirements *R1–R4*. Specifically, we focus on properties that cannot be easily observed on a BPMN diagram alone, but originate from its data perspective and are useful to improve process understanding and re-design.

To query the database of Fig. 7.8 we take advantage from the expressive power of the *tuple relational calculus*, which is a formal, declarative variant of first order logic predicate calculus which allows one to express first order queries on a given relational schema [187]. Relation **DataOp** includes set attributes C_{set} and A_{set} , that are complex valued attributes. Similarly, attribute **preds** of relation **Pred** is a set of activities.

As previously mentioned, in order to deal with such complex values, we rely on the extended version of the relational calculus presented in [187, 188]. In this first-order, many-sorted calculus variables may denote *sets* and, thus, quantification over sets is allowed. The three binary predicates = (equality), \in (membership), and \subseteq (set containment) are included, thus allowing one to express conditions such as “element e belongs to A ”, where A is a set of elements. For tuple comparison, we adopt the well-known principle of *deep equality*.

Prior to showing query formalization, we introduce a couple of shortcuts to express relational calculus formulas frequently used in the following queries.

- Given \mathbf{a} and \mathbf{do} , the following formula checks that \mathbf{a} is an activity and \mathbf{do} is a data operation on \mathbf{a} :

$$DataOpOf(\mathbf{a}, \mathbf{do}) \equiv \exists \mathbf{co} (\text{Activity}(\mathbf{a}) \wedge \text{DataOp}(\mathbf{do}) \wedge \text{ComposedOf}(\mathbf{co}) \wedge \mathbf{a}.avID = \mathbf{co}.avID \wedge \mathbf{co}.opID = \mathbf{do}.opID).$$
- Given \mathbf{a} and \mathbf{a}' , the following formula checks whether there exists a process path where \mathbf{a} is encountered before \mathbf{a}' (or equivalently \mathbf{a}' comes after \mathbf{a}):

$$SuccessorOf(\mathbf{a}, \mathbf{a}') \equiv \exists \mathbf{pr} (\text{Activity}(\mathbf{a}) \wedge \text{Activity}(\mathbf{a}') \wedge \text{Pred}(\mathbf{pr}) \wedge \mathbf{pr}.fnID = \mathbf{a}'.fnID \wedge \mathbf{a}.fnID \in \mathbf{pr}.preds).$$

The following queries are examples of different kinds of properties that can be discovered.

Queries **Q1** and **Q2** focus on identifying different activities that have equivalent data perspectives which can be considered for re-use, as explained by *R1*.

When comparing Activity Views one may check whether (i) all of their attributes coincide, as realized by **Q1**, or (ii) there exist activities that perform data operations that are a subset of those performed by another activity.

Q1 Are there any two distinct process activities a and a' that perform exactly the same data operations d ?

$$\begin{aligned} & \{a, a', d \mid \text{Activity}(a) \wedge \text{Activity}(a') \wedge \text{DataOpOf}(a,d) \wedge a \neq a' \wedge \\ & \forall do' (\text{DataOpOf}(a,do) \Rightarrow \exists do' (\text{DataOpOf}(a',do') \wedge do.cSet = do'.cSet \wedge \\ & do.aSet = do'.aSet \wedge do.accessType = do'.accessType \wedge \\ & do.accessTime = do'.accessTime \wedge do.numInstances = do'.numInstances)) \wedge \\ & \forall do' (\text{DataOpOf}(a',do') \Rightarrow \exists do (\text{DataOpOf}(a,do) \wedge do'.cSet = do.cSet \wedge \\ & do'.aSet = do.aSet \wedge do'.accessType = do.accessType \wedge \\ & do'.accessTime = do.accessTime \wedge do'.numInstances = do.numInstances))\} \end{aligned}$$

Variants of query **Q1** may be defined to compare other aspects of Activity Views by relaxing equality on selected attributes.

Query **Q2** considers also the distribution of activities across alternative process paths (i.e., activities sharing at least one contradictory literal in their labels) beside similarity among data operations, thus addressing both *R1* and *R2*.

Q2 Are there pairs of activities lying on alternative process paths that have the same Activity View?

$$\begin{aligned} & \{a, a', d \mid \text{Activity}(a) \wedge \text{Activity}(a') \wedge \text{DataOpOf}(a,d) \wedge a \neq a' \wedge \\ & \exists l (l \in a.label \wedge \neg l \in a'.label) \wedge \\ & \forall do' (\text{DataOpOf}(a,do) \Rightarrow \exists do' (\text{DataOpOf}(a',do') \wedge \\ & do.cSet = do'.cSet \wedge do.aSet = do'.aSet \wedge do.accessType = do'.accessType \wedge \\ & do.accessTime = do'.accessTime \wedge do.numInstances = do'.numInstances) \\ &) \wedge \forall do' (\text{DataOpOf}(a',do') \Rightarrow \exists do (\text{DataOpOf}(a,do) \wedge \\ & do.cSet = do'.cSet \wedge do.aSet = do'.aSet \wedge do.accessType = do'.accessType \wedge \\ & do.accessTime = do'.accessTime \wedge do.numInstances = do'.numInstances))\} \end{aligned}$$

Example 7.3. By applying **Q2** to the process of Fig. 7.6, labeled as shown in Fig. 7.9, we obtain ‘C, l’ as resulting activities. Despite having identical Activity Views, activities D and E are not returned being both labeled with ‘p’.

Query **Q2** is particularly useful in the context of well-structured process design. When following a well-structured and modular design approach, it may be necessary to repeat certain process elements or entire blocks within alternative paths, in order to maintain the correct nesting of SESE regions [107]. Usually, re-usable elements such as call activities may be defined [11], but this is not always the case. Query **Q2** provides a hint for improving the definition of re-usable

process activities, under a data perspective.

Queries **Q3–Q5** combine information related to process paths and data accesses, focusing on when and how data are needed in the process flow. These queries about data usage respond to requirement **R3**.

For instance, query **Q3** identifies which is the last activity to read a certain data class along a certain path. By changing the value of attribute `accessType` in **Q3** we may also consider writing access.

Q3 Which are the last activities a of the process to *read* a certain data class x ?

$$\{a \mid \text{Activity}(a) \wedge \exists \text{do}, c \text{ (DataOpOf}(a, \text{do}) \wedge \text{Class}(c) \wedge c \in \text{do.cSet} \wedge \\ c.\text{classname} = 'x' \wedge \nexists a', \text{do}' \text{ (DataOpOf}(a', \text{do}') \wedge \text{SuccessorOf}(a, a') \wedge \\ c \in \text{do}'.\text{cSet} \wedge \text{do}.\text{accessType} = 'R' \wedge \text{do}.\text{accessType} = \text{do}'.\text{accessType}))}\}$$

Example 7.4. Which are the last activities of the process of Fig. 7.6 to read class Interview? By applying **Q3** after replacing x with Interview, we obtain C and I.

The dual version of **Q3** retrieving the *first* activity a to access a certain data class x can be obtained by ensuring that none of the activities in the set(s) `preds` associated to a have access to x .

Query **Q4** considers the dependencies among reading/writing data operations and process activities in order to check whether data written by the process are needed afterwards by other activities. **Q4** returns all activities that read data classes that have been inserted/updated by a previous activity named y .

Q4 Which are the successor activities of an activity named y that read exactly the same data classes inserted or updated by y ?

$$\{a, a' \mid \text{Activity}(a) \wedge \text{Activity}(a') \wedge \\ \exists \text{do}' \text{ (DataOpOf}(a', \text{do}') \wedge \text{SuccessorOf}(a', a) \wedge a'.\text{name} = 'y' \wedge \\ (\text{do}'.\text{accessType} \in \{I, U\} \Rightarrow \\ \exists \text{do} \text{ (DataOpOf}(a, \text{do}) \wedge \text{do}.\text{accessType} = 'R' \wedge \forall c (c \in \text{do}'.\text{cSet} \Rightarrow c \in \text{do}.\text{cSet})))}\}$$

Being formulated from the standpoint of data classes, query **Q5** generalizes **Q4** by retrieving all data classes that are modified by some activity of the process and may be read by activities of the same process, if certain paths are chosen.

Q5 Which data classes are written by a process activity and read by a successive one along at least one process path?

$$\{c \mid \text{Class}(c) \wedge \exists a', \text{do}' \text{ (DataOpOf}(a', \text{do}') \wedge c \in \text{do}'.\text{cSet} \wedge \\ (\text{do}'.\text{accessType} \in \{I, U\} \Rightarrow \exists a, \text{do} \text{ (DataOpOf}(a, \text{do}) \wedge \text{SuccessorOf}(a', a) \wedge \\ \text{do}.\text{accessType} = 'R' \wedge c \in \text{do}.\text{cSet}))}\}$$

Query **Q6** retrieves all data classes that are accessed by potentially concurrent activities (i.e., located on parallel paths), thus addressing *R4*.

Q6 Which are the data classes that are accessed in writing mode by at least two concurrent process activities?

$$\{c, a, a' \mid \exists do, do' (DataOpOf(a, do) \wedge DataOpOf(a', do') \wedge c \in do.cSet \wedge c \in do'.cSet \wedge \#l (l \in a.label \wedge \neg l \in a'.label) \wedge \neg SuccessorOf(a, a') \wedge \neg SuccessorOf(a', a) \wedge do.AccessType \in \{I, U, D\} \wedge do'.AccessType \in \{I, U, D\})\}$$

Example 7.5. When applied to the process of Fig. 7.6, **Q6** returns ‘DiagExam, E, D’, ‘Report, E, D’.

Ultimately, queries **Q7** and **Q8** consider operations that are always performed, thus addressing both *R2* and *R3*. **Q7** retrieves all the data classes that are accessed in all process paths, by considering the following possibilities: (i) either one class is accessed by an activity labeled with \square (i.e., all process paths contain that activity) or (ii) it is accessed by multiple activities located on different paths and such that the conjunction of their labels corresponds to \square .

Q7 Which data classes are *read* in all process paths?

$$\{c \mid Class(c) \wedge (\exists a, do (DataOpOf(a, do) \wedge c \in do.cSet \wedge do.accessType = 'R' \wedge a.label = '\square') \vee \forall l, a, do ((DataOpOf(a, do) \wedge a.label \neq '\square' \wedge l \in a.label \wedge c \in do.cSet \wedge do.accessType = 'R') \Rightarrow \exists a', do' (DataOpOf(a', do') \wedge \neg l \in a'.label \wedge c \in do'.cSet \wedge do'.accessType = 'R'))))\}$$

Query **Q7** can be adapted with a slight effort to retrieve the data classes that are *updated*, *inserted*, or *deleted* by all process paths. By combining and re-arranging the properties expressed in **Q5** and **Q7**, one can easily find which data classes are modified by a process activity and are *always* read by a successive one, regardless of which process path is taken. This is realized by query **Q8**.

Q8 Which are the data classes that are written by a process activity and *always* read by a successive one?

$$\{c \mid Class(c) \wedge \exists a, do ((DataOpOf(a, do) \wedge c \in do.cSet \wedge do.accessType \in \{I, U\}) \Rightarrow \exists a', do' (DataOpOf(a', do') \wedge SuccessorOf(a, a') \wedge do'.accessType = 'R' \wedge c \in do'.cSet \wedge (a'.label = '\square' \vee \forall l (l \in a'.label \Rightarrow \exists a'' (SuccessorOf(a, a'') \wedge \neg l \in a''.label))))))\}$$

The discussed queries exemplify how to retrieve significant information from the data stored in the designed relational database. Of course, parts of **Q1–Q8** may be suitably combined to address more complex requirements. For example, by combining **Q3** and **Q4** one can retrieve the data classes that are written and read along parallel process paths.

Finally, quantitative queries on data access and manipulation may be formulated for analysis purposes. For instance, a designer may be interested in knowing

which are the classes of the database accessed more frequently. Frequency of access to a certain data class can be measured by considering (a) the number of data operations performed on that class, (b) the number of activities using it, or (c) the number of paths in which the class is needed.

Although counting is beyond the expressive power of the tuple relational calculus, extensions with aggregate functions have been introduced in [189]. Below, we introduce query **Q9** in tuple relational calculus, intentionally using the word *count* to express the corresponding aggregate function.

Q9 Which are the data classes accessed more frequently?

The frequency of access can be measured by considering (a) the number of data operations, (b) the number of activities, or (c) the number of paths.

(a) According to the highest number of data operations.

$$\{c \mid \text{Class}(c) \wedge \nexists c' (\text{Class}(c') \wedge \text{count}\{\text{do} \mid \text{DataOp}(\text{do}) \wedge c' \in \text{do.cSet}\} > \text{count}\{\text{do} \mid \text{DataOp}(\text{do}) \wedge c \in \text{do.cSet}\})\}$$

(b) According to the highest number of (distinct) activities.

$$\{c \mid \text{Class}(c) \wedge \nexists c' (\text{Class}(c') \wedge \text{count}\{a \mid \text{Activity}(a) \wedge \exists \text{do} (\text{DataOpOf}(a,\text{do}) \wedge c' \in \text{do.cSet})\} > \text{count}\{a \mid \text{Activity}(a) \wedge \exists \text{do} (\text{DataOpOf}(a,\text{do}) \wedge c \in \text{do.cSet})\})\}$$

(c) According to the highest number of (distinct) paths.

$$\{c \mid \text{Class}(c) \wedge \nexists c' (\text{Class}(c') \wedge \text{count}\{l \mid \exists a,\text{do} (\text{LabelingOf}(a,l) \wedge \text{DataOpOf}(a,\text{do}) \wedge c' \in \text{do.cSet})\} > \text{count}\{l \mid \exists a,\text{do} (\text{LabelingOf}(a,l) \wedge \text{DataOpOf}(a,\text{do}) \wedge c \in \text{do.cSet})\})\}$$

Despite counting is beyond the expressive power of the tuple relational calculus, the proposed approach can be easily implemented with a relational database that can be queried in SQL.

Using SQL as query language is beneficial for several reasons. First of all, SQL is the standard language for querying relational database management systems. Besides, SQL is more than relationally complete, that is, it allows one to express queries that cannot be expressed in tuple relational calculus.

An example of such queries are all those that make use of aggregate functions, such as the COUNT operator needed to express the variants (a)–(c) of query **Q9**.

As a proof of concept implementation, we created a database by using PostgreSQL, a well-known open-source relational database management system [190]. We made a few changes to the relational schema of Fig. 7.8 and implemented the database by translating all fields containing complex valued attributes into many-to-many relations.

For example, the predecessors of one activity, originally captured by complex attribute `preds: {Activity}` in Fig. 7.8, are now represented by a relation `NewPred(activity, predecessor)` that associates to each activity one or more preceding activities, depending on its position in the process. The schema of the implemented relational database is shown in Fig. 7.10.

1. Process(processid, name, documentation)
2. DataNode(dnid, processid)
3. DataStore(dnid, dsname, capacity)
4. DataObject(dnid, name, iscollection)
5. DataFlow(fnid, dnid, direction)
6. FlowNode(fnid, processid)
7. SequenceFlow(from, to)
8. Activity(fnid, activityname, isreusable, isatomic, type, hasboundary, boundaryeventrefs*, avID*)
9. Label(activityid, literal)
10. Gateway(fnid, gatewayname, routingType, splittype, hasquestion, question*)
11. Event(fnid, eventName, type, position, triggertype, isboundary, boundarytype*)
12. NewPred(activity, predecessor)
13. ActivityView(avid)
14. ComposedOf(avid, opid)
15. DataOp(opid, accesstype, accesstime, mininstances, maxinstances)
16. Cset(opid, attributename, classname, dbid)
17. Aset(opid, associationname, dbid)
18. DomainDB(dbid, namedb)
19. AssociationDB(associationname, dbid, leftcardinality, rightcardinality)
20. ClassDB(classname, dbid, assocclass, assocRef)
21. AttributeDB(attributename, belongstoclass, dbid, datatype)

Fig. 7.10: Relational schema derived from the one of Fig. 7.8, but having complex valued attributes translated into many-to-many relations.

Relations regarding the process were populated with the help of a BPMN process parser, designed to read the XML file of a BPMN process and to perform insertions in the database, implemented by using the BPMN model API developed by Camunda [25].

Similarly, we implemented Algorithm 1 to populate relations regarding the process structure, i.e., relations `Label` and `NewPred`.

The remaining relations of the database, i.e., those related to data operations and data classes, were populated by directly inserting data in the database through SQL statements.

As an example, below we report some of the queries `Q1–Q9` written in SQL and show the results obtained for the process model and database schema of

Figure 7.7. For simplicity, we omit the part of the query related to the selection of the process and of the domain database.

Query **Q2** identifies which activities lying on alternative process paths have the same activity view. In **Q2-SQL**, we may distinguish the following two main steps.

- (1) The selection of activities belonging to alternative paths. This requires checking that the labels associated to any two activities of the process have at least one literal in opposition. In our database we implemented literals as integers, where 0 stands for \square and opposite integers represent opposite literals (e.g., 1 and -1 stand for p and $\neg p$, respectively). Thus, we must check that the labels ℓ_1 and ℓ_2 of any two activities a_1 and a_2 are not equal to zero (i.e., $\ell_1 \neq 0$ and $\ell_2 \neq 0$) and contain opposite literals (i.e., $\exists l | l \in \ell_1 \wedge \neg l \in \ell_2$).
- (2) The selection of activities having the same activity view. This step requires a bi-directional comparison for any two activities to ensure that all the data operations of the first one are performed by the second one, and viceversa. To this end, all the details of every data operation are compared. Since data operations include class sets and association sets, we should include the respective relations in the comparison. In detail, (2.i) **Aset** relates **dataOp** and **AssociationDB**; (2.ii) **Cset** relates **DataOp** and **AttributeDB**;

Then, we use PostgreSQL aggregate function `string_agg` to concatenate the resulting classes with attributes in the cset and associations in the aset, and to put symbol `*` when all the attributes of one class are accessed, according to the format prescribed by the Activity View.

Q2-SQL Are there pairs of activities lying on alternative process paths that have the same Activity View?

WITH Q2 AS (

```
SELECT DISTINCT A1.fnid AS A1, A2.fnid AS A2, C.classname, C.attribute, A.association,
  DOP.accesstime, DOP.accesstype, DOP.mininstances AS MIN,
  DOP.maxinstances AS MAX
```

```
FROM label L1 JOIN activity A1 ON A1.fnid = L1.activityid JOIN composedof CO ON
  A1.avid=CO.avid JOIN dataop DOP ON CO.opid = DOP.opid JOIN cset C ON
```

```
  DOP.opid = C.opid LEFT JOIN aset A ON A.opid = DOP.opid, activity A2, label L2
```

```
WHERE A2.fnid = L2.activityid AND A1.fnid > A2.fnid
```

```
(1) AND L1.literal <> 0 AND L2.literal <> 0 AND L1.literal = -1 * L2.literal AND
```

```
(1)
```

```
NOT EXISTS(SELECT 1 -- first direction of comparison
```

```
  FROM composedof CO1 JOIN dataop DO1 ON CO1.opid = DO1.opid
```

```
  LEFT JOIN aset AS1 ON DO1.opid = AS1.opid
```

```
  WHERE CO1.avid = A1.avid AND
```

```
  NOT EXISTS(SELECT 1
```

```
    FROM Composedof CO2 JOIN Dataop DO2 ON CO2.opid = DO2.opid
```

```
    LEFT JOIN Aset AS2 ON DO2.opid = AS2.opid
```

```

(2.i) WHERE CO2.avid = A2.avid AND ((AS1.opid IS NULL AND AS2.opid
IS NULL) OR (AS1.opid IS NOT NULL AND AS2.opid IS NOT NULL
AND AS1.association = AS2.association AND AS1.dbid = AS2.dbid)) AND
DO1.mininstances = DO2.mininstances AND
DO1.maxinstances = DO2.maxinstances AND
DO1.accesstype = DO2.accesstype AND
DO1.accesstime = DO2.accesstime AND
(2.ii) NOT EXISTS(SELECT 1
FROM cset C1, cset C2
WHERE C1.opid = DO1.opid AND C2.opid = DO2.opid AND
C1.opid = C2.opid AND C1.classname = C2.classname AND
C1.dbid = C2.dbid AND C1.attribute = C2.attribute)))
AND NOT EXISTS(SELECT 1 -- second direction of comparison
FROM composedof CO2 JOIN dataop DO2 ON CO2.opid = DO2.opid
LEFT JOIN aset AS2 ON DO2.opid = AS2.opid
WHERE CO2.avid = A2.avid AND
NOT EXISTS(SELECT 1
FROM composedof CO1 JOIN dataop DO1 ON CO1.opid = DO1.opid
LEFT JOIN aset AS1 ON DO1.opid = AS1.opid
(2.i) WHERE CO1.avid = A1.avid AND ((AS1.opid IS NULL AND AS2.opid
IS NULL) OR (AS1.opid IS NOT NULL AND AS2.opid IS NOT NULL
AND AS1.association = AS2.association AND AS1.dbid = AS2.dbid)) AND
DO1.mininstances = DO2.mininstances AND
DO1.maxinstances = DO2.maxinstances AND
DO1.accesstype = DO2.accesstype AND
DO1.accesstime = DO2.accesstime AND
(2.ii) NOT EXISTS(SELECT 1
FROM cset C1, cset C2
WHERE C1.opid = DO1.opid AND C2.opid = DO2.opid AND
C1.opid = C2.opid AND C1.classname = C2.classname AND
C1.dbid = C2.dbid AND C1.attribute = C2.attribute))),
TMP AS ( -- aggregates attributes in a list for each class of Q2
SELECT A1, A2, classname, association, accesstime, accesstype, MIN, MAX,
'(' || string_agg(attribute, ', ' ORDER BY attribute) || ') ' AS attributes
FROM Q2
GROUP BY A1, A2, classname, association, accesstime, accesstype, MIN, MAX),
TMP2 AS ( -- aggregates attributes in a list for each class of the database
SELECT C.classname, '(' || string_agg(attributename, ', ' ORDER BY attributename) || ') '
AS class_attributes
FROM clasddb C JOIN attributedb A ON (A.belongstoclass, A.dbid) = (C.classname,
C.dbid)
GROUP BY classname)

```

-- The last query visualizes the result by aggregating associations in aset and putting a * if all attributes of one class are accessed

```

SELECT A1, A2, '{ || string_agg(DISTINCT association, ', ' order by association)||}'
  AS aset, accesstime, accesstype, MIN, MAX, '{ ' || string_agg(T.classname ||
(CASE WHEN T.attributes = T2.class_attributes THEN '(*)'
  ELSE t.attributes END) , ', ') || '}' AS cset
FROM tmp T JOIN tmp2 T2 ON (T.classname = T2.classname)
GROUP BY A1, A2, accesstime, accesstype, MIN, MAX;

```

a1	a2	aset	accesstime	accesstype	min	max	cset
14	5		during	insertion	1	*	{Monitoring(*)}
14	5		end	update	1	1	{Surgery(opReport)}
14	5		start	insertion	1	*	{Treatment(*)}
14	5		start	insertion	1	1	{Surgery(date, opReport, place, reason, time)}
14	5	{anamnesis}	start	read	1	*	{Admission(*), Interview(*)}

(5 rows)

In our database, activity with fnid equal to 14 is Appendectomy, whereas activity with fnid equal to 5 is Surgery.

Despite being alternative activities, Appendectomy and Surgery are both surgical operations and, thus, they access and manipulate the same information.

Query **Q5-SQL** takes any two activities accessing the same data class, and checks whether the one with writing access to the class is among the predecessors of the other one, which has reading access on the class.

Q5-SQL Which data classes are written by a process activity and read by a successive one along at least one process path?

```

SELECT DISTINCT C1.classname, A1.fnid, A1.activityname AS WRITTEN_BY, A2.fnid,
  A2.activityname AS READ_BY
FROM cset C1 JOIN dataop D1 ON C1.opid = D1.opid JOIN composedof CO1 ON
  D1.opid = CO1.opid JOIN activity A1 ON CO1.avid = A1.avid, cset C2 JOIN
  dataop D2 ON C2.opid = D2.opid JOIN composedof CO2 ON D2.opid = CO2.opid
  JOIN activity A2 ON CO2.avid = A2.avid
WHERE D1.accesstype IN ('update', 'insertion') AND D2.accesstype ILIKE 'read' AND
  C1.classname = C2.classname AND
  A1.fnid IN(SELECT P.predecessor
    FROM NewPred P
    WHERE P.activity = A2.fnid);

```

Query **Q6-SQL** looks for data classes that are written by any two concurrent activities. The retrieval of the data class is similar to the one done in **Q5-SQL**. To establish whether two activities are concurrent, we must ensure that their labels differ from \square (i.e., 0) and do not have any literal in opposition.

classname	fnid	written_by	fnid	read_by
DiagExam	8	abdominal ultrasound	11	diagnosis
DiagExam	9	blood tests and urinalysis	11	diagnosis
Diagnosis	11	diagnosis	16	discharge
Interview	2	patient interview	5	surgery
Interview	2	patient interview	14	appendectomy
Report	8	abdominal ultrasound	11	diagnosis
Report	9	blood tests and urinalysis	11	diagnosis
Surgery	14	appendectomy	16	discharge

(8 rows)

Q6-SQL Which are the data classes that are accessed in writing mode by at least two concurrent process activities?

```

SELECT DISTINCT C1.classname, A1.fnid, A1.activityname, A2.fnid, A2.activityname
FROM cset C1 JOIN dataop D1 ON C1.opid = D1.opid JOIN composedof CO1 ON
  D1.opid = CO1.opid JOIN activity A1 ON CO1.avid = A1.avid, cset C2 JOIN
  dataop D2 ON C2.opid = D2.opid JOIN composedof CO2 ON D2.opid = CO2.opid
WHERE A1.fnid < A2.fnid AND D1.accesstype IN ('update', 'insertion', 'delete') AND
  D2.accesstype IN ('update', 'insertion', 'delete') AND
  C1.classname = C2.classname AND
  NOT EXISTS (SELECT 1 --check whether activities are concurrent
    FROM Label L1, Label L2
    WHERE L1.activityid = A1.fnid AND L2.activityid = A2.fnid AND
      L1.literal <> 0 AND L2.literal <> 0 AND L1.literal -1*L2.literal);

```

classname	fnid	activityname	fnid	activityname
DiagExam	8	abdominal ultrasound	9	blood tests and urinalysis
Report	8	abdominal ultrasound	9	blood tests and urinalysis

(2 rows)

Finally, **Q9.(b)-SQL** counts the number of activities that access each class and, then, selects the data classes that are accessed by the highest number of activities.

Q9.(b)-SQL Which are the data classes accessed more frequently, based on the number of distinct activities?

```

WITH NumActivitiesPerClass AS( --counts the number of activities that access
  each class
SELECT C.className, COUNT(DISTINCT A.fnID) AS NumActivities
FROM activity A JOIN composedOf CO ON CO.avID = A.avID JOIN dataOp D ON
  CO.opID = D.opID JOIN Cset CS ON D.opid = CS.opid JOIN classdb C ON
  (C.className, C.dbid) = (CS.classname, CS.dbid)

```

```

GROUP BY C.className);

SELECT className, NumActivities
FROM NumActivitiesPerClass
WHERE NumActivities IN(SELECT MAX(NumActivities) FROM NumActivitiesPerClass);

```

classname	numactivities
Admission	4
Surgery	4

(2 rows)

All the other queries presented in this section, may be translated in SQL as done for the exemplified ones and their results may be used for process analysis, as well as, for visualization purposes.

7.3 Conclusion

In this chapter, we introduced and formalized the Activity View, a novel approach aimed to realize the connection between a process model and a conceptual database schema, while allowing designers to also detail data operations performed by process tasks.

In Sect. 7.1, we showed how using the Activity View allows one to obtain interesting insights related to the connected perspectives. Then, in Sect. 7.2, we described an approach based on the relational database model to capture and query the data perspective of business processes. By exploring data manipulations realized by process activities, our approach provides designers with a flexible overview of how data are used by a process, fostering process (re)-design and improvement.

For future work, we aim to continue with the implementation of the relational framework proposed in Sect. 7.2 and, ideally, design a graphical tool that may be integrated into existing process editors.

Besides, we aim to extend the concept of Activity View presented for process tasks in order to represent the data perspective of different process abstraction levels, starting from sub-processes. Reaching this goal would require looking into advances in the field of process model abstraction [191].

Conceptual Modeling of Inter-dependencies between Processes and Data

This chapter is based on results published in [56].

In this chapter, we show how the Activity View can be used to detect possible inconsistencies that may arise between linked process models and data schemata.

The conceptual modeling of the link between processes and data brings many benefits. It bridges the gap among stakeholders involved in process and data management, and enables reasoning on a previously “hidden” informational perspective of the process.

By supporting the visualization of different data operations performed by a process and their properties, the Activity View reveals a novel dimension affecting process execution that cannot be disregarded during analysis. Indeed, most activities require data to be executed and routing choices are typically based on data, thus making the control flow data-dependent.

In this chapter we propose an approach to detect at design time possible inconsistencies arising among data access operations performed during process execution. More specifically, we intend to ensure that data operations performed by different process activities are consistent with each other, by considering their ordering dictated by the process control flow and the kind of access to data.

Our final goal is to support the detection and prevention of data flaws in process models that could otherwise remain undiscovered until run-time.

The remainder of the chapter is structured as follows. Sect. 8.1 provides some background concepts, while Sect. 8.2 introduces the foundations of our approach. Sect. 8.3 addresses consistency between different data access operations, at different abstraction levels. Finally, Sect. 8.4 concludes the chapter.

8.1 Introduction and Motivation

To begin with, we briefly introduce a suitable motivating example taken from the domain of emergency medicine.

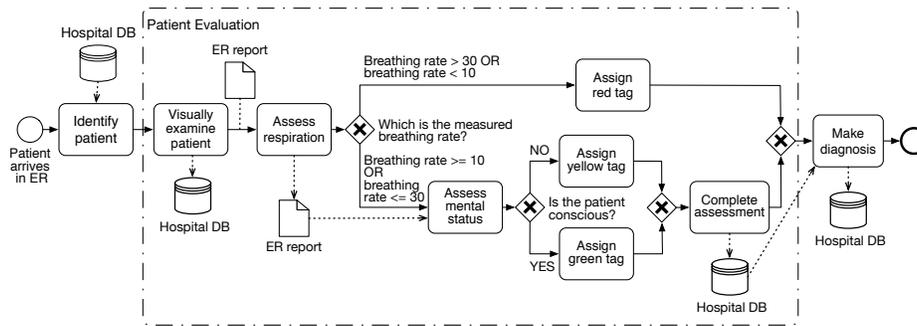


Fig. 8.1: BPMN process model depicting the main steps of Emergency Room triage, performed in hospital to prioritize and classify patients according to the severity of their conditions.

Let us consider the BPMN process shown in Fig. 8.1, depicting a few essential steps of Emergency Room (ER) triage.

Example 8.1 (Emergency Room Triage). Triage is the clinical procedure of prioritizing incoming patients for treatment, according to the seriousness of their condition, which is executed by expert triage nurses who cooperate with physicians when necessary. Color tags are used to summarize the gravity of patient conditions and the average time to wait prior to receiving treatment.

The process of Fig. 8.1 begins when a Patient arrives in ER. Identify patient is the first task to be executed and requires that basic information regarding the patient’s generalities is obtained from the hospital database (Hospital DB). After having collected important information about the cause of the admission, specialized nurses proceed to Visually examine patient. During this phase, the presence of evident trauma or injuries is recorded in the hospital database, together with a brief description of the factors provoking hospital admission.

Then, patient evaluation continues with a stepwise assessment, starting with task Assess Respiration. All the details gathered during patient assessment are recorded in the ER report, a document that is exchanged and updated along with patient management, prior to be inserted in the hospital database upon assessment completion.

Depending on the answer to question Which is the measured breathing rate?, the process flow is directed towards task Assign red tag, if the breathing rate is greater than 30 breaths per minute or lower than 10 breaths per minute, or towards task Assess mental status otherwise. Then, depending on the degree of patient consciousness, nurses choose to Assign yellow tag or to Assign green tag.

Then, nurses can Complete assessment and all the gathered data are recorded in the Hospital DB. Finally, based on the gathered information and degree of urgency, a physician is called in to Make diagnosis. Depending on the diagnosed

problem, which is recorded in the Hospital DB, the patient is provided most appropriate treatment directly in ER or in the designated hospital ward.

In the scenario described by Example 8.1, the role of a common information system is crucial, as data are needed to execute process activities and persistent information must be shared between different hospital wards.

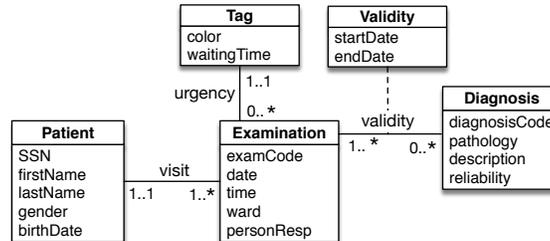


Fig. 8.2: UML class diagram depicting a conceptual data schema of a small part of a hospital database.

An example of UML class diagram, depicting part of a hospital database related to the example of Fig. 8.1 is shown in Fig. 8.2. *Classes* Patient and Examination are linked by *association* visit. According to the represented *multiplicity*, a patient can be examined once or many times, whereas each examination corresponds exactly to one patient. *Association class* Validity contains attributes startDate and endDate that characterize the association.

8.2 Foundational Concepts

In this section, we recall and formalize some key notions regarding process models and data schemata that will be used as a reference for the remainder of this chapter.

In the context of data modeling and database design, conceptual models are used to give an abstract representation of the real world, by providing insights on interesting features of the data needed by an organization [180].

In order to be coherent with the BPMN and UML standards for process and data modeling, we adopt an object-oriented design approach for representing process-related data at a conceptual level. In the remainder of the section, we provide the definitions of data class, data schema, and process trace to specify which are the modeling elements considered in this chapter.

Definition 8.1 (Data Class). A data class $c_i = (a_1, \dots, a_n)$ is identified by a unique name c_i and consists of a list $A = (a_1, \dots, a_n)$ of attributes, such that $n \geq 1$. Each attribute a_i has a unique name within the class c_i it belongs to.

Data classes are organized in a data schema. The following definition formally describes a generic data schema, based on key concepts belonging to the domain of object oriented data modeling [63, 180].

Definition 8.2 (Data Schema). A data schema $DS = (Cl, As, AsC)$ consists of a finite non-empty set $Cl = \{c_1, \dots, c_n\}$ of data classes, a finite set $As = \{as_1, \dots, as_m\}$ of associations between the objects of Cl , and a finite set $AsC = \{asc_1, \dots, asc_m\}$ of associations classes. A multiplicity $m = (m_{min} .. m_{max})$ is assigned to each class involved in an association, where m_{min} (m_{max}) denotes the minimum (maximum) number of objects of that class that can participate in the association. Allowed multiplicity values are ‘0’, ‘1’, ‘*’, where the asterisk * is used for representing no maximum limit on participation.

Fig. 8.2 shows a data schema $DS = (Cl, As, AsC)$, where $Cl = \{Patient, Examination, Diagnosis, Tag\}$, $As = \{visit, urgency, \}$ and $AsC = \{Validity\}$.

As for process models, we refer to Def. 2.1 and consider dealing with well-structured processes (cf. Sect. 2.1.2). Without loss of generality, we assume that given a process model $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \rho, \mathcal{L})$, $R = \emptyset$, i.e., we overlook the resource perspective of the process.

Finally, in order to understand the execution behavior of a process, it is important to define the traces of a given process model, which represent all the possible executions of the model [192, 193].

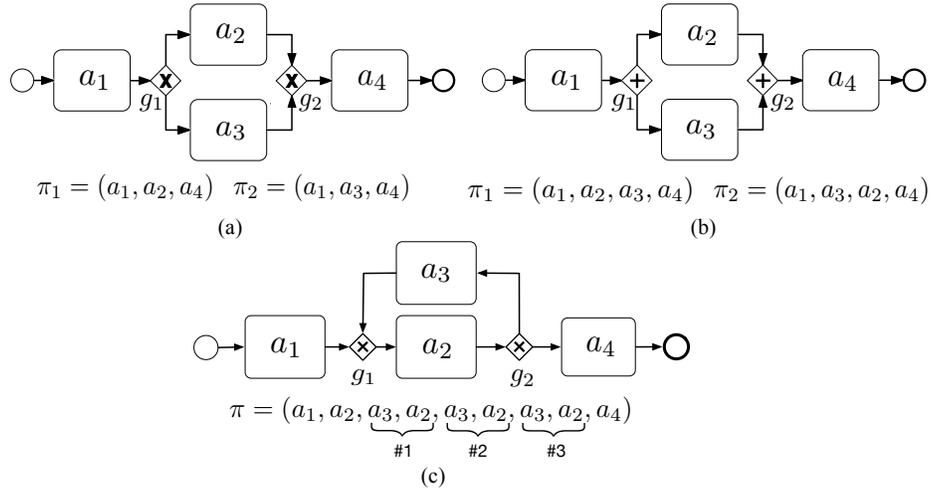


Fig. 8.3: Example of process traces in case of (a) exclusive gateways, (b) parallel gateways, or (c) loops with a fixed number of iterations (e.g., three).

Definition 8.3 (Process Trace). Given a process model m , a trace $\pi = (a_1, \dots, a_n)$ is any sequence of activity executions, according to the semantics of BPMN [11]. Π_m is the set of all traces of a process model m .

By following trace semantics [192], we deal with process branching for well-structured process models as summarized in Fig. 8.3. That is:

- (a) alternative activities located in process regions delimited by exclusive gateways must necessarily belong to different traces;
- (b) activities that belong to a process region delimited by parallel gateways are allowed to be executed one at a time, that is, each process trace corresponds to a possible interleaving of such activities;
- (c) a loop in the process model is unfolded into a process trace, as we consider having a bounded number of loop iterations (e.g., three in Fig. 8.3(c)).

Finally, we consider Activity Views as a means to connect process models and data schemata. In this chapter, we refine Def. 7.1 in order to focus only on the set of classes accessed by a certain process activity, i.e., the C_{set} but abstracting from attributes, and on the type of access to the identified portion of the database, i.e., the *Access-type*. In addition, we assume the tuples composing one activity view to follow a given order.

Definition 8.4 (Simplified Activity View). Let us consider dealing with a process model $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$ (cf. Def. 2.1). Given an activity $ac \in A \subseteq N$, its Activity View $av_{ac} = (t_1, \dots, t_n)$ is a *sequence* of tuples, where each tuple t_i denotes a particular data access operation performed by activity ac given database schema. The latter is composed of a set of classes Cl , a set of associations As , and a set of association classes AsC .

Each tuple $t_i = \langle Cnames, AccessType \rangle$ denotes a particular data access operation to a set of data classes, where:

- $Cnames_i = \{c_1, \dots, c_j\} \subseteq (Cl \cup AsC)$, is the set of connected classes accessed by process activity ac . Each class c_i is represented only by its unique name.
- $AccessType_i \in \{R, I, D, U\}$ defines the type of access to the related information. R denotes a *read* of elements of C_{set} , whereas I , D , and U denote different kinds of write operations, namely I indicates an *insertion*, D stands for a *deletion*, while U denotes an *update*.

As introduced in Chapter 7, Activity Views constitute a simple new conceptual “bridge” that allows process designers to identify which is the part of a database affected by process execution, thus suggesting which are the concepts of interest that should be jointly considered by both data and process modelers.

Simplified Activity Views are defined on process activities, regardless of the fact that these may be connected to data nodes in the process models. Indeed, activities are linked to data objects or data stores in order to explicitly represent the process data flow, but other data requirements are often handled during

implementation and are not visualized in process models. Thus, (Simplified) Activity Views complete the picture by allowing designers to detail data operations without requiring the specification of data objects/stores, as we advocate that process designers should feel free choose the preferred level of (data) abstraction.

As an example, let us consider the process depicted in Fig. 8.1.

Activity Evaluate Patient reads from data store Hospital DB, although it is not clear whether nurses rely on the patient’s history, their personal expertise, or domain knowledge to perform the task. Similarly, activity Make Diagnosis shows a writing access to the hospital database, but the representation does not allow one to understand which and how many data operations are performed by the process. Moreover, in the process of Fig. 8.1, data object ER report is exchanged during patient evaluation, but it is impossible to understand if it contains information that is later stored in the Hospital DB. Last but not least, when looking at the data schema of Fig. 8.2, which represents the hospital database, there is no data class containing data related to the ER report. However, it is impossible to determine whether this means that the database was not designed to store ER report data, or that data are saved somewhere else, under a different label.

This lack of clear understanding that encompasses process-related data stems from the fact that activity-centric processes are not meant to focus on data and, often, their level of abstraction does not allow designers to detail data aspects related to activities.

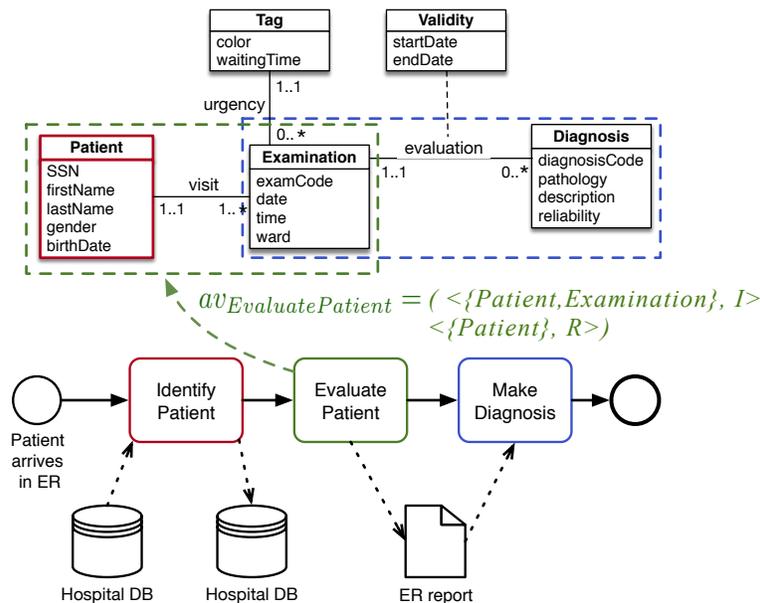


Fig. 8.4: Example of activity view linking task Evaluate Patient to accessible data classes Patient and Examination, highlighted in green.

To overcome this limitation, Simplified Activity Views provide a means to add information about data operations to a process and support the design of activities considering existing data schemata. This approach is useful when a new process needs to be added to a consolidated organizational asset or when processes are re-engineered to fit (new) data requirements.

For instance, let us consider the simple sequential process shown in Fig. 8.4. At a conceptual level, Simplified Activity Views allow one to visualize which are the parts of a data schema accessible by each process activity, as highlighted by the coloured dashed frames that enclose data classes. As an example, Simplified Activity View *avEvaluatePatient* of task **Evaluate Patient** shows that some object(s) subsequently some object(s) of classes **Patient** and **Examination** is inserted, whereas some object(s) of class **Patient** is subsequently read by the task. Similarly, it becomes easier to see that both tasks **Evaluate Patient** and task **Make Diagnosis** have access to data class **Examination**, as highlighted by the colored dashed frames.

In the remainder of the chapter, we show how Simplified Activity Views offer a starting point for designers to identify possible inconsistencies that may exist between process and data diagrams in terms of data access operations.

8.3 Inconsistencies among Data Operations

In this section, we deal with the problem of ensuring that different Simplified Activity Views defined on a process are consistent with each other.

We refer to *inconsistency* among data operations in the broad sense of the word, i.e., we consider various undesired situations that may occur if data operations are not designed correctly, e.g., missing data.

In detail, we address the problem of inconsistent data operations in a process by considering two levels of data abstraction. First, we describe the evaluation of *potential inconsistencies* between data operations, starting by considering the classes of a data schema accessed by multiple process activities. Then, we move to a lower level of data representation and deal with class instances in order to determine which of the identified potential inconsistencies exist in a real process run, based on which are the accessed objects.

In general, process modeling calls for a careful integration of different characteristics. Adding knowledge about data access to a process model enables exploring different properties of data operations performed during process execution.

The control-flow defines the execution order of activities. As a result, data operations are also performed according to a specific order dictated by the process structure. Hence, we need to ensure at design time that data operations performed by different process activities are consistent with each other. That is, beside guaranteeing that data access operations are mapped correctly to a given data schema, it is beneficial to detect which data operations may interfere with each other and lead to undesired situations caused by design mistakes.

Assuming that the ordering of data access operations described by a Simplified Activity View is correct by definition, we take advantage of this conceptual representation to discover sequences of data operations performed by different activities that may be inconsistent with each other, at design time.

Simplified Activity Views provide an overview of which is the part of a data schema used by a process. As a first step, we consider the set of classes involved in each Simplified Activity View and their inter-dependencies. In detail, if different Simplified Activity Views have shared access to one or more data classes, potential inconsistencies may arise between data operations, depending on the type of data access.

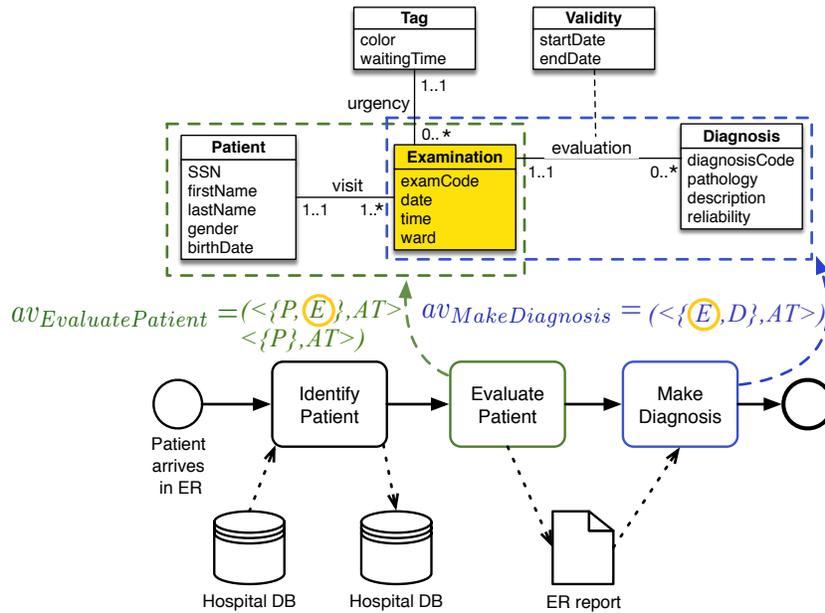


Fig. 8.5: Example of simplified Activity Views for process tasks Evaluate Patient and Make Diagnosis, both operating on highlighted data class Examination (E). When data classes are shared between different Simplified Activity Views, potential inconsistencies between data operations can arise, depending on the Access-Type to data (AT).

For example, consider the simple sequential process of Fig. 8.5, which is an excerpt of the one shown in Fig. 8.4 composed of activities Evaluate Patient and Make Diagnosis annotated with their corresponding Simplified Activity Views.

Since activities Evaluate Patient and Make Diagnosis have shared access to data class Examination, potential inconsistencies can stem from specific orderings of data access operations, depending on which data operation is performed and on which is the order of activities in the process flow.

Table 8.1 summarizes all the potential inconsistencies related to the considered example, by identifying the sequences of data operations performed by activities *Evaluate Patient* and *Make Diagnosis* on the shared data class *Examination* that may lead to undesired situations. For instance, let us suppose that task *Make Diagnosis* of Fig. 8.5 is executed by a physician, who needs to read the details of the latest examination of a certain patient to evaluate the overall clinical picture. If data regarding the examination are deleted previous task *Evaluate Patient*, then the physician cannot properly make a diagnosis because important, required information is missing.

Table 8.1 summarizes all the possible pair combinations of the four data access types *R*, *U*, *D*, *I* performed by two subsequent activities in a process model. Potential inconsistencies (PI) are labeled as (1) – (6).

From the standpoint of the data schema, at this level of abstraction we only deal with data classes, without addressing specific process or data instances. This means that we provide an overview of *potential* inconsistencies only by observing which data classes are accessed by more than one process activity. Such an overview of intersecting data classes, suggests to designers which are the modeled process activities that may be lead to potential data inconsistencies.

Clearly, the intersection of two or more Simplified Activity Views does not necessarily imply that data access operations are performed on the same sets of objects and, thus, not all potential inconsistencies lead to actual design mistakes.

For this reason, it is desirable to analyze class instances. As an example, suppose that an examination is deleted during task *Evaluate Patient*, that is, prior to be read by task *Make Diagnosis*. To state that a *Delete* operation followed by a *Read* is inconsistent, it is necessary to know the identifier of the examination on which the operations are performed. Indeed, a physician can reasonably delete a wrongly inserted examination of a patient A during task *Evaluate Patient*, without affecting the read of an examination of a patient B during task *Make Diagnosis*.

Provided that the inconsistencies introduced in Table 8.1 are only potential, in the following paragraphs we consider data instances and process traces to better detail when undesired situations may occur. This means moving one level down to discover the situations in which potential inconsistencies can become actual, depending on the sets of objects on which data operations are performed.

8.3.1 Discovering Inconsistencies Considering Data Instances

Actual inconsistencies between data operations can be discovered by considering the structure of the process schema, all the admissible execution orderings of activities (i.e., all the process traces), and the related data operations described in Simplified Activity Views.

In this section, we do not only consider data classes, but we evaluate interdependencies between Simplified Activity Views defined on specific sets of objects. In detail, we assume to be given a process model annotated with activity views, a data schema, and the instance of the database at the beginning of process

Evaluate Patient	Make Diagnosis	PI	Evaluate Patient	Make Diagnosis	PI
Read	Read	NO	Update	Read	NO
Read	Delete	NO	Update	Delete	NO
Read	Update	NO	Update	Update	NO
Read	Insert	YES (1)	Update	Insert	YES (5)
Delete	Read	YES (2)	Insert	Read	NO
Delete	Delete	YES (3)	Insert	Delete	NO
Delete	Update	YES (4)	Insert	Update	NO
Delete	Insert	NO	Insert	Insert	YES (6)

Table 8.1: Overview of all possible combinations of data access operations and of potential inconsistencies (PI) that may arise when certain data operations are performed on shared instances of data class *Examination* of Fig. 8.5 by activities *Evaluate Patient* and *Make Diagnosis*. Inconsistencies (1) – (6) hold for any two subsequent tasks that operate on shared data.

execution. Our goal is to use the information regarding data access encoded in Simplified Activity Views to discover inconsistent sequences of data operations.

To this end, given a process model m , we consider the set Π_m of process traces and compare the data access operations included within Simplified Activity Views with the aim of finding erroneous sequences of data operations, at design time.

In the remainder of this section, we use the following simplified notation for individual data operations (i.e., Simplified Activity View tuples): $Access\text{-}Type_{t_j}(d)$, meaning that d is the set of objects accessed by a tuple t_j of a given Simplified Activity View.

Reasoning on process traces allows us to also consider the related sequences of data access operations. In other words, given a process trace, we can directly retrieve the corresponding sequence of data operations by following the order of activities in a trace and the order of data access operations within each activity.

For instance, let us consider the sequential process m of Fig. 8.6. The set of traces is $\Pi_m = \{\pi\}$, with $\pi = (a_1, \dots, a_j, \dots, a_i, \dots, a_n)$. Similarly, we define the corresponding sequence of data access operations as follows.

Definition 8.5 (Sequence of data operations). Given a process trace $\pi = a_1 \dots a_n$, the corresponding sequence of data access operations is the sequence of tuples $\theta = t_a, \dots, t_s$ contained in the Simplified Activity Views $av_{a_1} \dots av_{a_n}$. $\Theta = \{\theta_1, \dots, \theta_n\}$ is the set of all sequences of data access operations θ_i , where each θ_i corresponds to trace $\pi_i \in \Pi$.

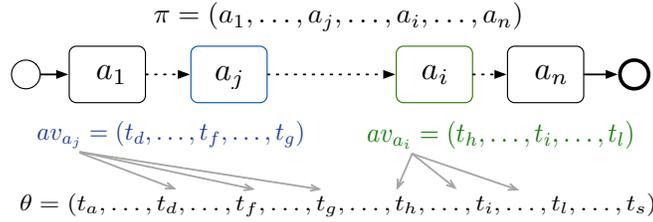


Fig. 8.6: Sequential process having a single trace $\pi = a_1 \dots a_j$ and corresponding sequence Θ of data access operations

We define the instance of the database accessed by the process at the beginning of its execution, as follows.

Definition 8.6 (Initial database instance). *Given a data schema DS of a database, $\Delta_s = \{o_i | o_i \text{ is an object of a class } c_i \in DS\}$ is the instance of DS accessed at the beginning of process execution.*

Then, given the four data access types R, U, D, I , we identify the admissible sequences of operations that do not lead to undesired situations (cf. Fig. 8.7(a)) and derive all the possible sequences of “forbidden” operations, sketched in Fig. 8.7(b). In Fig. 8.7, every transition denotes that two data access operations are performed successively on the same set of objects. For brevity, we denote this consecutiveness with a “ \rightarrow ” that connects two data operations.

For example, starting from $\theta = (I_{t_1}(d_1), R_{t_2}(d_2), U_{t_3}(d_1), R_{t_4}(d_3), D_{t_5}(d_1))$, the sequence of operations performed on the set of objects d_1 is $\theta(d_1) = (I_{t_1}(d_1) \rightarrow U_{t_3}(d_1) \rightarrow D_{t_5}(d_1))$. The latter, does not contain any erroneous (sub-)sequence of data operations, according to Fig. 8.7(b).

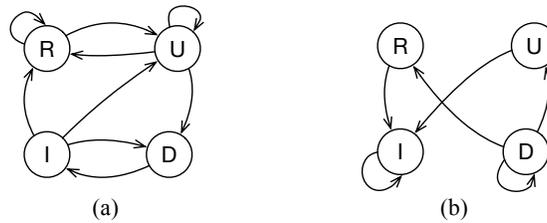


Fig. 8.7: State diagrams representing (a) admissible and (b) forbidden sequences of data access operation. All states R, U, D, I can be both initial and final.

Forbidden transitions between operations performed on the same objects and identified in Fig. 8.7(b) are $R \rightarrow I, D \rightarrow R, D \rightarrow D$, and $D \rightarrow U, U \rightarrow I, I \rightarrow I$, which correspond to the potential inconsistencies of Table 8.1 labeled with (1) – (6).

Given the notions of process trace π and of sequence of data access operations θ , we can retrieve inconsistencies with respect to data operation t_i by starting from t_i and checking if there exists $t_j \in \theta$ preceding t_i such that $t_j \rightarrow t_i$ is a forbidden sequence of data access operation.

To this end, we define the following inconsistency rules **I1**–**I2** that allow us to detect sequences of $t_j \rightarrow t_i$, with $1 < i \leq |\theta|$ and a preceding operation t_j , with $1 \leq j \leq i - 1$, if it exists. When t_i with $i = 1$ is the first data access operation performed by the process, inconsistencies may arise if data exist or not in the initial data instance Δ_s , depending on the type of access. In this latter case, we denote the inconsistency as $\Delta_s \rightarrow t_i$.

I1 – MISSING DATA

DESCRIPTION

The problem of missing data is generated by $D \rightarrow R$, $D \rightarrow U$, and $D \rightarrow D$. Missing data occurs whenever $t_i = \{R(d) \mid U(d) \mid D(d)\}$ belongs to θ and:

- (a) the information needed by t_i was deleted by a previous operation t_j and there is no operation t_k with $j \leq k < i$ between t_j and t_i that inserts it;
- (b) t_i is the first operation of the process and the information needed by t_i is not present in the initial data instance Δ_s .

FORMALIZATION

- (a) $t_i = \{R(d) \mid U(d) \mid D(d)\} \in \theta$ with $1 < i \leq |\theta| \wedge \exists t_j = D(d) \in \theta$ with $1 \leq j \leq i - 1 \wedge \nexists t_k = I(d) \in \theta$ with $j < k < i$;
- (b) $t_i = \{R(d) \mid U(d) \mid D(d)\} \in \theta$ with $i = 1 \wedge d \notin \Delta_s$.

I2 – CREATION OF EXISTING DATA

DESCRIPTION

The problem of creation of existing data is generated by $I \rightarrow I$.

Creation of existing data occurs whenever $t_i = I(d)$ belongs to θ and:

- (a) the information inserted by t_i was already inserted by a previous operation t_j and there is no operation t_k , with $j < k < i$ between t_j and t_i that deletes it;
- (b) t_i with $i = 1$ and the information inserted by t_i is already present in the initial data instance Δ_s .

FORMALIZATION

- (a) $t_i = I(d) \in \theta$ with $1 < i \leq |\theta| \wedge \exists t_j = I(d) \in \theta$ with $1 \leq j < i \wedge \nexists t_k = D(d) \in \theta$ with $j < k < i$;
- (b) $t_i = I(d) \in \theta$ with $i = 1 \wedge d \subseteq \Delta_s$.

CONSIDERATIONS ON d FOR **I1** AND **I2**

Let us call $(d)_{t_j}$ and $(d)_{t_i}$ the sets of objects accessed in t_j and t_i , respectively:

- **I1(I2)** holds if both $(d)_{t_j}$ and $(d)_{t_i}$ are singletons and $(d)_{t_j} = (d)_{t_i}$;
- **I1(I2)** holds if $(d)_{t_j}$ is a set of objects with $|(d)_{t_j}| > 1$ and $(d)_{t_i}$ is a singleton such that $(d)_{t_i} \subset (d)_{t_j}$;
- **I1(I2)** holds if both $(d)_{t_j}$ and $(d)_{t_i}$ are set of objects with $|(d)_{t_j}| > 1$ and $|(d)_{t_i}| > 1$ and $(d)_{t_i} \cap (d)_{t_j} \neq \emptyset$.

Pairs $R \rightarrow I$ and $U \rightarrow I$ are also be detected by **I1** or **I2**, depending on the chosen viewpoint. In particular, if some information d is read or updated prior to be inserted, we have a case of **I1**. Instead, if the information that is read or updated is present, an attempt to insert it again would be inconsistent for **I2**.

To discover all the inconsistencies between data operations, we need to find all the forbidden sequences of data operations that operate on the same or intersecting sets of objects, by checking all the possible traces of a given process.

Algorithm 2: findInconsistencies

```

1: Input: set  $\Theta = \{\theta_1, \dots, \theta_n\}$  of sequences of data operations, with  $t_i, t_j \in \theta_m$ ; starting data instance  $\Delta_s$ 
2: Initialization: set of inconsistencies  $\Gamma := \emptyset$  for all inconsistencies found in  $\Theta$ ;
3: for each  $\theta_m, m := 1$  to  $|\Theta|$  do
4:    $A = \emptyset$ ;
5:   if  $((t_1 == \{R(d) \mid U(d) \mid D(d)\}) \wedge (d \notin \Delta_s))$  then
6:      $A := A \cup \{I1 : \Delta_s \rightarrow t_1\}$ 
7:   end if
8:   if  $((t_1 == I(d)) \wedge (d \in \Delta_s))$  then
9:      $A := A \cup \{I2 : \Delta_s \rightarrow t_1\}$ 
10:  end if
11:  for each  $t_i, i := 2$  to  $|\theta_m|$  do
12:     $continue := true$ ;
13:    for each  $t_j, j := i - 1$  to  $1$  do
14:      if  $((t_i == \{R(d) \mid U(d) \mid D(d)\})$  AND  $(I1.(a)(t_j, t_i) == true) \wedge$ 
         $(continue))$  then
15:         $A := A \cup \{I1 : t_j \rightarrow t_i\}$ 
16:         $continue := false$ ;
17:      else if  $((t_i == I(d)) \wedge (I2.(b)(t_j, t_i) == true) \wedge (continue))$  then
18:         $A := A \cup \{I2 : t_j \rightarrow t_i\}$ 
19:         $continue := false$ ;
20:      end if
21:    end for
22:  end for
23:   $\Gamma = \Gamma \cup \{A\}$ 
24: end for
25: return  $\Gamma$ 

```

Algorithm 2 *findInconsistencies* applies the introduced inconsistency rules **I1** and **I2** to each operation t_i of $\theta_m \in \Theta$ as follows.

- For each operation $t_i \in \theta_m$:
 - if t_i is the first operation of θ_m data instance Δ_s is considered in order to check if the information needed by the operation is present or not;
 - otherwise,
 - if $t_i = \{R(d)|U(d)|D(d)\}$ then, rule **I1** is applied between t_i and a previous t_j that operates on the same d , if such t_j exists;
 - instead, if $t_i = I(d)$ then, rule **I2** is applied between t_i and a previous t_j that operates on the same d , if such t_j exists;
 - if rule **I1(I2)** holds, that is $t_j \rightarrow t_i$ is forbidden, then the result $\{I1(I2) : t_j \rightarrow t_i\}$ is added to the set Λ_i that stores inconsistencies for θ_i .

Finally, the set $\Gamma = \{\Lambda_1, \dots, \Lambda_n\}$ containing all the inconsistencies found in all sequences of operations $\theta_1, \dots, \theta_n$ is returned.

A sample application of Algorithm 2 is provided by considering process model m of Fig. 8.8.

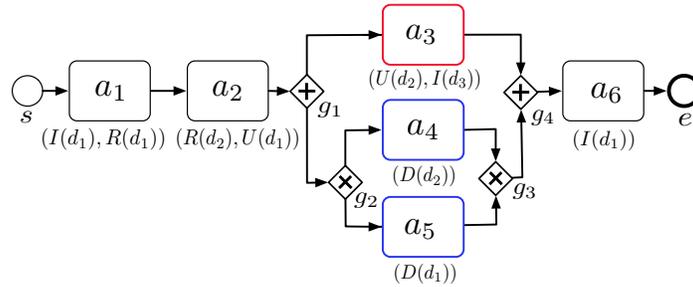


Fig. 8.8: Example of process model, annotated with Simplified Activity Views.

$\Pi_m = \{\pi_1, \pi_2, \pi_3, \pi_4\}$ is the set of process traces, where:

- $\pi_1 = (a_1, a_2, a_3, a_4, a_6)$;
- $\pi_2 = (a_1, a_2, a_4, a_3, a_6)$;
- $\pi_3 = (a_1, a_2, a_3, a_5, a_6)$;
- $\pi_4 = (a_1, a_2, a_5, a_3, a_6)$.

Accordingly, the set $\Theta = \{\theta_1, \theta_2, \theta_3, \theta_4\}$ of sequences of data operations is:

- $\theta_1 = (I(d_1), R(d_1), R(d_2), U(d_1), U(d_2), I(d_3), D(d_2), I(d_1))$;
- $\theta_2 = (I(d_1), R(d_1), R(d_2), U(d_1), D(d_2), U(d_2), I(d_3), I(d_1))$;
- $\theta_3 = (I(d_1), R(d_1), R(d_2), U(d_1), U(d_2), I(d_3), D(d_1), I(d_1))$;
- $\theta_4 = (I(d_1), R(d_1), R(d_2), U(d_1), D(d_1), U(d_2), I(d_3), I(d_1))$.

Let us assume that the initial data instance $\Delta_s = \{d_0\}$ and consider having the following sets of objects. d_1 is a singleton, d_2 is a set of objects with $|d_2| > 1$

and $d_1 \subseteq d_2$, and d_3 is a set of objects with $|d_3| > 1$, where $d_2 \cap d_3 = \emptyset$, and $d_1 \not\subseteq d_3$. Finally, $d_2 \subseteq \Delta_s$ and $d_3 \not\subseteq \Delta_s$.

For the process of Fig. 8.8, Algorithm 2 returns the following $\Gamma = \{A_1 = (\text{I1: } D(d_2) \rightarrow R(d_2)), A_2 = (\{\text{I1: } D(d_2) \rightarrow U(d_2)\}, \{\text{I1: } D(d_2) \rightarrow R(d_2)\}), A_3 = \emptyset, A_4 = \emptyset\}$, meaning that traces π_1 and π_2 lead to inconsistencies due to missing data, whereas traces π_3 and π_4 do not lead to any inconsistency.

8.4 Conclusion

Process and data are often modeled by designers having different expertise, and, thus, they are conceived separately. Modeling inter-dependencies between processes and data is crucial to prevent the occurrence of undesired situations during process execution, caused by bad design of the interactions between process models and data schemata.

In this chapter, we proposed an approach to discover inconsistencies that may arise when activities operate on shared data. Specifically, we exploited the information enclosed in Simplified Activity Views at two levels of abstractions, starting from considering accessed data classes and, then, moving down to evaluate operations on sets of objects. We identified common design mistakes caused by missing data or attempted creation of existing data.

For future work, we intend to consider a wider set of data inconsistencies, to improve their detection, and to find strategies to resolve them. In this chapter we did not deal with the problem of repairing detected inconsistencies, as we feel that finding a satisfactory solution would require a deep understanding of the dependencies between different operations of a data trace.

For example, let us consider an activity a that reads some missing data d . Missing data can occur either if d have been deleted and never re-inserted before being read, or if d are read without being priorly inserted in the database. However, to repair this inconsistency, different actions may be taken: (i) the insertion of d may be added to the process in-between the deletion and the read of d or d may be inserted in the initial database instance; (ii) the deletion of d may be removed; (iii) the read of d may be removed. Of course, knowledge about the process is needed to decide which repair action should be enacted. In addition, it is worth noticing that each repair action will have some repercussion on the process, possibly giving rise to other inconsistencies.

In this setting, designers could be interested in finding a repair strategy that solves data inconsistencies with the fewest repair operations.

Experimental Evaluation of the Activity View

Some of the results presented in this chapter have been published in [57].

An experiment is a formal, rigorous and controlled investigation [194].

Experimentation provides a systematic, disciplined, quantifiable and controlled way of evaluating human-based activities. Especially when dealing with conceptual models, the way model quality is perceived is strongly related with the satisfaction of the end-users.

In this chapter, we describe in detail the empirical evaluation of the Activity View and discuss the obtained results. For the experiments, we considered the complete definition of Activity View, introduced in Chapter 7. Our main aim is to acquire a tangible outcome of its usability in practice and to provide insights of what should be improved in our proposal.

Sect. 9.1 describes experiment planning and design, while Sect. 9.2 shows the obtained results, reports subjects perception of the proposed conceptual model, and discusses possible improvements.

9.1 Experiment Planning and Design

In order to analyze the usability of the Activity View, we conducted a human-oriented single factor controlled experiment following the design principles explained in [194] and took inspiration from case studies conducted in the field of information systems [195, 196, 197, 198]. In addition, we also administered a modeling exercise and a questionnaire, to gather information about the subjects' perception of the Activity View.

Overall, the empirical evaluation of the Activity View was organized in three distinguished phases: PHASE I aimed to introduce the Activity View to the subjects, PHASE II focused on the controlled experiment, and PHASE III aimed to evaluate subjects perception of the Activity View through a questionnaire.

The chosen factor is the *Activity View*, which represents our controlled variable, with factor levels *present* and *absent*. Specifically, we evaluated how the

using Activity Views can improve both the modeling and comprehension of the interplay between a process model and a related database schema.

In order to analyze such improvement quantitatively and qualitatively, we formulated the following hypothesis.

H1– The Activity View improves the comprehension of integrated data and processes: The Activity View improves the comprehension of which data of a conceptual database schema are needed by a process activity to be executed. Improved comprehension task performance is quantified in terms of reduced execution time and error rate.

In addition, we involved subjects in a modeling exercise followed by a questionnaire to assess whether the Activity View is perceived as easy to read, understand, use, write and adaptable to different application domains.

Subjects are 21 students enrolled in the M.Sc. degree in Computer Science Engineering, 8 students enrolled in the M.Sc. degree in Medical Bioinformatics, and 4 researchers in the field of database design. All of the 33 subjects attended at least one information system course where BPMN is explained, and at least one complete database course. Among these, 8 subjects have working experience in the field of UML-based database design, whereas none of them has worked with BPMN at a professional level.

The details of the experiment are summarized in Table 9.1.

OBJECTIVE	Evaluate whether the Activity View improves comprehension of the interplay between a process model and a database schema.
INDEPENDENT VAR.	Activity View (present or absent).
DEPENDENT VAR.	Time needed to execute the exercises, correctness of the answers.
SUBJECTS	Trained students enrolled in the M.Sc. in computer science engineering and in the M.Sc. in medical bioinformatics.
CONTEXT	Process and data modeling: understanding insights brought by the use of the Activity View.

Table 9.1: Setting of the performed controlled experiment.

The proposed three-phased empirical evaluation is organized as shown in Fig. 9.1. During PHASE I the subjects attended a tutorial on the Activity View, where fundamental concepts and motivations were explained.

In detail, PHASE I consisted of a 30 minutes tutorial addressing the problem of connecting business process models and database schemata at a conceptual level. The tutorial was based on recent literature [19, 45, 56, 74, 83] and discussed the motivations behind two main research questions.

The first one “How is the connection between persistent data used by business processes and databases realized at the conceptual level?” addresses the importance of choosing an abstraction level that is suitable to sit between existing process and conceptual database models. The second one “How does the process interact with the database?” discusses which are common kinds of operations performed by a process on persistent data and how they can be represented.

Then, the tutorial recalled which BPMN elements (such as data objects, process variables, message flows, and events that carry data) may be used to represent data within a process and, precisely, we stressed on the concept of *persistent data* and on the use of BPMN data stores in practice.

Once having addressed the research problem, we introduced the Activity View as a possible solution for bridging the gap between processes and data and explained its formalization (cf. Def. 7.1) step-by-step. Then, we discussed all the insights detailed in Sect. 7.1.3 brought by the Activity View, by also referring to practical examples. Last but not least, we explained the structure of the experimental evaluation, clarified subjects doubts, and randomly divided all of the participants in two groups, GROUP 1 and GROUP 2.

PHASE II consisted of the actual controlled experiment, aimed at evaluating hypothesis *H1*. Subjects were asked to execute the same comprehension task on paper and their performance was measured by comparing observations for the same subject.

In detail, PHASE II was divided into two runs (RUN 1 and RUN 2 of Fig. 9.1) for an overall duration of 60 minutes. Each run consisted of a questionnaire containing 7 questions regarding the conceptual insights brought by the Activity View (cf. Sect. 7.1.3). We used a within-groups approach, that is, we randomly divided the number of subjects into two groups, and each group performed the task with and without the Activity View. In detail, we provided all the subjects

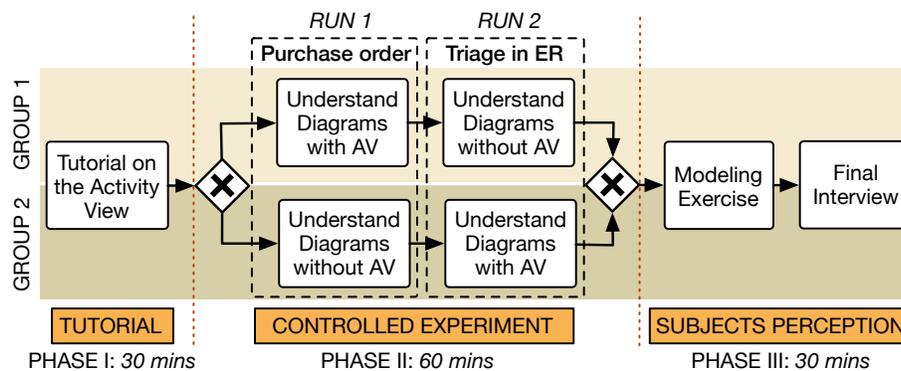


Fig. 9.1: Main phases of the empirical evaluation of the Activity View.

with a textual description of a process and its related data operations, and with the corresponding BPMN and UML diagrams.

At each run, one group was also provided with the tabular representation of the Activity Views related the process and data diagrams. During the first run, GROUP 1 was asked to execute the experimental task using also the provided Activity Views while GROUP 2 was asked to execute the same task but relying only the textual description of the context and on the BPMN process model and UML database schema. During the second run we switched groups, i.e., GROUP 1 executed the task without Activity Views, whereas GROUP 2 used the Activity Views, and changed the application domain in order to avoid potential learning. The chosen domains were: Purchase order on a web-pharmacy for RUN 1 and triage in Emergency Room for RUN 2.

Given that our experiment was designed in order to have observations on the same subjects with and without the Activity View, we relied on paired analysis [199].

Our main goal was to evaluate whether and how much (i) subjects provided with the Activity View were faster in answering the questions, (ii) the accuracy of the answers improved with the help of the Activity View. Both aspects are strongly related to the comprehension of the proposed model, as the Activity View must be well-understood in order to be used quickly and correctly by the subjects. During the whole PHASE II, the formal description of the Activity View was written on the blackboard so that participants could consult it.

The text of the designed exercises is detailed in the following pages. We also report the provided Activity Views and the questionnaires, where correct answers are highlighted in blue.

Exercise 1 Text - Purchase Order on a Web Pharmacy

The process describes a purchase order on a web-pharmacy.

A new process instance is created when an order request is received from a customer. The order request contains information about the customer, the order and the ordered items. Since all customers need to be registered in the database, the information about the customer contained in the order request is compared with the data of customers already saved in the database. If the customer is new, he or she must be added to the database of the pharmacy. Then, the order is added to the database. An order contains a preamble of general information, such as its number, priority, and shipping costs, but it also contains the list of purchased drugs and the related quantity. Afterwards the ordered drugs are obtained from the warehouse and are boxed for shipment. While obtaining the drugs and boxing them, an invoice is created and, then, it is sent to the customer. When invoice creation begins, the order number and the number of the purchasing customer must be checked and, then, the new invoice is created

and added to the database. Then, the process waits for the payment to be received. The operator that receives the proof of payment must record it in the database, prior to updating the status of the order to “Ready for Shipment”. Then, the parcel is shipped to the customer. The process ends when the order is fulfilled.

The BPMN diagram and the UML class diagram corresponding to the description above are provided below.

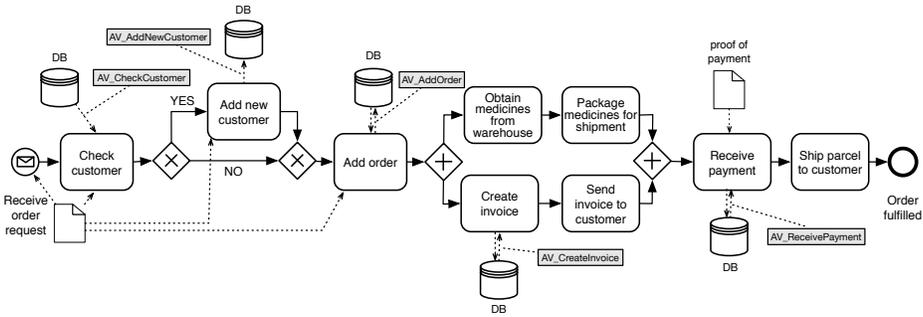


Fig. 9.2: The process model of Exercise 1.

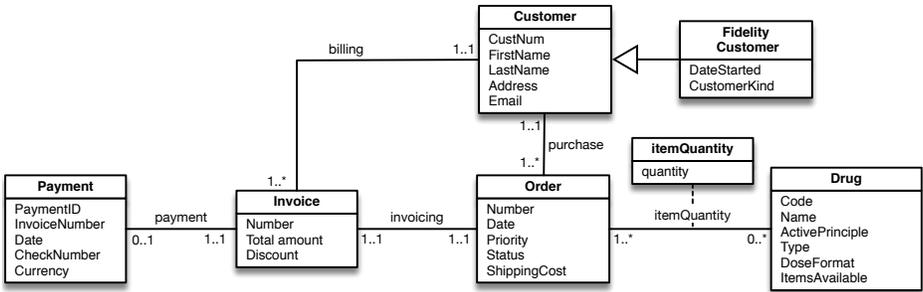


Fig. 9.3: The database schema of Exercise 1.

Exercise 1 - Questionnaire

Correct answers have been reported in between parentheses in blue, for completeness purposes.

Preliminary questions

- Have you ever had any working/internship experience in the field of BPMN modeling? _____

– Have you ever had any working/internship experience in the field of UML data modeling? _____

Exercise 1 - Purchase order on a web-pharmacy. Questions.

1. Which data classes does activity “Check customer” access?
_____ (Customer)
2. Which data classes does activity “Receive Payment” access?
_____ (Payment, Order)
3. (a) Do the sets of classes accessed by activities “Add order” and “Create invoice” intersect? YES NO (YES)
If they do, which data classes belong to their intersection?
_____ (Customer, Order)
4. Are there classes in the data schema that are never accessed by activities of the process? If so, which ones?
_____ (Fidelity Customer)
5. Which are the classes of the process used by the highest number of activities?
_____ (Customer)
6. Which are the activities of the process that access class “Order”?
_____ (Add order, Receive Payment, Create Invoice)
7. Are there classes that are used only for read operations? If so, which ones?
_____ (NO)

AV_CheckCustomer					
TUPLE	CLASS SET	ASSOC SET	ACCESS TYPE	ACCESS TIME	INSTANCES
T1	{Customer(CustNum, FirstName, Lastname)}	∅	R	During	(1,*)

AV_AddNewCustomer					
TUPLE	CLASS SET	ASSOC SET	ACCESS TYPE	ACCESS TIME	INSTANCES
T1	{Customer(*)}	∅	I	During	(1,1)

AV_AddOrder					
TUPLE	CLASS SET	ASSOC SET	ACCESS TYPE	ACCESS TIME	INSTANCES
T1	{Customer(Number)}	∅	R	Start	(1,1)
T2	{Order(*), itemQuantity(quantity), Drug(Code)}	itemQuantity	I	During	(1,*)

AV_CreateInvoice					
TUPLE	CLASS SET	ASSOC SET	ACCESS TYPE	ACCESS TIME	INSTANCES
T1	{Customer(Number), Order(Number)}	purchase	R	Start	(1,1)
T2	Invoice(*)	∅	I	During	(1,1)

AV_ReceivePayment					
TUPLE	CLASS SET	ASSOC SET	ACCESS TYPE	ACCESS TIME	INSTANCES
T2	Payment(*)	∅	I	During	(1,1)
T3	Order(Status)	∅	U	End	(1,1)

Fig. 9.4: Exercise 1: The Activity Views provided to the participants of Group 1.

Exercise 2 Text - Triage in Emergency Room

The process describes the main steps performed by a triage nurse to prioritise patients coming to a hospital emergency room (ER).

A new process instance is created when a patient arrives in ER. Since we assume that all the patients are already registered in the database, when creating a new admission, the nurse must compare the information regarding the arrived patient with the database. Then, a new admission is created for that patient and stored in the database. Then, the nurse interviews the patient quickly and assesses his or her respiration through a breathing test, whose parameters are saved in the database. If the breathing rate is abnormal, a red tag must be assigned to the patient. Instead, if the patient does not present respiratory abnormalities, his or her cog-

nitive status is checked and, depending on the degree of consciousness, either a yellow or a green tag is assigned. Information about assigned tags is recorded during assessment completion, during which also the collected information regarding the overall assessment of the patient is updated. Finally, a physician is called in to make a diagnosis, based on data regarding the current admission, on previous therapies, if any, and on the results of the respiration test. The diagnosis is saved in the database together with its validity, which holds starting from the moment it the diagnosis recorded.

The BPMN diagram and the UML class diagram corresponding to the description above are provided below, together with the Activity Views given to Group 2.

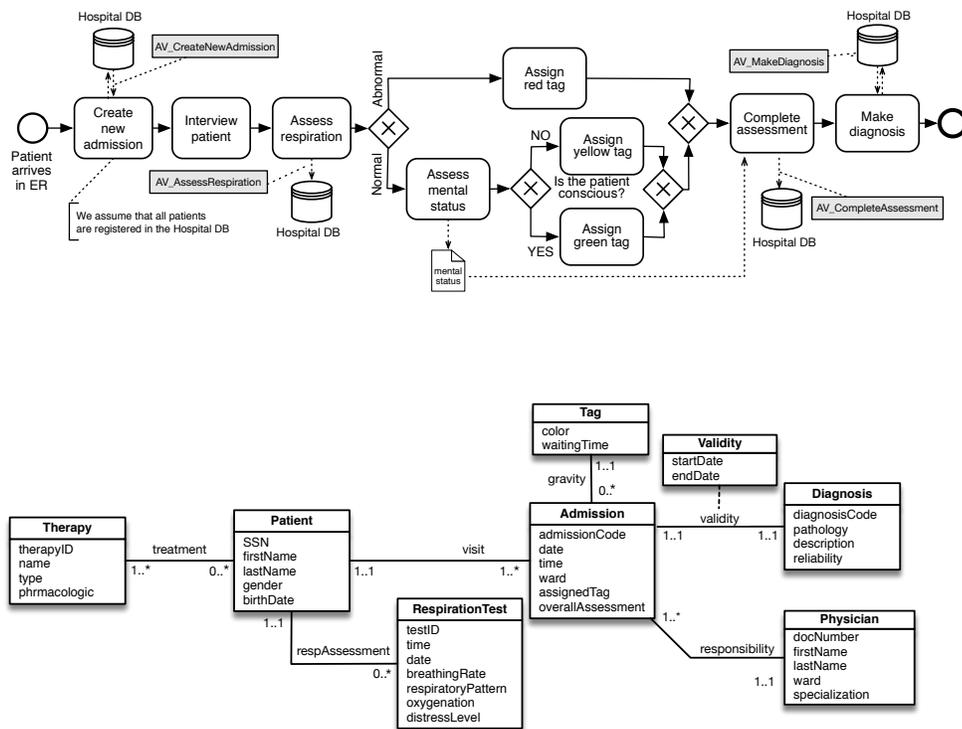


Fig. 9.5: The process model and database schema of Exercise 2.

Exercise 2 - Questionnaire

Correct answers have been reported in between parentheses in blue, for completeness purposes.

Exercise 2 - Triage in Emergency Room. Questions.

1. Which data classes does activity “Create new admission” access?
 _____ (Patient, Admission)
2. Which data classes does activity “Complete Assessment” access?
 _____ (Admission)
3. (a) Do the sets of classes accessed by activities “Make Diagnosis” and “Create New Admission” intersect? YES NO (YES)
 (b) If they do, which data classes belong to their intersection?
 _____ (Patient, Admission)
4. Are there classes in the data schema that are never accessed by activities of the process? If so, which ones?
 _____ (Physician, Tag)
5. Which are the classes of the process used by the highest number of activities?
 _____ (Admission)
6. Which are the activities of the process that access class “Patient”?
 _____ (Create new admission, Make diagnosis)
7. Are there classes that are used only for read operations? If so, which ones?
 _____ (YES. Patient, Therapy)

AV_CreateNewAdmission					
TUPLE	CLASS SET	ASSOC SET	ACCESS TYPE	ACCESS TIME	CARDINALITY
T1	{Patient(*)}	∅	R	S	(1,1)
T2	{Admission(*)}	∅	I	D	(0,1)

AV_AssessRespiration					
TUPLE	CLASS SET	ASSOC SET	ACCESS TYPE	ACCESS TIME	CARDINALITY
T2	{RespirationTest(*)}	∅	I	E	(1,1)

AV_CompleteAssessment					
TUPLE	CLASS SET	ASSOC SET	ACCESS TYPE	ACCESS TIME	CARDINALITY
T1	{Admission(overallAssessment, assignedTag)}	∅	U	D	(1,1)

AV_MakeDiagnosis					
TUPLE	CLASS SET	ASSOC SET	ACCESS TYPE	ACCESS TIME	CARDINALITY
T1	{Patient(*), Admission(*)}	{visit}	R	S	(1,1)
T2	{Patient(*), Therapy}	{treatment}	R	S	(1,*)
T3	{Patient(*), RespirationTest(*)}	{respAss}	R	S	(1,1)
T4	{Diagnosis(*)}	∅	I	D	(1,1)
T5	{Validity(startDate)}	∅	I	D	(1,1)

Fig. 9.6: Exercise 2: The Activity Views provided to the participants of Group 2.

Finally, PHASE III aimed to give participants the possibility of writing the Activity View during process modeling. This Modeling Exercise (cf. Fig. 9.1) gave us the possibility of asking participants more detailed questions about the use of the Activity View during the Final Interview.

All subjects were asked to model a BPMN process and to write the related Activity Views, given a textual description of the process and the UML class diagram of the referred domain database, shown in Fig. 9.7. We evaluated both the correctness of the designed processes and of the related Activity Views. The text of Exercise 3 is provided below.

Exercise 3 Text - Modeling a student examination process

Design a BPMN process diagram that corresponds to the following description.

Let us consider the process of student examination, from the perspective of a professor. For simplicity, let us assume to have a single student, willing to take an oral exam, and a single examining professor.

The first activity of the process is the definition of the day of the exam. The professor must add to Esse3¹ all the possible dates for the exam, considering the exam name, scheduled day, room, and the possibility of having a lab session.

Then, when the student comes on the exam day, the professor must check if the student has registered. This is done by checking that in Esse3 there is some registration corresponding to the student's immatriculation number for the given exam. (For simplicity, we assume that all students are registered prior to take the exam). Then the professor examines the student.

Finally, when the exam is over, the professor grades the student and registers all the details of the grade in Esse3.

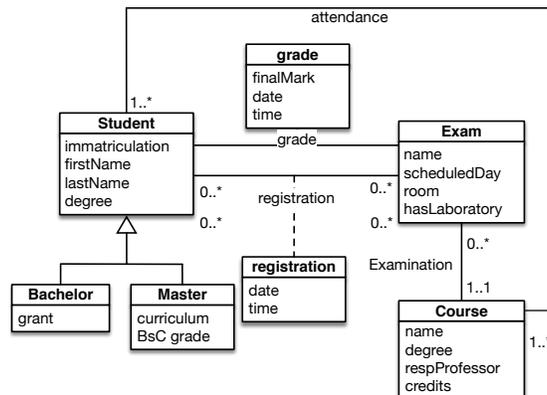


Fig. 9.7: Database schema representing the described domain of student examinations.

As a final step of PHASE III, we conducted a questionnaire-based interview to measure subjects perception the Activity View, considering both process modeling and the discovery of conceptual insights.

¹ Esse3 is a web-based system supporting the management of student careers in several academic institutions in Italy.

9.2 Results, Analysis, and Discussion

Overall, the obtained results confirmed our hypothesis *H1* that the Activity View improves the integrated design and understanding of processes and related data, both in terms of reduced comprehension task execution times and increased answer correctness.

The comprehension task executed during PHASE II allowed us to quantitatively evaluate the use of the Activity View for connected process and data model analysis. For each subject, we measured the task execution time and counted how many of the questions were answered correctly. At first, we applied the most restrictive requirements for correctness, that is, answers were considered correct only if they were both right and complete. In other words, if one answer was only partially correct or incomplete, it was considered as a wrong answer.

The detailed raw results are shown in Fig. 9.8 and are summarized by the histograms shown in Fig. 9.9.

In the first run, subjects provided with the Activity View took an average of 12,45 minutes and 84,03% of the answers was evaluated correct. Instead, the group without Activity View took an average of 21,57 minutes to complete the task, but only 39,29% of the answers was correct.

Results related to the second run showed a reduction in answering times for both groups, especially for the those not having the Activity View, as shown in Fig. 9.9. Accordingly, subjects claimed that they learned what the questions were asking for. However, the correctness of the answers also decreased.

By combining the results of both runs, we see that by using the Activity View task times decrease by 37,68% on average, while the number of correct answers improves by 44,15%.

To evaluate the statistic significance of the obtained results, we applied the paired t-test to both execution times and answer correctness by matching the results of one subject without the Activity View to those of the same subject with the Activity View.

The paired t-test is a statistical parametric test that is typically employed to compare two measurements on the same subject [200].

Paired t-test: preconditions. Being a parametric test, the paired t-test requires that data meet certain preconditions [199]: (i) the samples must be paired and the matching must be planned prior to collecting data; (ii) the observations must be independent from one another; (iii) the pairs must be representative of a larger population; (iv) the difference scores are normally distributed.

Preconditions (i) and (ii) can be directly assumed from the design of our experiment.

(iii) Given that our subjects were selected among students/database experts having different background but approximately the same level of BPM knowledge, we say that our data could represent a larger population of people having such characteristics.

EXERCISE 1 - Purchase Order on a Web Pharmacy																	
GROUP 1 - WITH ACTIVITY VIEW							TIME	ANSWERS	GROUP 2 - WITHOUT ACTIVITY VIEW							TIME	ANSWERS
Q1	Q2	Q3	Q4	Q5	Q6	Q7			Q1	Q2	Q3	Q4	Q5	Q6	Q7		
Y	Y	Y	Y	Y	Y	N	13	6	Y	N	Y	N	Y	Y	N	23	4
Y	Y	Y	N	Y	Y	Y	14	6	Y	N	Y	N	Y	Y	N	24	4
Y	Y	Y	Y	Y	Y	Y	9,84	7	N	N	Y	N	N	Y	N	24	2
Y	Y	Y	Y	Y	Y	Y	9,87	7	Y	N	Y	Y	N	N	N	25	3
Y	Y	Y	N	Y	Y	Y	11	6	Y	N	N	Y	Y	N	N	25	3
Y	Y	Y	Y	Y	Y	N	13	6	Y	N	N	Y	Y	Y	N	24	4
Y	Y	Y	N	Y	N	Y	15	5	Y	N	N	Y	N	N	N	21	2
Y	Y	N	N	Y	N	Y	14	4	Y	N	N	Y	Y	N	N	24	3
Y	Y	N	N	Y	N	Y	16	4	Y	N	N	Y	Y	N	N	20	3
Y	Y	Y	N	Y	Y	Y	11	6	Y	N	N	N	Y	N	N	25	2
Y	Y	Y	Y	Y	Y	Y	11	7	N	N	N	N	Y	N	N	27	1
Y	Y	Y	N	N	N	Y	14	4	N	N	N	N	Y	N	Y	21	2
Y	Y	N	N	Y	Y	Y	13	5	Y	N	N	N	Y	N	N	22,16	2
Y	Y	Y	Y	Y	Y	Y	12	7	Y	Y	Y	N	Y	N	N	13	4
Y	Y	Y	Y	Y	Y	Y	12	7	Y	N	Y	Y	Y	Y	N	14	5
Y	Y	Y	Y	Y	Y	Y	13	7	N	N	N	N	N	N	N	13	0
Y	Y	Y	Y	Y	Y	N	10	6	N	N	N	N	N	N	N		
AVERAGE TIME AND CORRECTNESS							12,45	5,88	AVERAGE TIME AND CORRECTNESS							21,57	2,75
TOTAL % CORRECT ANSWERS							84,03		TOTAL % CORRECT ANSWERS							39,29	
STANDARD DEVIATION							1,83	1,11	STANDARD DEVIATION							4,46	1,29

EXERCISE 2 - Triage in Emergency Room																	
GROUP 1 - WITHOUT ACTIVITY VIEW							TIME	ANSWERS	GROUP 2 - WITH ACTIVITY VIEW							TIME	ANSWERS
Q1	Q2	Q3	Q4	Q5	Q6	Q7			Q1	Q2	Q3	Q4	Q5	Q6	Q7		
N	N	N	N	Y	N	N	9	1	Y	Y	Y	Y	Y	Y	N	8,5	6
Y	N	Y	N	Y	N	N	13	3	Y	Y	Y	Y	Y	N	N	9	5
N	N	N	N	Y	N	N	10	1	Y	Y	N	Y	Y	Y	N	8,6	5
Y	N	Y	N	N	N	N	12	2	N	N	Y	N	Y	N	N	5	2
N	N	N	N	N	N	N	14	0	Y	Y	Y	Y	Y	Y	N	12	6
Y	N	N	N	N	N	Y	14	2	Y	Y	Y	Y	Y	Y	N	13	6
N	N	N	N	N	N	N	14	0	Y	N	Y	N	Y	Y	Y	10	5
N	N	N	Y	N	N	N	15	1	Y	N	Y	N	Y	Y	N	11	4
N	N	Y	N	N	Y	N	15	2	Y	N	Y	Y	N	N	N	14	3
N	N	N	N	N	N	Y	15	1	Y	Y	Y	N	Y	Y	N	7	5
Y	N	Y	N	Y	Y	Y	18	5	Y	Y	N	N	Y	N	Y	13	4
Y	N	Y	N	Y	N	N	22	3	Y	Y	Y	Y	Y	Y	N	13	6
Y	N	N	N	Y	N	N	19	2	Y	Y	Y	Y	Y	Y	Y	11,92	7
N	N	N	N	Y	N	Y	19	2	Y	Y	Y	Y	Y	Y	N	8	6
Y	Y	Y	N	Y	N	N	17,45	4	Y	Y	Y	N	Y	Y	Y	9	6
Y	N	Y	N	Y	N	N	18	3	Y	Y	Y	N	Y	N	N	11	4
Y	N	N	N	Y	N	N	13	2									
AVERAGE TIME AND CORRECTNESS							15,14	2,00	AVERAGE TIME AND CORRECTNESS							10,25	5,00
TOTAL % CORRECT ANSWERS							28,57		TOTAL % CORRECT ANSWERS							71,43	
STANDARD DEVIATION							3,41	1,32	STANDARD DEVIATION							2,53	1,32

Fig. 9.8: Results of Exercise 1 and Exercise 2 of PHASE II corrected adopting the most restrictive requirements for correctness.

(iv) Since normality was hard to assess graphically on such a small sample, we used the Shapiro-Wilk normality test [201] to check whether the differences between observations were normally distributed.

For brevity, let us all D_t the set of differences between paired observations of comprehension task execution times with and without the Activity View and D_c the set of paired observations of comprehension task correctness with and without the Activity View. For both D_t and D_c , we were able to assess approximate normality, considering a significance level $\alpha = 0.01$. The W statistics for D_t was $W = 0.914$, whereas for D_c we had $W = 0.962$. For both cases, the 99% critical value accepted range was $[0.9104 : 1.000]$. However, the p-value obtained for execution times is greater than α , when the significance level is set to $\alpha = 0.05$.

Therefore, since the normality hypothesis for execution times is weak, we began with applying the t-test, but followed up with the help of other paired analysis methods.

The null hypothesis for the paired t-test states that “the performance of the comprehension task, quantified in terms of exercise execution time and correctness, is the same with and without the Activity View”. The previously formulated hypothesis $H1$ rejects the null hypothesis. We obtained a p-value < 0.001 and thus, the results are statistically significant. The details of the calculation of p are summarized in the following paragraph.

Paired t-test applied to Execution Times. The mean of differences of the measurements with the Activity View and those without the Activity View is equal to -6.8449. The 95% critical value accepted interval of the mean of differences is [-2.1200 : 2.1200]. The intermediate values used in the calculations of the p-value are: $t = -6.5804$, degrees of freedom = 32, and standard error of difference = 1.040. Thus, the obtained two-tailed p-value is less than 0.001.

Paired t-test applied to Answer Correctness. The mean of differences of the measurements with the Activity View and those without the Activity View is equal to 3.091. The 95% critical value accepted interval of the mean of differences is [-0.6300 : 0.6300]. The intermediate values used in the calculations of the p-value are: $t = 10.1631$, degrees of freedom = 32, and standard error of difference = 0.304. Thus, the obtained two-tailed p-value is less than 0.001.

Another approach to paired analysis is the Wilcoxon signed rank sum test [199], a non-parametric hypothesis test used to compare two paired groups assuming the following preconditions on data: (i) the pairs are representative of a larger population; (ii) the samples are paired and the matching is decided

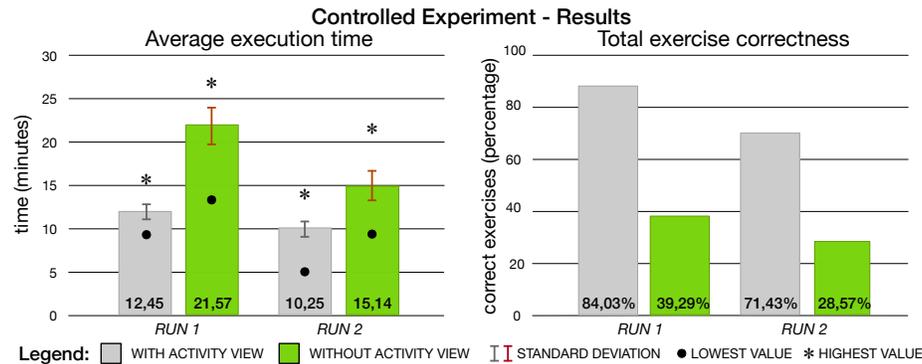


Fig. 9.9: Average execution time with standard deviation (left) and total percentage of correct answers (right) for the two runs of PHASE II.

EXERCISE 1 - Purchase order on a web pharmacy																			
GROUP 1 - WITH ACTIVITY VIEW							TIME	ANSWERS	GROUP 2 - WITHOUT ACTIVITY VIEW							TIME	ANSWERS		
Q1	Q2	Q3	Q4	Q5	Q6	Q7			Q1	Q2	Q3	Q4	Q5	Q6	Q7				
1	1	1	1	1	1	0	13	85,71	1	0,5	1	0	1	1	0	23	64,29		
1	1	1	0	1	1	1	14	85,71	1	0	1	0	1	1	0	24	57,14		
1	1	1	1	1	1	1	9,84	100,00	0,5	0	1	0	0,5	1	0	24	42,86		
1	1	1	1	1	1	1	9,87	100,00	1	0,5	1	1	0	0,66	0	25	59,43		
1	1	1	0,5	1	1	1	11	92,86	1	0,5	0,25	1	1	0,33	0	25	58,29		
1	1	1	1	1	1	0	13	85,71	1	0,5	0,5	1	1	1	0	24	71,43		
1	1	1	0	1	1	1	11	85,71	1	0,5	0	1	0	0,66	0	21	45,14		
1	1	1	0	1	1	1	15	85,71	1	0,5	0	1	1	0,66	0	24	59,43		
1	1	0	0	1	0,66	1	14	66,57	1	0,5	0,5	1	1	0,66	0	20	66,57		
1	1	0,5	0	1	0,33	1	16	69,00	1	1	1	0	1	0,33	0	25	61,86		
1	1	1	1	1	1	1	13	100,00	1	0,5	1	1	1	1	0,5	27	85,71		
1	1	1	1	1	1	1	12	100,00	0,5	0	0	0	0,5	0	0	21	14,29		
1	1	1	1	1	1	0,5	10	92,86	0	0,5	0	0	1	0,66	1	22,16	45,14		
1	1	1	1	1	1	1	11	100,00	1	0,5	0,75	0	1	0,66	0	13	55,86		
1	1	1	0	0	0,66	1	14	66,57	1	0,5	0,5	0	1	0,33	0	14	47,57		
1	1	0,75	0	1	1	1	13	82,14	0,5	0,5	0,5	0	1	0	0	13	35,71		
1	1	1	1	1	1	1	12	100,00											
							AVERAGE TIME	12,45								AVERAGE TIME	21,57		
							TOTAL % CORRECT ANSWERS	88,15								TOTAL % CORRECT ANSWERS	54,42		
							STANDARD DEVIATION	1,83	11,86								STANDARD DEVIATION	4,46	16,24

EXERCISE 2 - Triage in Emergency Room																			
GROUP 1 - WITHOUT ACTIVITY VIEW							TIME	ANSWERS	GROUP 2 - WITH ACTIVITY VIEW							TIME	ANSWERS		
Q1	Q2	Q3	Q4	Q5	Q6	Q7			Q1	Q2	Q3	Q4	Q5	Q6	Q7				
0,5	0,5	0	0	1	0,5	0,5	9	42,86	1	1	1	1	1	1	0,5	8,5	92,86		
1	0	1	0	1	0,5	0	13	50,00	1	1	1	1	1	0,5	0,5	9	85,71		
0,5	0	0,25	0	1	0	0,5	10	32,14	1	1	0	1	1	1	0,5	8,6	78,57		
1	0	1	0	0	0,5	0	12	35,71	0,5	0	1	0	1	0,5	0	5	42,86		
0,5	0	0,75	0,5	0,5	0,5	0	14	39,29	1	1	1	1	1	1	0,5	12	92,86		
1	0	0,25	0	0,5	0,5	1	14	46,43	1	1	1	1	1	1	0,5	13	92,86		
0,5	0	0	0,5	0	0	0,5	14	21,43	1	0,5	1	0,5	1	1	1	10	85,71		
0,5	0,5	0,75	1	0,5	0,5	0,5	15	60,71	1	0,5	1	0,5	1	1	0,5	11	78,57		
0,5	0	1	0	0	1	0,5	15	42,86	1	0,5	1	1	1	0,5	0,5	14	64,29		
0,5	0	0	0,5	0	0	1	15	28,57	1	1	1	0,5	1	1	0,5	7	85,71		
1	0,5	1	0,5	1	1	1	18	85,71	1	1	0,5	0,5	1	0,5	1	13	78,57		
1	0,5	1	0,5	1	0	0,5	22	64,29	1	1	1	1	1	1	0,5	13	92,86		
1	0	0,5	0	1	0,5	0,5	19	50,00	1	1	1	1	1	1	1	11,92	100,00		
0,5	0,5	0,75	0,5	1	0	1	19	60,71	1	1	1	1	1	1	0,5	8	92,86		
1	1	1	0,5	1	0,5	0,5	17,45	78,57	1	1	1	0	1	1	1	9	85,71		
1	0,5	1	0	1	0,5	0	18	57,14	1	1	1	0,5	1	0,5	0,5	11	78,57		
1	0	0	0	1	0,5	0	13	35,71											
							AVERAGE TIME	15,14								AVERAGE TIME	10,31		
							TOTAL % CORRECT ANSWERS	48,95								TOTAL % CORRECT ANSWERS	83,04		
							STANDARD DEVIATION	3,41	17,30								STANDARD DEVIATION	2,53	13,77

Fig. 9.10: Results of Exercise 1 and Exercise 2 of PHASE II corrected by giving scoring also to partially correct answers.

before data collection; (iii) each pair is selected in an independent way.

Wilcoxon signed rank sum test. All preconditions (i)–(iii) may be assumed from the way our experiment was designed. The null hypothesis considered for the Wilcoxon signed rank sum test is the one considered for the paired t-test, which is rejected by $H1$. We applied the Wilcoxon signed rank sum test to execution times and obtained $W = -514$ and $Z = -4.59$. Given that the number of samples is 33, we used normal tables and relied on the statistic Z , which is not in the 95% critical value accepted range of $[-1.9600 : 1.9600]$. Given that the obtained p-value is 0.000004332, we reject the null hypothesis and confirm $H1$.

For completeness, we applied the Wilcoxon signed rank sum test also for exercise correctness and obtained $W = 493.5$ and $Z = -4.83$. Again, the p-value

is extremely small ($p = 000001359$) and, thus, we may accept $H1$.

Subject	With AV	Without AV	Difference	Abs Difference	Rank	Signed Rank
1	9,84	10	-0,16	0,16	1	-1
2	14	13	1	1	4	4
3	13	14	-1	1	4	-4
4	15	14	1	1	4	4
5	14	15	-1	1	4	-4
6	16	15	1	1	4	4
7	11	13	-2	2	7	-7
8	9,87	12	-2,13	2,13	8	-8
9	11	14	-3	3	9,5	-9,5
10	10	13	-3	3	9,5	-9,5
11	13	9	4	4	11,5	11,5
12	11	15	-4	4	11,5	-11,5
13	13	18	-5	5	14	-14
14	8	13	-5	5	14	-14
15	9	14	-5	5	14	-14
16	12	17,45	-5,45	5,45	16	-16
17	13	19	-6	6	17,5	-17,5
18	14	20	-6	6	17,5	-17,5
19	11	18	-7	7	19,5	-19,5
20	12	19	-7	7	19,5	-19,5
21	14	22	-8	8	21	-21
22	14	22,16	-8,16	8,16	22	-22
23	13	24	-11	11	23,5	-23,5
24	10	21	-11	11	23,5	-23,5
25	13	25	-12	12	25	-25
26	12	25	-13	13	26,5	-26,5
27	11	24	-13	13	26,5	-26,5
28	7	21	-14	14	28	-28
29	8,5	23	-14,5	14,5	29	-29
30	9	24	-15	15	30	-30
31	11,92	27	-15,08	15,08	31	-31
32	8,6	24	-15,4	15,4	32	-32
33	5	25	-20	20	33	-33

Fig. 9.11: Intermediate values used in the computation of statistics W and Z for the Wilcoxon test applied to execution times. $W^+ = 23,5$, whereas $W^- = 537,5$.

Once having corrected all the exercises, we organized a meeting with all the participating subjects to discuss results. The difference of execution times between *Exercise 1* and *Exercise 2*, especially for those not having the Activity View is probably due to some form of learning: Subjects claimed that they became familiar with looking at process model and data schemata as a whole and they had learned how the task was structured.

Then, we discussed some of the questions that lead to the highest number of mistakes. In particular, we analyzed the results of questions $Q2$ and $Q7$ from *Exercise 1*, and question $Q4$ and $Q7$ of *Exercise 2*.

- Question $Q2$ of *Exercise 1*: Most of the subjects of “Group 2” added class “Customer” to the answer, thinking that also the customer was needed to execute the activity. However, at a conceptual level, the operation involves only classes “Order” and “Payment”, as the customer is known from the beginning of the process and the order already contains information about the customer identifier.

- Question Q7 of *Exercise 1*: Most of the subjects of “Group 2” wrote that class “Drug” is only read by the process when, in reality, they are read by some operator to be inserted in the database, but there is no activity that reads them from the database.
- Question Q4 of *Exercise 2*: Most of the subjects of “Group 1” forgot to report class “Tag” among those not used by the process and only put “Physician”. Probably the fact the some tasks concern tag assignment was misleading. However, from the provided data schema it is clear that class “Tag” serves as data dictionary.
- Question Q7 of *Exercise 2*: Most of the subjects of both groups forgot to report class “Therapy” beside “Patient”.

The difference in measured execution times and answer correctness remains significant even when the correction requirements are relaxed and also partially correct answers are assigned a positive score, as shown in Fig. 9.10.

The experimental task of PHASE III was reviewed by assigning one point for each correctly written Activity View tuple, thus considering each attribute of the tuple worth 0.20 points. Besides, we also considered the correctness of the BPMN process. Overall, the 58,89% of the written Activity Views was correct and the 83,94% of the BPMN process diagrams was designed properly.

Common mistakes stemmed from the use of a different access times, which were probably hard to understand from the text of the exercise. Indeed, the percentage of correct Activity Views increases to 61,11% when excluding mistakes related to the access time, which was not easily determinable by reading the exercise. Moreover, several participants included more classes and associations than needed.

The results of this last exercise were in line with the outcome of the final interview conducted at the end of PHASE III, during which we asked subjects to describe their overall perception of the proposed model and to suggest possible improvements.

All the participants declared that executing the first experimental task without the help of the Activity View was more difficult, and the 93% of them answered positively when asked whether the Activity View improves the modeling of the link between processes and related data.

Then, we asked questions about to the use of the Activity View based on a rating scale from 1 to 5, with 1 meaning “strongly disagree” and 5 denoting “strongly agree”. The average results of this questionnaire-based final interview are reported in Fig. 9.12. Overall the Activity View was perceived as more than satisfactory, despite writing it results harder than the other comprehension tasks.

Last but not least, we asked for suggestions related to the graphical visualization of the Activity View, as our goal is to provide a compact representation of our proposal that could be easily blended within existing process modeling tools. Somebody suggested to encode data operations through graphical symbols, to shorten their description. As for other comments, a couple of people pointed

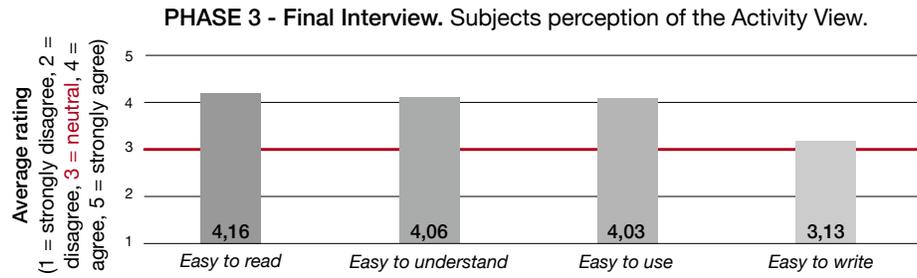


Fig. 9.12: Average rating of subjects perception of the Activity View, retrieved from questionnaire-based interviews.

out that a minimum of experience in database design is required to be able to understand the Activity View completely, while one person expressed the concern that the Activity View may become more complex for bigger and articulated processes.

Overall, the results of the conducted empirical evaluation were encouraging and provided a good starting point for understanding practical needs related to the joint design of processes and data.

Being most of the subjects computer science students, our evaluation approach is limited with respect to being generalizable to real organizational environments, where processes are more complex and people receive professional training. Indeed, exercises were designed in a didactic way to avoid task misunderstanding. Moreover, results are limited to the kinds of asked questions and rely on the preparation of the students in the fields of process and data modeling.

Related Work

As exhaustively surveyed in [74], multiple approaches have been proposed in the BPM and database theory communities in an effort to link processes and data.

However, there exist at least two main challenges to proper integration that are yet to be overcome.

The first one is the perception that humans have of the relative importance of processes and data within an organization, which strongly depends on their IT expertise. On the one hand, professionals concerned with BPM tend to downplay the importance of data, and view it as subsidiary to the processes that manage them. On the other hand, data management experts consider data as the main driver of the processes, and assume that guaranteeing data quality is sufficient to deal with process improvement. As a result, process and data modeling remain independent from each other, and the interaction between the two is often neglected [45].

The second challenge is related to the conceptual representation of the linkage between processes and data. Indeed, since the need of representing data in (conceptual) process models is not perceived as being of primary importance, tools supporting process execution and monitoring need to provide a proprietary way to integrate data. However this connection is realized programmatically, by defining an internal data model and associating it to the BPMN constructs in the process through suitable business rules written in a programming language. Unfortunately, connecting data and processes during process implementation, leaves the conceptual gap among the process and data worlds open.

The goal of this chapter is to provide an overview of existing relevant approaches by highlighting their strengths and weaknesses, and to compare the approach presented in Chapter 7 and Chapter 8 with selected contributions.

The chapter is structured into two main sections. Sect. 10.1 introduces contributions dealing with data representation in the field of business process management, with particular focus on conceptual modeling approaches. Sect. 10.2 discusses approaches addressing the verification of connected process and data models, and the detection of inconsistencies among the data operations performed by a process.

10.1 Linking Data to Business Processes

Despite being endowed with their own body of mature methods and tools, data and process engineering remain loosely integrated in current information systems, thus leading to a sort of “impedance mismatch” between the process layer and the business logic and data layers [37].

In the field of BPM, several research efforts have recently been carried out to address the shortcomings generated by this divide [74].

Existing approaches may be distinguished into two main categories: those that aim to either increase the suitability of activity-centric processes for modeling data [45, 19, 83, 202] and those that focus on the definition of novel data-centric paradigms [42, 91, 183].

Activity-centric modeling paradigms, and especially those relying on BPMN, are by all means the most used in practice, despite the support of the standard for the data perspective remains limited [22, 70]. Yet, this limitation is often perceived as a design choice and data are combined with processes, at a lower, engineering level [45].

A couple of recent proposals remarked the lack of conceptual modeling frameworks supporting the integration of activity-centric processes and data [45, 83].

Db-nets are proposed in [83] as a three-layered formal model for data-aware process design and verification. (i) A data persistence layer manages persistent data relevant for the application domain, (ii) a data logic layer allows one to extract/update data from a database instance, according to a transactional semantics, while (iii) a control layer based on coloured Petri nets captures the process flow and uses data logic to interact with the database.

This approach goes beyond the conceptual modeling of data needed by a process, as it focuses on the modeling and formal verification [177] of a “connected system”, where an instance of a database is subjected to changes imposed by the control layer.

A standard-based framework that realizes the connection between processes and data is presented in [45]. The authors draw inspiration from the artifact-centric approach and propose a way to formalize the data operations performed by a BPMN process on the underlying information model, represented by a UML class diagram. This is achieved through OCL (Object Constraint Language [203]) expressions on process variables, that are stored within a data class called *Artifact*, designed to collect all the process variables associated with a process instance. First of all, the UML class diagram is encoded as a relational database manageable through SQL. Then, BPMN processes are translated into Petri nets. Finally, activities are specified as OCL operation contracts that are encoded into a set of logic derivation rules that derive the insertions/deletions/updates that should be performed on the SQL database.

In [19] the authors propose a technique for automatically derive SQL queries from annotated BPMN data objects with the aim to capture data requirements for activity execution. Data objects are extended with identifiers, information

about their life-cycle, and fields that express complex correlations with other objects.

A framework based on Constraint Logic Programming for representing business processes and reasoning on their behavior and data properties is introduced in [202]. The authors define logical language and a formal semantics to describe data object manipulation and to explicitly represent the interaction of a process with an underlying database. BPMN is enriched by annotations that define the preconditions and effects of single flow elements in terms of data objects.

Preliminary work tackling the connection of activity-centric clinical processes and data at a conceptual level is presented in [147, 204]. A framework for supporting the seamless design of clinical pathways and related data, based on workflows and TIMEER diagrams, is discussed in [204]. Similarly, in [147] data associations of BPMN processes are annotated to provide clinicians with a data-aware overview of care procedures by displaying critical data for decision support.

An approach based on data-flow matrices for representing input and output data of workflow activities is presented in [205]. A data-flow matrix summarizes all the read and write operations performed on the process data objects.

Ultimately, query languages have been used to reason about data quality and process compliance. In [206], the authors propose an expressive formalism to model processes that manipulate persistent data and to understand whether queries performed by process activities are *stable* or change during execution.

Over the last decade, artifact-centric models have emerged as a means to combine data and process aspects in order to represent the informational entities that are valuable for the organization, together with their evolution within the process [91]. Artifact-centric data models aim at conceptually integrating the process layer and the business logic and data layers [37], i.e., they are data driven but also aware of basic control-flow information.

In [92] the authors introduce the framework lying at the core of the data-centric design methodology. Moving from top to bottom, the three logical levels of the framework are: (i) the business operations model, which provides a logical specification of business process execution, (ii) the conceptual workflow, capturing the procedural aspects of the business operation model, and (iii) the operational workflow system, which relies on executable services for message-based communication and artifacts manipulation.

A breakthrough in the development of artifact-centric process models was the definition of the Guard-Stage-Milestone meta-model for artifact life-cycles [41]. The GSM meta-model supports the declarative specification of interaction between business artifacts and is based on (i) a milestone, that corresponds to a business-relevant operational objective expressed as a condition over the information model, (ii) a stage body, encompassing the activities needed to achieve a milestone, and (iii) a guard that controls when a stage is activated [41].

Besides being used to support the interaction between artifact instances, the GSM meta-model [41], has also covered an important role in the definition of the novel Case Management Model and Notation (CMMN) standard [62].

A recent line of research aiming at combining processes and data and focusing on the formal verification of the connected system is the one dealing with Data-centric dynamic systems (DCDS) [74]. DCDS are systems where both the process controlling the dynamics and the manipulation of data are equally central [40].

However, probably due to the lack of a standard modeling language and of proper IT support, the uptake of artifact-centric process models remains limited [37]. Accordingly, the results of a recent study on user perception of existing data-centric approaches show that the integration between process and data in data-centric process models is not perceived as intuitive by business users, at least without IT support [207].

Motivated by the evidence that process designers are used to traditional activity-centric development tools, the approach presented in [44] aims to recover activity-centric processes from data-aware workflow specifications. The authors consider the sequences of services applied to a business artifact, i.e., different “views” of the artifact system, and show how the regularity of such views determines whether they may be represented as sequences of tasks.

Last but not least, PHILharmonicFlows is a comprehensive framework enabling object-aware process support by fostering the understanding of the various relationships between processes, data, functions, and users [208]. Compared to other artifact-centric approaches [41, 92], PHILharmonicFlows allows handling process and data models separately and the process model is derived directly from the information perspective of the overall software system. In addition, flexible process execution is achieved through a combination of micro processes (i.e., object life cycles) and macro processes (i.e., object interactions) [183].

Since the approaches presented in Chapters 7-9 consider activity-centric process models, in the remainder of this section we focus on selected approaches belonging to this research field, yet including PHILharmonicFlows in our discussion.

Despite being substantially different from each other, both the approaches introduced in [45, 83] focus on closing the gap between control flow and data aspects. In line with [83], the Activity View introduced in Chapter 7 embraces the idea to keep both process and data diagrams untouched, while conceptually interconnecting them.

Our perspective focuses on querying conceptual diagrams to improve design and understand their interactions, rather than formalizing the behavior of the overall “connected system” and automating the data perspective of process activities. As we do not propose an extension of BPMN and we are not interested in automating the data perspective, our approach differs also from the one in [19].

Despite recognizing the need of linking processes and data conceptually, the approaches introduced in [45, 19, 202] address the connection of processes and data at a lower (logical) level, by considering process variables and data instances, and by providing valuable details for formalizing and automating queries on process-related data. Instead, the contributions of Chapter 7 and Chapter 8 provide a higher-level, conceptual view of the connection between processes and

data schemata, without excluding the possibility of mapping our approach to any of [45, 202, 19] when moving down to a lower implementation level.

PHILharmonicFlows [183] is probably the most advanced and complete framework belonging to the field of object-oriented process modeling and, when considering requirements for integrated process and data models, it definitely has some overlapping with the Activity View, especially regarding the data perspective. First of all, PHILharmonicFlows aims to keep the process and the data model separate in order to improve understanding, but considering the interaction between different perspectives. This viewpoint is the same we adopted when devising the Activity View. Moreover, PHILharmonicFlows considers challenges related to data integration, cardinality, and mandatory information, which are also tackled by the Activity View.

However, a fundamental difference between the PHILharmonicFlows and the Activity View remains the level considered for data modeling: on the one hand is logical, considering domain objects, their types, and relationships, on the other hand is conceptual, considering classes and relationships among them. Secondly, PHILharmonicFlows still considers data-driven process execution, thus not allowing for the process model to be handled apart from the data model [208]. Finally, PHILharmonicFlows addresses organizational modeling as a third important perspective, which instead is not considered in this thesis. Probably, PHILharmonicFlows will be a promising reference framework for future research when considering novel process abstraction and data modeling levels for the Activity View.

To conclude, we briefly discuss why we chose to work with BPMN process models rather than using other activity-centric approaches that may be perceived easier to integrate with UML class diagrams, e.g., UML activity diagrams.

BPMN and UML activity diagrams have proven to be comparable in terms of suitability for activity-centric process modeling [70, 209, 210]. The limited support provided by BPMN to the data perspective of business processes can be equated to the one provided by other activity-centric process modeling languages, such as UML activity diagrams [210]. A recent survey conducted among BPM experts [209] highlights that UML activity diagrams and BPMN share similar needs in terms of data-awareness. Therefore, since there is no evidence that connecting UML activity diagrams to UML class diagrams is significantly easier than connecting the latter ones to BPMN process models, we chose to use BPMN as it is the current standard for process modeling.

10.2 Unravelling Data Flaws in Process Models

In Chapter 8 we introduced a simplified version of the Activity View and focused on detecting potential inconsistencies that may occur between data operations performed by different process activities. In particular, we considered inconsistencies depending on the order of activity execution, the kind of access to data, and the accessed data instances.

In general, the formal verification of a system integrating data and business processes must take into account that the presence of data subtly affects the behavior described by the process [83]. However, when combining process and data verification problems become undecidable [74].

The verification of data-aware processes is addressed in [177]. The authors present a formal framework, called RAW-SYS, which combines workflow nets, relational data model, and data-centric dynamic systems for capturing the effects of process activities on data.

Other approaches dealing with the validation and verification of the process data-flow are presented in [42, 185, 211, 212, 213, 214].

In [42] the authors propose a formal model for connecting artifact-centric processes with enterprise relational databases. The proposal focuses on a mapping from a relational data model to a set of business entities representing artifacts of interest. Novel notions of updatability and isolation are formulated for business entities to ensure the consistency between changes on the latter ones and on the database.

In activity-centric process models the data exchanged between tasks are often represented in terms of objects. The notions of compliance and conformance between business process models and data object life-cycles is introduced in [211]. In detail, the authors propose a method to derive process models that are compliant with a given data object life-cycle to ensure consistency within business processes of one organization and to support the correct execution of business processes that span several organizations.

Consistency between data operations in business processes is also addressed in [185, 212, 213, 214].

One of the earliest works dealing with data flow validation is presented in [212]. The authors discuss requirements for data modeling requirements and identify seven corresponding validation issues, such as redundant, lost, mismatched, and insufficient data among others.

In [213] the authors provide a formal analysis approach for detecting data-flow anomalies related to missing data, redundant data, and conflicting data in UML activity diagrams. Data read and write operations performed by process activities on data items are recorded in a *data-flow matrix*. Then, algorithms for detecting and eliminating data-flow errors are proposed.

In [185], the authors introduce a framework for analyzing the process data-flow based on anti-patterns that are formalized in the temporal logic CTL* [215]. The nine identified anti-patterns capture repeated mistakes, such as missing, redundant, and inconsistent data, to allow for their detection and repair.

In [214], the authors discuss anomalies related to the use of data objects as preconditions for either BPMN activities or for gateway routing. The authors formalize in Petri Nets the identified data flaws and propose strategies for their resolution. The data anomalies identified differ from those of [185, 212, 213], as they consider both the process data and control flows, with particular attention to deadlock situations.

Finally, the proposal in [216] extends previous research [185, 213] by considering also optionality of data and alternatives for data when checking data-flow correctness in executable BPMN models. Optionality refers to the fact that access to data can be either optional or mandatory, while data alternatives mean that an activity may read or write data elements out of alternative sets.

Compared to the approaches in [185, 216], in Chapter 8 we introduce a higher level of abstraction to detect shared data access, by evaluating which process activities operate on the same classes of a data schema. Besides, we identify a set of inconsistencies between data access operations performed on particular sets of data instances, and define two rules to capture inconsistencies related to missing data and attempted creation of existing data. Specifically, we do not consider “redundant” or “not destroyed” data to be a problem, as we assume that data created by a process and stored in a database, can be used by other processes or applications. Besides, we do not deal with the “inconsistent data” considered in [185], as we advocate that issues related to concurrent data access operations require elaborating on database transactions.

Among related work, the approach presented in [213] is probably the most similar to ours, especially in the algorithms designed to verify the presence of missing data. Instead, our notion of “creation of existing data” is different from the one of “redundant data” provided in [213].

**The Decision Perspective of
Healthcare Processes**

Reconciling and integrating processes and decisions is of paramount importance, both when it comes to modelling the two concerns consistently, as well as in terms of automated discovery of process-decision models [217].

Decision-making plays a crucial role in BPM, as decisions may be based on running processes and may affect process outcomes. The recent introduction of the DMN standard confirms the increasing interest in documenting, modeling, and analyzing the decision dimension of business processes [50].

Recently, the separation of concerns has attracted attention to improve maintenance, flexibility, scalability, and re-usability of process and decision models [20, 49]. However, integrating decision and process models is of paramount importance, especially for those application domains centred around decision- or knowledge-intensive processes [75].

One of the aims of DMN is to bridge the gap between business process models describing the coordination of decision-making and decision logic models [47]. This is done by introducing decision requirements diagrams (DRDs), that define the decisions made in process tasks and their inter-relationships. However, understanding the interplay between DRDs and BPMN process models and separating concerns is not trivial, especially when dealing with aspects such as data that play different yet crucial roles in both models.

Part IV tackles the joint modeling of processes and decisions with the BPMN and DMN standards considering real-world healthcare processes as complex application scenarios.

In Chapter 11, we begin by proposing an approach to derive (fragments of) decision models from the data perspective of process models, considering a set of patterns representing data used by decision activities in BPMN processes. We show the application of the presented pattern-based approach on a real healthcare process dealing with patients affected by Chronic Obstructive Pulmonary Disease (COPD).

In Chapter 12, we look deeper into the management of COPD patients and take a real healthcare system as a starting point for defining a general methodological framework based on BPMN and DMN to design and simulate care pathways encompassing several decision-making tasks. In this chapter, we also discuss how different process perspectives must be considered when modeling multi-disciplinary healthcare processes and provide connection points with the work presented in previous chapters of this thesis.

Finally, in Chapter 13 we give an overview of related work regarding the integrated modeling of processes and decisions with BPMN and DMN, and discuss how such a modeling approach compares to existing approaches for the formalization of clinical-interpretable guidelines.

Deriving Decision Models from Process Models

*This chapter is based on results published in [60, 61].
I would like to thank my colleague and friend Ekaterina Bazhenova,
who also contributed to this work as part of her doctoral thesis.*

In the field of business process management, decisions are becoming increasingly integrated into business processes [48], as they boost organizational competitiveness and may be analyzed, implemented, and reused in multiple processes for improving business outcomes [218]. However, since both BPM and decision management have existed long without proper integration, process modeling languages have often been misused for designing decisions [219], thus resulting in complex “spaghetti” models [220] that are hard to read and to maintain [20].

To overcome this limitation, the Decision Model and Notation [47], introduced in Sect. 2.2, was developed by the OMG for modeling decisions at different levels of detail. One of the purposes of the DMN standard is to complement BPMN for decision design, thus contributing to create a standard-based framework to support the design of processes and related decisions.

One of the ways to link DMN decision models to BPMN process models is by associating decisions with those process activities within which the decision-making takes place [47]. Such *decision activities* may be linked to a decision model that details the decision requirements and the inner decision logic of the activity.

The combination of BPMN and DMN models allows one to naturally model decision logic separately from process logic, thus achieving a “separation of concerns” [221], which eases process and decision model maintainability [20, 50]. Indeed, if decisions are defined in process structures, any modification of the decision logic would require the process model to be modified accordingly. Thus, separating concerns is worthwhile, especially when process and decision models are maintained and re-engineered by different stakeholders.

In this direction, the need for a separate, yet integrated modeling of decisions and processes has become central in BPM research [50]. Indeed, in organizational

realities processes and decisions are closely interrelated since decisions may drive the process flow and processes may manipulate information that is used to make decisions. In particular, decision activities take in input data created or acquired earlier in the process, and produce decision outputs which may be used later in the process. Whereas separating concerns is easy for newly modeled processes, separating concerns in existing processes becomes quite challenging, especially when decision-making aspects are integrated within process models, and both process and decision models rely on the same, shared business data.

In this chapter, we consider the extraction of DMN decision models from BPMN process models [20, 219, 220], focusing on the data perspective of process models and providing an approach to build a DMN model including such decisions. Previous work tackled the discovery of DMN decision models from the process control flow [20, 219], or from event logs [222, 223]. Nonetheless, how to unbundle decisions encoded in the data perspective of BPMN process models is yet to be investigated.

The main contribution of this chapter is to provide a pattern-based approach to support decision designers and analysts in understanding how the data explicitly represented in a process model may be modeled in a separate, yet integrated decision model. Henceforth, we will refer to such data as *process-related data used by activities to make decisions*. More specifically, we distinguish a set of BPMN process patterns that characterize process-related data used for making decisions in process models and suggest how such data can be represented in DMN decision models. Thus, we also provide a mapping of such BPMN patterns towards the corresponding elements of a DMN model. The final goal is to define a correspondence between the conceptual representation of data in BPMN and DMN elements. This correspondence can be used by designers and analysts to (i) guide the extraction and separation of a decision model from a given process model and (ii) improve understanding of integrated process and decision models, always under a data perspective.

Our proposal deals with process modeling at conceptual level. This means that it aims to improve communication between different stakeholders and it can also be used during process analysis [1], to ease knowledge workers into better refine existing process and decision models.

The steps that we followed to devise the proposed pattern-based approach can be summarized as follows. Firstly, we identified a set of *decision patterns* that describe data elements commonly used in BPMN processes to represent data potentially used by activities to make decisions. To this end, we analyzed the representation and use of data in the BPMN and DMN standards, by also considering real-world processes. Then, we defined a formal mapping of the identified decision patterns towards dedicated (groups of) DMN elements. Finally, we discussed the post-processing of the obtained decision models. To exemplify the applicability of our approach, we show the application of the discussed steps to a process taken from a real-world clinical domain.

The main limitation of the introduced approach is the lack of automation in the use and mapping of the proposed patterns. Indeed, lying in the context of

BPMN process modeling and analysis [1], the presented approach respects a high, conceptual level of abstraction, thus requiring expertise in domain knowledge to understand context-dependent process and decision modeling aspects. Thus, we assume that knowledge about the “as-is” processes of an enterprise is available to decision analysts, regardless of whether it is gathered by multi-disciplinary teams or through direct interaction with stakeholders [20].

The remainder of the chapter structured as follows. Sect. 11.1 introduces a clinical motivating example. Sect. 11.2 describes the approach proposed to design decision models from process-related data used by decision activities. Sect. 11.3 explores how such process-related data are represented in BPMN. Sect. 11.4 introduces the set of identified BPMN decision patterns. Sect. 11.5 presents the mapping of the distinguished BPMN patterns to DMN decision models. Sect. 11.6 discusses the application of our approach to a real healthcare process. Sect. 11.7 concludes the chapter by reviewing our contribution.

11.1 Motivating Example and Foundations

In this section, we introduce a clinical example that shows the motivations behind our proposal and recall key concepts of the BPMN [11] and DMN [47] standards that will be used throughout the whole chapter.

Let us consider the process for diagnosing and treating patients affected by Chronic Obstructive Pulmonary Disease (COPD) [99], carried out by physicians and pulmonologists working in a hospital.

COPD is a chronic and irreversible condition of the lungs, caused by tobacco smoke or exposure to polluted environments. Hospital care for COPD is mostly focused on monitoring and reducing the patient’s symptoms, whose severity determines which is the *stage* of the illness and, accordingly, how the patient must be treated.

In this chapter, we consider dealing with the presentation in a hospital of patients complaining about respiratory discomfort suggestive of COPD. For simplicity, we consider patients that either need to be diagnosed with COPD, or that are known to have COPD and are experiencing a sudden worsening episode, i.e., a “COPD exacerbation” [59, 99]. More detailed processes for the management of patients affected with COPD are reported in Chapter 12.

The main steps of the process introduced in Example 11.1 are represented by the BPMN process diagram of Fig. 11.1.

Example 11.1 (Simple Diagnostic Process for COPD). The process begins when start message event Patient Request (A) is triggered upon receiving a patient request. In BPMN, some events such as message, signal, escalation, and error events, have the capability of carrying data. Among these, messages are used to depict the physical or information items exchanged during a communication between two participants [11]. In the considered case, the request exchanged

between a patient and the receiving physician is assumed to include the patient’s biographical data and the reason of presentation.

Process resource Physician (B) conducts activity Evaluate Request by assessing the degree of emergency based on the patient request. In BPMN, activities may be decorated with markers that denote their type or loop characteristics. In this chapter, we focus on user activities, executed by a human performer, and business rule activities that interact with a business rule engine [11]. Graphically, user activities such as Evaluate request in Fig. 11.1 include a human figure marker, while business rule activities have a marker that resembles a decision table.

Then, the process flow is split into two branches, based on whether the patient has already been assigned a COPD stage or not.

If the patient is already staged, activity Evaluate hospitalization is conducted. Evaluating hospitalization is a decision that requires a physician to consider both the factors described in a text annotation (C) and the patient history recorded in the Electronic Health Record EHR (D), represented as a data store. In hospitals, exacerbation treatment is under the care of a Pulmonologist.

Otherwise, if hospitalization is not required, activity Prescribe therapy is conducted. Then, the physician must Plan an Examination to re-evaluate the patient. The date is chosen based on the patient conditions and on the days available in the ward’s calendar. If the notification of a new Free time slot (F) appears during appointment scheduling, as depicted by the corresponding message boundary non-interrupting event, the physician considers it.

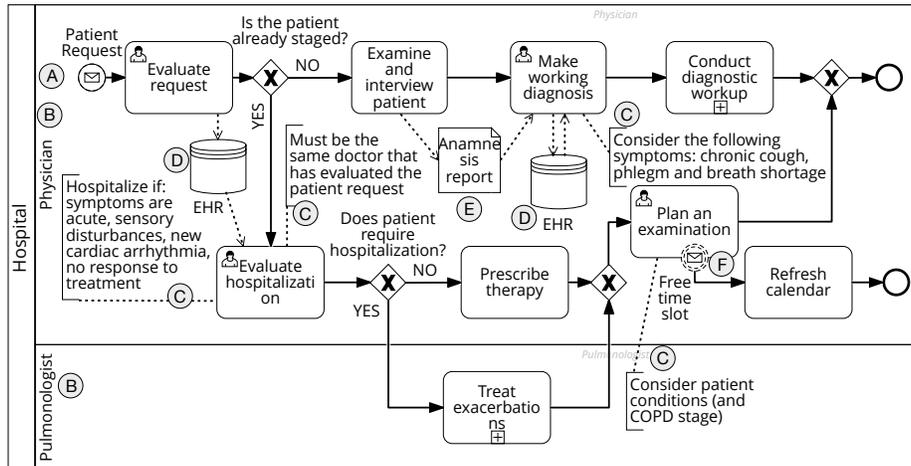


Fig. 11.1: Sample diagnostic process for patients with suspected Chronic Obstructive Pulmonary Disease, conducted in a hospital. Several kinds of data are used by process activities to make decisions: (A) start message events; (B) resources; (C) text annotations; (D) data stores; (E) data objects; and (F) boundary non-interrupting events.

Instead, if the patient is non-staged, the physician must **Examine and interview** the patient to collect data regarding symptoms, signs, and smoking habits. All these data are summarized in the **Anamnesis report** ⑤, depicted as a data object representing volatile data exchanged by activities. Then, these data are used by a physician to formulate a **Working Diagnosis**. Next, a set of diagnostic steps is conducted to either confirm a diagnosis of COPD or solve respiratory discomfort.

The process diagram from Fig. 11.1 shows how data represented by data objects, text annotations, data stores, and events are provided as an input for activities **Evaluate Request**, **Evaluate hospitalization**, **Make working diagnosis**, and **Plan an examination**. Since they all involve evaluation, planning, and other decision-intensive tasks such as clinical diagnosis, the mentioned activities are likely to use the associated data as an input for making decisions. However, since process models are not meant to represent decisions, it is not always easy to understand which process-related data concern also decision-making.

Generally speaking, a decision is the act of determining an output value, from a number of input values, using decision logic to define how the output is determined from the inputs [47, 224]. This definition of decision is in line with the one provided in the DMN standard [47].

An example of DMN Decision Requirement Diagram related to the domain described by the process diagram of Fig. 11.1 is shown in Fig. 11.2.

Fig. 11.2 shows two decisions **Evaluate request** and **Evaluate hospitalization** depicted as rectangles. A patient request is usually evaluated based on the patient's biographical data and on the reason of presentation, which describes the current symptoms considering whether the patient is already undergoing COPD therapy. Only if this is the case, i.e., the patient is already staged and is exacerbating, hospitalization must be evaluated following clinical guidelines to assess symptoms acuity and response to previous treatment.

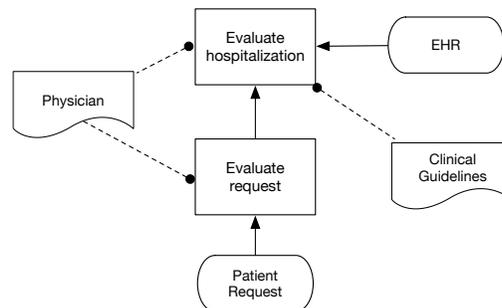


Fig. 11.2: Example of DMN Decision Requirements Diagram, representing a decision **Evaluate hospitalization**, based on sub-decision **Evaluate request**, related input data and knowledge sources.

Accordingly, Fig. 11.2 shows that the output of decision **Evaluate request** is used by decision **Evaluate hospitalization**, together with input data **EHR**, enclosing the results of the request evaluation and, if present, the patient stage. Input data denote the information used as an input by the decision and are depicted as a shape with two parallel straight sides and two semi-circular ends.

Finally, a knowledge source denotes an authority for a decision, which can either be a domain expert responsible for maintaining the decision (e.g., **Physician**) or source documents from which the decision is derived (e.g., **Clinical guidelines**) [47]. Graphically, knowledge sources are depicted as shapes with three straight sides and a wavy one.

The dependencies between DRD elements are expressed by different kinds of requirements. In Fig. 11.2, decision **Evaluate hospitalization** is connected to knowledge sources **Clinical guidelines** and **Physician** by authority requirements, while input data **Patient Request** and **EHR** are connected to decisions **Evaluate Request**, respectively **Evaluate hospitalization**, by an information requirement.

The introduced example shows a DMN DRD that has been derived from a BPMN process model. In such a scenario, the connection between the process and decision models strongly relies on (1) decision activities and (2) data used within the process that also have potential decisional value. However, the identification of which process-related data may have decisional value and what concern (e.g., input data or knowledge source) they address when externalized in a dedicated decision model remains a challenging task for decision analysts and designers.

Since DRDs were devised to bridge business process models and decision logic, in this chapter we focus on the decision requirements level and investigate how process-related data used for decision-making may be represented in DRDs.

We also rely on the definitions of process model (cf. Def. 2.1) and decision requirements diagram (cf. Def. 2.9) formalized in Chapter 2.

However, we slightly modify Def. 2.1 to characterize decision activities, as described by the following Def. 11.1.

Definition 11.1 (Set of Decision Activities). Let us consider a process $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$, conforming to Def. 2.1. Let us call $DA \subseteq A$ the set of decision activities of process model m . Then, it holds that $\forall da \in DA$ either $\alpha_k(da) \mapsto subprocess$ or, if $\alpha_k(da) \mapsto task$ then $\alpha_t(da) \mapsto user$ or $\alpha_t(da) \mapsto business rule$.

According to Def. 11.1 states that decisions can be represented in process models either as subprocesses or as tasks of kind *user* or *business rule*. In this chapter we comprehensively refer to the tasks and sub-processes captured by Def. 11.1 as decision activities.

The DMN standard recommends that decision activities are executed either manually as in *user* tasks, or in a semi-automated manner, as in *business rule* tasks [47]. In BPMN, a *business rule* task provides a mechanism for the process to provide input to a business rule engine and to get the output of calculations that the business rule engine could provide. A *user* task is executed by a human performer. Thereby, decisions may be represented as user or business rule tasks.

Instead, decision subprocesses do not have a specified type, but rather enclose multiple steps of decision-making [110]. Further in this work, we omit the type of decision tasks when the context is clear.

However, it is worth noticing that not all user and business rule tasks represent a decision activity. Indeed, the task type may help decision analysts in the identification of decision-making tasks, but usually approval of stakeholders is needed to properly identify decisions [20].

As exemplified in the DMN standard [47] and in the motivating example of Fig. 11.1, the name of the activity may suggest that decision-making is performed. For instance, when the label of a task starts with a verb that implies a decision, such as “decide”, “evaluate”, or “check”, then the activity is likely to be a decision activity. Also approaches coming from the world of decision mining and decision point analysis may support the identification of decision activities [225].

11.2 A Stepwise Pattern-based Approach

In this section, we describe the main design steps that led to the pattern-based approach proposed in this chapter.

In BPM, the problem of having decisions hard-encoded into process models is definitely relevant as it gives rise to maintainability, flexibility, re-usability, and scalability issues [50, 226]. In particular, process models can contain a lot of data elements that may carry hidden decisional value, such as data objects, data stores, or events. Often, these data are connected to decision activities and used as an input for decision-making. However, by simply observing a process diagram it is difficult to understand which information shall be unbundled and included in a dedicated decision model.

As a possible solution to help decision analysts and designers to recognize these data elements and use them when designing a DMN decision model, we provide a set of BPMN patterns capturing the data perspective of process models and a mapping towards corresponding DMN DRD fragments.

To our knowledge, this work is the first one to focus on the data perspective of BPMN process models at a conceptual level, as other proposals have addressed the same problem dealing with control flow aspects [20, 48] or have focused on the separation of concerns by considering decision services [227, 228].

In this section, we describe the stepwise approach that we followed to design our solution, starting from a given BPMN process model and identifying the process-related data used by decision activities. By grounding our research on the established BPMN [11] and DMN [47] standards, while analyzing real-world scenarios, we provide an approach that is based on well-known design languages, while having empirical evidence in processes coming from different real-world healthcare domains [58, 229].

In the first step, we analyze the BPMN elements defined in BPMN and identify which process elements may carry data. The result of this analysis is

a classification of elements with respect to their relevance in capturing data that can be used for decision-making. Then, in the second step, we define the BPMN patterns including the process elements discovered in the previous step. Finally, in the third step, we define the mapping of BPMN patterns towards possible DMN elements.

Step 1: Analysis of the BPMN standard for Identifying Decision Patterns. For specifying a complete and well-grounded set of decision patterns, we conducted a systematic qualitative analysis of the BPMN standard and identified which elements of the notation may carry data that are used by decision activities. In particular, we focused on those that are represented in BPMN process diagrams and can be visualized by designers, analysts, and stakeholders. The detailed description of the BPMN standard analysis is reported in Sect. 11.3.

Step 2: Definition of Decision Patterns in BPMN. Starting from the results of Step 1, we defined and formalized a set of *decision patterns* capturing the data perspective of BPMN. Each pattern corresponds to a process fragment representing a decision which is based on process-related data, and can be extracted to be represented in a separate decision model. In Sect. 11.4 we present the definition and formalization of the considered set of data-centric decision patterns.

Step 3: Mapping of BPMN Decision Patterns onto DMN DRDs. Once the main decision patterns are identified and defined, they shall be mapped to elements or fragments of DMN Decision Requirements Diagrams. In Sect. 11.5 we proposed and formalized the mapping between BPMN decision patterns and the corresponding elements of DMN DRDs.

The BPMN patterns classified in the second step can be used by analysts and designers to identify decision patterns in a process model, and the mapping defined in the third step can guide the extraction of a set of DRD fragments from the process model. Such DRD fragments constitute an unadapted DMN decision model. In order to obtain a complete DRD, designers must combine the obtained fragments by considering the correlation between different decisions dictated by the structure and the data flow of the process.

The considered process model may be re-engineered by evaluating whether process-related data should be kept in the process model or it is sufficient to keep them in the extracted decision model. In Sect. 11.6, we discuss in detail the steps that should be carried out to use the proposed patterns to unbundle decisions from an existing process model.

11.3 Identification of Process-Related Data Used for Decision-Making

In this section, we discuss which kinds of data may be valuable for decision-making, starting from those explicitly represented in BPMN process models. Then, we complete our analysis by discussing the role of other kinds of process-related data for decision-making.

11.3.1 Analysis of data representation in BPMN

In order to identify which process elements may carrying data that are used by decision activities in process models, we conducted a systematic qualitative analysis of the BPMN standard [11] and considered previous works addressing the suitability of BPMN for modeling decisions and (related) data [70, 230, 12].

The presented analysis aims to distinguish which BPMN elements capture data explicitly represented in process models, that can be used by decision activities for making decisions. We provide a selection of process elements relevant to modeling data valuable for decision-making, starting from data artifacts, events, and resources associated to decision activities.

Since we are considering decision activities as the main BPMN element directly or indirectly “connected” to data elements, our analysis has a defined scope. Moreover, we consider only BPMN graphical elements that can be visualized in a process diagram, thus covering the whole BPMN descriptive process modeling conformance [11]. On the one hand, we have data flow elements (e.g., data objects, data stores) linked to decision activities. On the other hand, some control flow elements can be connected to decision activities and represent other kinds of data in a process model (e.g., events providing input data for decision activities).

In detail, we analyzed version v.2.0.2 of the BPMN standard [11] and considered 116 process elements in total.

The five basic categories of elements found in a BPMN process: (1) Flow objects, (2) Data, (3) Swimlanes, and (4) Artifacts, and (5) Connecting objects. A short summary of the analyzed elements is presented in Table 11.1, where the relevance of BPMN elements for representing data that can be related with decision activities is denoted by a “+” symbol for full relevance, by a “-” symbol for full irrelevance, and by a “+/-” for partial relevance.

Whether process elements connected to decision activities really contain data useful for decision-making may depend on the modeled domain and on the specific process instance. However, at a conceptual level, we may assume that process designers and decision analysts have the domain knowledge expertise required to assess data relevance for decision-making.

Therefore, we consider a group of process elements (e.g., activities, gateways) to be fully relevant with respect to the considered analysis, when all the objects belonging to that group may be connected to decision activities, thus potentially

Cat.	Elements	Relevance	Brief Summary
Flow Objects	Events	+/-	Only start events, intermediate catching events, and boundary events are relevant for our goal. If interrupting a boundary event is relevant when immediately followed by a decision activity located on its outgoing exception flow, if non-interrupting it is relevant when attached to a decision activity.
	Activities	-	Not relevant as activities are units of work: they may represent decisions, but not related data.
	Gateways	-	Not relevant as gateways are not meant to represent data. Exclusive gateways are points where a previously made decision is applied [20].
Data	Data objects Data inputs/outputs	+	Relevant, combined with data associations that connect them to decision activities.
	Data stores	+	Relevant, in combination with data associations that connect them to decision activities.
Swimlanes	Pools	+/-	Relevant only if a pool consists of one lane because a decision activity can belong to only one lane.
	Lanes	+/-	Relevant when it contains data about roles responsible for executing and maintaining decisions.
Artifacts	Groups	-	Not relevant because groups represent only an informal visual mechanism for grouping elements, and they do not carry decisional data.
	Text annotations	+	Relevant, combined with associations that connect them to decision activities.
Connecting objects	Sequence flows	+/-	Sequence flows do not represent data. However, when they are used to connect events to decision activities they are relevant for our goal.
	Conditional flows	-	Not relevant, as they encompass automatic “routing decisions”.
	Message flows	-	Not relevant. Message flows do not connect data to decision activities.
	Associations	+/-	Relevant, when combination with text annotations.
	Data associations	+/-	Relevant when used in combination with data objects, data inputs, data outputs, or data stores.
	Exception flow	+/-	Relevant only when it connects a boundary event with an immediately following decision activity.

Table 11.1: Relevance of BPMN elements for capturing data explicitly represented in process models that can be used by decision activities for making decisions. Full relevance is shown as a “+” symbol, full irrelevance as a “-” symbol, and partial relevance as a “+/-” symbol.

including data useful for decision-making. Similarly, we consider a group of process elements to be partially relevant with respect to the conducted analysis when (i) only part of the elements in that group satisfy our requirements (e.g., events) or (ii) the elements of the group are used to connect data to decision activities (e.g., sequence flows). The group of elements that do not satisfy the above requirements are marked as not relevant.

- (1) *Flow objects* are the main graphical elements used to define the behavior of a business process. BPMN defines three kinds of flow objects:
 - *Events*. Events represent facts that occur during process execution and affect the process flow. BPMN distinguishes events based on (a) their triggering behavior and (b) their position in the process. Some events such as message, escalation, error, signal, and multiple events have the capability to carry data, whereas others such as timer and conditional events may represent (temporal) conditions of the environment that may impact the way a decision is made.
 - (a) Based on their triggering behavior, events are classified in *throwing* and *catching* events depending on whether they release (throw) a trigger or react to (catch) it. For catching events, when the trigger occurs output data become automatically available to be further used in the process.
 - (b) Based on their position in the process flow, events are classified into *start*, *intermediate*, or *end* events. Intermediate events may also be attached to the boundary of activities and may have either an interrupting behavior or a non-interrupting one.

Here, we considered only those events that may affect the execution of decision activities. Start events are relevant for our analysis since they are all catching events and, thus, they may carry data used by immediately following decision activities, connected by a sequence flow edge. Among intermediate events, we included only catching events, as throwing events are not suitable for our purpose. Intermediate events can be connected to a decision activity through a sequence flow (or an exception flow in case they attached to the boundary of a previous activity) or they can be attached to its boundary. In both cases, the decision activity may use the data carried by the event. However, in order for a decision activity to use data carried by a boundary event, the latter must be non-interrupting as an interrupted activity cannot use data. Since nothing can follow end events, they are not relevant for our purpose.

- *Activities*. An activity is used to represent work that is performed within the process. Naturally, activities are not meant to represent data in processes, therefore they are not relevant for our analysis.
- *Gateways*. Gateways are used to control the divergence and convergence of sequence flow in a process. Therefore, gateways are part of process control flow, so they do not represent data in processes. Even when exclusive gateways execute a data-based routing decision, the actual decision shall

be made in the activity preceding the decision gateway [20] (i.e., gateways use the output of a previously made decision to route the flow).

- (2) *Data* is represented by the following elements:
 - *Data objects, data inputs, data outputs.* Data objects describe information that is needed for activities to be performed, or is produced during activity execution. Data inputs and data outputs provide the same information for processes. As we are considering data used by decision activities, data objects and data inputs connected to decision activities are naturally relevant for our selection and should be considered in combination with the data associations that link them. Instead, we exclude data outputs as they may represent the result of a decision, but not the used data. If the data output resulting from a decision is used as an input by another decision, it will be likely duplicated in the process or connected to both decision activities.
 - *Data stores.* A data store allows activities to retrieve or update stored information that will persist beyond the scope of the process. Naturally, when connected to decision activities, data stores may provide information relevant for decision-making. Again, data stores should be linked to decision activities through data associations.
- (3) *Swimlanes* provide a graphical account for participants in a process and group other flow objects in the following two ways.
 - *Pools.* A pool is the graphical representation of participants of the process. It can consist of several lanes. A decision activity can belong to only one lane of the pool. The assignment of a resource to an activity is done by placing the activity within the selected lane of the pool.
 - *Lanes.* Lanes are used to partition pools in order to organize and categorize activities. The assignment of a resource to an activity is done by placing an activity in the lane.

Despite they do not represent data directly, pools and lanes may be associated to decision activities. The information related to the (decisional) role assigned to the resource and corresponding data access restrictions may influence the outcome of a decision. Resources are relevant when they are also responsible for the governance of the decision-making.

- (4) *Artifacts* are used to provide additional information about the process.
 - *Groups.* BPMN defines groups as a “visual mechanism to group elements of a diagram informally” [11]. Groups cannot be connected to any BPMN elements, and they do not affect the process flow, but they rather serve to ease perception of process models to users. They are not relevant for our selection.
 - *Text annotations.* Text annotations are a mechanism for a modeler to provide additional information in natural language to help the readers of a process model. When connected to decision activities, these artifacts can contain significant information for decision-making (e.g., data, constraints on the way a decision activity is executed, or business rules [231]).

Therefore, we include them into our selection, together with the associations connecting them to the corresponding decision activities.

- (5) *Connecting objects* represent different ways of linking flow objects to each other or to additional information represented by artifacts. Although they do not represent data, they are needed to convey the information contained in events and to connect data artifacts to decision activities. Therefore, some connecting elements are given partial relevance.
- *Sequence flows*. A sequence flow can be used to show a partial ordering of activities in a process. As previously mentioned, we are interested in the sequence flows used to connect start/intermediate events that carry data to the decision activities that use them. For this reason, their relevance for the analysis goal is partial.
 - *Conditional flows*. A conditional flow is a special kind of sequence flow having a condition expression that is evaluated at runtime to determine whether that process path can be followed or not. Despite relying on the evaluation of a data-based condition, conditional flows encompass an automatic “routing decision”. Since decision-wise their behavior is similar to that of exclusive gateways, they are not relevant for our selection.
 - *Message flows*. A message flow is used to show the flow of messages between two participants that are prepared to send and receive them. BPMN defines a special kind of activities and events, called send and receive activities (events), to perform such information exchange. Since receive tasks in BPMN cannot represent decision-making, the only way to use the content of a message for decision-making is by considering catching message events properly connected to decision activities. Thus, the message flow is not relevant for our purpose.
 - *Associations*. An association is used to link text annotations with other flow elements. When connected to a decision activity, text annotations can contain any kind of comment written in natural language. These comments may include data/information used by decision activities. Thus, we include them in our selection with partial relevance.
 - *Data associations*. A data association is used to link data objects, data inputs, data outputs, or data stores with activities or events. Since data objects or data stores may contain data used by decision activities, the data associations connecting such data artifacts to decision activities are relevant for our purpose.
 - *Exception flow*. An exception flow occurs outside the normal flow of the process and it originates from an intermediate event attached to the boundary of an activity that occurs during process execution. As we considered intermediate events followed by decision activities, we shall include the exception flow when it connects a boundary event with a following decision activity. For this reason, its relevance is partial.

11.3.2 Other Kinds of Data Used for Decision-Making

Here, we complete our analysis by considering (implicit) process-related data that are often used for decision-making. In detail, we discuss why some data (i) do not need to be externalized in decision models or (ii) are used for representing process-related information, without being explicitly included in BPMN models.

In general, not all kinds of data specified within process models need to be externalized in dedicated decision models. This happens when data elements include no decision input/knowledge or when execution/external events are used exclusively to automatically route the process flow. Indeed, decision models are often employed to represent and improve understanding of operational decisions, mostly focusing on decision algorithms and logics to support human decision-making. Therefore, lower-level process logic should not be represented by decision models at the requirements level.

As an example, let us consider catching events in the configuration of event-based gateways. In this case, event occurrence drives the process flow based on “instantaneous decisions” that are managed by process engines. Such kind of decisions made by a process engine should not be included in decision models, as they are based on process logic and routing rules, that depend on event processing.

For instance, let us consider an event-based gateway having events “Receive accept e-mail” and “Receive reject e-mail” in its configuration. Depending on whether the request is accepted or requested, the process behaves differently. Yet, the “decision” of which path must be taken is a mere reaction to a decision, made outside the scope of the process (in this case, made by the sender) and whose output events, determine how the process should proceed its execution.

From another perspective, information used for decision-making and included in decision models may not be explicitly represented in process models. For instance, domain knowledge, Key Performance Indicators (KPIs), or process execution logs often drive decision-making, but they are represented as meta-information or at a lower modeling level rather than being included in BPMN process models.

Reference data and domain knowledge. Human decision-making is performed by organization personnel, who interpret data according to their knowledge, personal experience, past organizational trends, and reference information enclosed within textual guidelines [58]. Indeed, background domain knowledge is often used to complement known decision inputs, before outputs are inferred [232].

Key performance indicators (KPIs). Indicators can be defined during process specification to evaluate the process execution performance and to measure the progress towards the achievement of business and organizational outcomes, based on specific objectives and milestones. KPIs can have a *local* or a *global* scope, depending on how and when they impact the process [59]. Local KPIs are often defined quantitatively to measure the process performance

and can be used to adapt the process dynamically. An example of local KPI is activity duration: If a certain task lasts longer than expected, future process steps can be skipped. Conversely, global KPIs are defined qualitatively by aggregating information regarding several processes, and can be used to support higher level decisions, related to process re-engineering or role re-definition. For instance, process managers may decide to re-design part of a process, according to customers satisfaction reports. Decisions can be linked to the KPIs and objectives of the organization they impact. These, may coincide with the global KPIs referred by processes within which the decisions are made [233].

Execution information. Likewise KPIs, information extracted from process logs can also be used to dynamically adapt the process flow to prescribed execution requirements in order to achieve improved decision outcomes [225, 233].

Despite not being explicitly represented in BPMN process models, the information enclosed within domain knowledge, KPIs, and process logs may be integrated by decision designers once DMN models are created.

11.4 Decision Patterns for the Data Perspective of BPMN

In this section, we define and formalize a set of patterns that combine the process elements selected during the analysis of BPMN with decision activities.

Fig. 11.3 shows the set of decision patterns *II1–II6*, that was derived from the systematic analysis of the BPMN standard [11] and whose empirical evidence was found in real-world healthcare process models from prior research [20, 59, 229].

Each pattern corresponds to a process fragment, that is, a subset of the process model defined as a tuple. Thereby, each pattern contains a connected subgraph of a process model, which represents a decision based on process-related data that can be externalized into a separate decision model. The introduced patterns identify common ways to represent data related to decisions in process models. When the corresponding decision model is designed, a stakeholder (or a decision analyst having the required domain knowledge expertise) shall establish whether such data are significant for decision-making, as done in [20].

For readability, we always show one process element of a kind attached to a decision activity, thus intentionally omitting the representation of multiple elements connected to a single decision activity. For example, pattern *II1* shows only one data object connected with a decision activity, although the formalization of *II1* provides that multiple data objects can be attached to a single decision activity. For each pattern, we provide its description, formalization, connection with the motivating example of Fig. 11.1, and discuss possible variants.

It is worth noticing that we do not include decision outputs in our patterns. This gives us more flexibility when identifying decision activities as we select those that necessarily have some input data, but may or may not have an output explicitly represented in the process model. Indeed, decision outputs do not

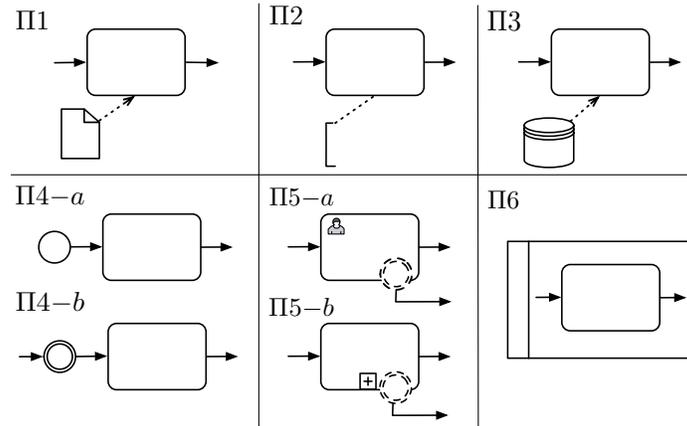


Fig. 11.3: Summary of BPMN decision patterns. The possible plurality of the elements connected to a single decision activity is omitted for readability.

have to be explicitly represented as data elements in a process, as they can be captured by the output flows of an exclusive gateway [50]. Besides, the presence/absence of a decision output represented as data in the process does not change the structure of the obtained DMN decision requirements diagram.

PI1 – Data objects used by a decision activity. Data objects represent data that is generated, consumed, and exchanged by process activities [11]. The BPMN standard does not say much about the inner structure of a data object. In practice, data objects can be used to capture a single data class of a databases, a collection of data classes, or a complex document [182]. Therefore, the information contained in data objects may or may not be used for decision-making and can be of any kind. When used for decision-making, the information contained in data objects is represented as input data for decision-making [20].

PI1 - DATA OBJECTS USED BY A DECISION ACTIVITY

DESCRIPTION

A decision activity uses the information contained within one or more data objects attached to it as input data for decision-making.

ELEMENTS

Decision activity da , data objects do_1, \dots, do_n attached to da through directed data associations $(do_1, da), \dots, (do_n, da)$.

FORMALIZATION

Let $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$ be a process model. A decision activity $da \in DA \subseteq A$ uses the set of data objects $DO' \subseteq DO \subseteq DN$ if and only if $\forall do \in DO', (do, da) \in F$.

II1 is a process fragment that consists of decision activity $da \in DA$, a set of data objects $DO' \subseteq DO$, and a set of data associations $F_{DO'} = \{(do, da) \mid do \in DO'\} \subseteq F$.

As an example, let us consider an healthcare process, as the one of Fig. 11.1. A data object can represent the identifier of the patient, the list of previously prescribed therapies or a whole document, such as the anamnesis report. Regardless of data granularity, this information is related to a specific patient and it is represented as input data for the decision activity.

II2 – Text annotations used by a decision activity. Text annotations attached to activities are used to provide additional text information about that activity [11]. Therefore, as there is no limitation on their content, text annotations can represent both data, and requirements for decision-making. For instance, textual annotations may represent business rules integrated into process models [234].

According to our analysis, annotations are used in processes to capture comments related to which data can be used to execute the task, constraints on the execution of the task (such as deadlines, procedural aspects to be observed, or business rules), and information regarding activity actors and authorities for decision-making. Of course, the same text annotation can contain diverse kinds of data and not all of them may be needed for decision-making.

II2 - TEXT ANNOTATION USED BY A DECISION ACTIVITY

DESCRIPTION

A decision activity uses the information contained within one or more text annotations attached to it as input data for decision-making or to provide details about decision sources or decision makers.

ELEMENTS

Decision activity da , text annotation ta_1, \dots, ta_n attached to da through undirected data associations $(ta_1, da), \dots, (ta_n, da)$.

FORMALIZATION

Let $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$ be a process model. A decision activity $da \in DA \subseteq A$ uses the set of text annotations $TA' \subseteq TA$ if and only if $\forall ta \in TA', (ta, da) \in T$. *II2* is a process fragment that consists of decision activity $da \in DA$, a set of text annotations $TA' \subseteq TA$, and a set of undirected associations $T' = \{(ta, da) \mid ta \in TA'\}$.

For example, let us consider the process of Fig. 11.1. Text annotation “Consider patient conditions and stage”, associated to decision activity Plan an examination suggests that the stage of COPD and the conditions of the patient must be

considered during examination planning. In the considered domain, the more severe is the patient the earliest and most frequently medical checks should be planned. Instead, annotation “Must be the same doctor that has evaluated the patient request” does not provide any information useful for decision-making.

II3 – Data stores used by a decision activity. Data stores represent data that persist beyond the scope of the process [11]. Data stores are often used to represent databases that support process execution. Therefore, information retrieved from data store is likely to represent input data for decision activities.

In the process of Fig. 11.1, data store EHR provides input data for decision activity Evaluate hospitalization. The same data store is connected also to decision activity Make working diagnosis.

II3 – DATA STORES USED BY A DECISION ACTIVITY

DESCRIPTION

A decision activity uses the information retrieved from one or more data stores as input data for decision-making.

ELEMENTS

Decision activity da , data stores $ds_1 \dots ds_n$ attached to da through directed data association $(ds_1, da), \dots, (ds_n, da)$.

FORMALIZATION

Let $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$ be a process model. A decision activity $da \in DA \subseteq A$ uses the set of data stores $DS' \subseteq DS \subset DN$ if and only if $\forall ds \in DS', (ds, da) \in F$. II3 is a process fragment that consists of decision activity $da \in DA$, a set of data stores $DS' \subseteq DS$, and a set of data associations $F_{DS'} = \{(ds, da) \mid ds \in DS'\} \subseteq F$.

II4 – Event data used by a subsequent decision activity. In BPMN, some events such as message, escalation, error, and signal events have the capability to carry data [11]. In addition, timer events also encode temporal information that may have a value for business decision-making [54] and data obtained within a certain time frame may be used for re-evaluating decisions in real time [235].

When immediately followed by decision activities, the information carried by events can be used as input data for decision-making. According to the position of the triggered event, that is, either start or intermediate, we distinguish two variants of this pattern.

II4 – EVENT DATA USED BY A SUBSEQUENT DECISION ACTIVITY

DESCRIPTION

A decision activity uses the information carried by a previously occurred event as input data for decision-making.

VARIANTS

- Π_4-a is a process fragment that consists of start event e , control flow $(e, da) \in C$, and decision activity da .
- Π_4-b is a process fragment that consists of intermediate event e , control flow $(e, da) \in C$, and decision activity da .

FORMALIZATION

- Π_4-a : Let $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$ be a process model. A decision activity $da \in DA \subseteq A$ uses data carried by a previously occurred start event $e \in E_{start}$ if and only if $(e, da) \in C$.
- Π_4-b : Let $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$ be a process model. A decision activity $da \in DA \subseteq A$ uses data carried by a previously occurred intermediate event $e \in E_{int}$ if and only if $(e, da) \in C$.

For example, let us consider the process of Fig. 11.1. The request of a patient is represented as a message start event that contains the patient’s biographical data and the reason of presentation, and triggers the beginning process. In this case, the information included in the message is used by the subsequent decision activity **Evaluate request**.

Boundary interrupting events attached to an activity and leading to a decision activity located on the outgoing exception flow are a special case of pattern Π_4-b .

Π_5 – Boundary event data used by a decision activity. Sometimes boundary events can be directly attached to decision activities. If they interrupt the activity, the information that they carry cannot be used by the decision that has been interrupted. Instead, if the boundary event is non-interrupting the carried data may be used while the decision is being made: non-interrupting events trigger the corresponding event handlers that run in parallel with the on-going decision activity [54].

This consideration holds for user tasks or for subprocesses representing decisions and spanning for a certain amount of time, during which the boundary event can occur. On the contrary, a standalone business rule task representing a decision activity invokes the associated business rule or decision model upon activation [11, 47]. This is executed instantly, thus leaving no time for event occurrence.

Π_5 – BOUNDARY EVENT DATA USED BY A DECISION ACTIVITY

DESCRIPTION

A decision activity uses the data carried by one or more non-interrupting boundary events as input data for decision-making.

VARIANTS

- *II5-a* is a process fragment that consists of decision task da of type user and one or more non-interrupting boundary events eb_1, \dots, eb_n attached to its border.
- *II5-b* is a process fragment that consists of decision subprocess da , and one or more non-interrupting boundary events eb_1, \dots, eb_n attached to its border.

FORMALIZATION

- *II5-a*: Let $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$ be a process model. A running decision activity $da \in DA \subseteq A$ may be influenced by the occurrence of a set of non-interrupting boundary events $E'_B \subseteq E_B$ if and only if $\forall eb \in E'_B, \beta'(da) = E'_B$ where $\beta' : A \rightarrow 2^{E'_B}$ is the restriction of β to E'_B , eb occurs while da is running, $\epsilon_k(eb) = \text{non-interrupting}$, $\alpha_k(da) = \text{task}$, and $\alpha_t(da) = \text{user}$.
- *II5-b*: Let $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$ be a process model. A running decision activity $da \in DA \subseteq A$ may be influenced by the occurrence of a set of non-interrupting boundary events $E'_B \subseteq E_B$ if and only if $\forall eb \in E'_B, \beta'(da) = E'_B$ where $\beta' : A \rightarrow 2^{E'_B}$ is the restriction of β to E'_B , eb occurs while da is running, $\epsilon_k(eb) = \text{non-interrupting}$, and $\alpha_k(da) = \text{subprocess}$.

For example, in Fig. 11.1 non-interrupting boundary event **Free time slot** may notify that a new time slot is available in the agenda, while the physician is planning the future examination of a patient. This may occur whenever the hospital scheduling system detects that another patient has cancelled or rescheduled an appointment. The physician may refresh the calendar and see whether the new availability may be an option for the considered patient: if the vacant time slot is too far away with respect to the severity of the patient, the physician may simply ignore this information.

II6 – Decision activity associated to a specific role/resource. In BPMN, lanes are used to partition and organize the activities of a process and, often, they are used to represent resources, internal roles, systems or departments [11].

Here, we take the view that the information regarding internal roles associated to a decision activity may be used to determine who is the authority for the decision and which must be his or her expertise. Accordingly, the DMN standard defines knowledge sources “to model governance of decision-making by people (e.g., a manager), regulatory bodies (e.g., an ombudsman), documents (e.g., a policy booklet) or bodies of legislation (e.g., a government statute)” [47].

Besides, it is likely that different process roles may have diverse decisional power, also based on access privileges to sensitive information. If the same decision can be made by resources having different/hierarchical roles in the process, these may correspond to multiple/hierarchical authorities in a decision model.

$\Pi 6$ – DECISION ACTIVITY ASSOCIATED TO A SPECIFIC RESOURCE

DESCRIPTION

A decision activity is executed by a resource having a specific process role. The information related to the role can be used to determine whether the decision maker is also an authority for the decision.

ELEMENTS

Decision activity da and associated resource r .

FORMALIZATION

Let $m = (N, DN, C, TA, F, T, R, \alpha_k, \alpha_t, \beta, \gamma_r, \gamma_{ty}, \epsilon_{tr}, \epsilon_{ty}, \epsilon_k, \rho, \mathcal{L})$ be a process model. A decision activity $da \in DA \subseteq A$ is executed by a process resource $r \in R$ having a specific role if and only if $\rho(da) = r$.

The process of Fig. 11.1 is contained in one main pool **Hospital** partitioned into two lanes, **Physician** and **Pulmonologist**. Whereas pulmonologists are not responsible for the governance of decision-making, for crucial decision activities, such as diagnosis, **Physicians** have a dual role. In the process they are responsible for taking care of the patient and for making the right clinical decisions, while in the decision model, they are responsible for defining the treatment and diagnostic steps, and for maintaining clinical guidelines.

11.5 Mapping BPMN patterns to DMN DRDs

In this section, we introduce the formal mapping between the set of proposed decision patterns and DMN decision requirements diagrams.

To this end, we define the set $\Delta = \{\Delta 1, \dots, \Delta 6\}$ of DRD fragments, which corresponds to the set of decision patterns $\Pi = \{\Pi 1, \dots, \Pi 6\}$.

Our mapping is based on a correspondence relation $\Gamma = \{\Gamma 1, \dots, \Gamma 6\}$, such that $\Gamma \subset \Pi \times \Delta$. The DRD fragments $\Delta 1, \dots, \Delta 6$ are subgraphs contained in a DRD as a tuple, such that $d \in D, I' \subseteq I, K' \subseteq K, IR' \subseteq IR, AR' \subseteq AR$.

The correspondence relation Γ is visualized with the help of correspondence graphs in Fig. 11.4 and a detailed discussion of the mapping is provided below. Since some BPMN elements may be mapped to multiple DMN elements, a few DRD elements are considered to be optional, meaning that, even if they have been identified as data useful for decision-making, not all the corresponding BPMN elements have to always be mapped to them. Fig. 11.4 depicts such DRD elements with a gray-shaded filling. For readability reasons, we do not show the possible plurality of elements of the same kind connected to a decision activity.

In detail, the correspondence relation Γ always maps decision activity $da \in DA$ of each BPMN fragment constituting a decision pattern to decision $d \in D$ of the corresponding DRD fragment. Bearing this in mind, below we discuss only the correspondences of the other elements for each mapping.

BPMN fragment	Correspondence graphs	DRD fragment
$\Pi 1$	$\Gamma 1$	$\Delta 1$
$\Pi 2$	$\Gamma 2$	$\Delta 2$
$\Pi 3$	$\Gamma 3$	$\Delta 3$
$\Pi 4-a$	$\Gamma 4$	$\Delta 4$
$\Pi 4-b$		
$\Pi 5-a$	$\Gamma 5$	$\Delta 5$
$\Pi 5-b$		
$\Pi 6$	$\Gamma 6$	$\Delta 6$

Fig. 11.4: Mapping of BPMN patterns introduced in Fig. 11.3 to corresponding DRD fragments. The shading of the DRD shapes means that the elements are optional for modeling and execution.

$\Gamma 1$ A mapping $\Gamma 1$ is a correspondence relation between the BPMN pattern $\Pi 1 = (da, DO', F_{DO'})$ and the DRD fragment $\Delta 1 = (d, I', IR')$. Each data object $do \in DO'$ is mapped onto input data $i \in I'$ since they both represent

operational data used by a decision (activity). Each corresponding data association $(do, da) \in F_{DO'}$ corresponds to information requirement $ir \in IR'$.

- $\Gamma 2$** A mapping $\Gamma 2$ is a correspondence relation between the BPMN pattern $\Pi 2 = (da, TA', T')$ and the DRD fragment $\Delta 1 = (d, I', K', IR', AR')$. Text annotation $ta \in TA'$ corresponds to input data $i \in I'$ whenever it represents operational data needed for decision-making. In this case, undirected association $t \in T'$ is mapped to information requirement $ir \in IR'$. Alternatively, a text annotation may include information related to documents used for making the decision. In this case, $ta \in TA'$ is mapped onto knowledge source $k \in K'$ whenever it represents a non-functional requirement for decision-making. In this latter case, undirected association $t \in T'$ is mapped onto authority requirement $ar \in AR'$. Yet, since text annotations do not always represent both input data and knowledge sources, in the DRD fragment i and k are represented as optional, as highlighted by the shading.
- $\Gamma 3$** A mapping $\Gamma 3$ is a correspondence relation between the BPMN pattern $\Pi 3 = (da, DS', F_{DS'})$ and the DRD fragment $\Delta 3 = (d, I', IR')$. Each data store $ds \in DS'$ corresponds to input data $i \in I'$ as data stores are used to represent databases used by decision activities. Each data association $(ds, da) \in F_{DS'}$ corresponds to information requirement $ir \in IR'$.
- $\Gamma 4$** A mapping $\Gamma 4$ is a correspondence relation between the BPMN pattern $\Pi 4 = (da, e, (e, da))$ and the DRD fragment $\Delta 4 = (d, i, ir)$. The mapping considers both pattern variants $\Pi 4-a$ and $\Pi 4-b$, as they have the same formal structure. Event e is either a start or an intermediate event that carries data influencing the decision, and it corresponds to input data $i \in I$. The corresponding control flow edge $(e, da) \in C$ is mapped onto information requirement $ir \in IR$.
- $\Gamma 5$** A mapping $\Gamma 5$ is a correspondence relation between the BPMN pattern $\Pi 5 = (da, E'_B, \beta')$ and the DRD fragment $\Delta 5 = (d, I', IR')$. Each boundary event $e_b \in E'_B$ carries data used for making the decision, and it corresponds to input data $i \in I'$. Similarly, the corresponding relation $\beta'(da) \mapsto E'_B$ which associates boundary events to one activity is mapped onto information requirement $ir \in IR'$, even if it is not visualized in the process diagram. Since in both cases the boundary event may not occur, the corresponding input data element i in the DRD fragment is shown as optional.
- $\Gamma 6$** A mapping $\Gamma 6$ is a correspondence relation between the BPMN pattern $\Pi 6 = (da, r, (da, r))$ and the DRD fragment $\Delta 6 = (d, k, ar)$. Resource r is mapped to knowledge source $k \in K$ if it represents a non-functional requirement for decision-making (e.g., the authority responsible for decision governance), and then $\rho(da) \mapsto r$ should be mapped onto authority requirement $ar \in AR$.

The choice of representing resources as knowledge sources comes from the definition of the latter ones in the DMN standard [47]. Knowledge sources may be domain experts responsible for defining or maintaining them, source documents from which business knowledge models are derived, or sets of test cases with which the decisions must be consistent.

In addition, all *decisions* belonging to the introduced DRD fragments can also reference *business knowledge models* enclosing decision logic [47]. This is recommended if the decision logic is reused by multiple decisions. In this case, the corresponding *knowledge requirements* should also be provided. However, since business logic is typically not present in graphical process models, we did not include business knowledge models in our mapping.

11.6 Applying the Pattern-based Approach to Real-World Process Models

In this section, we describe the different steps that decision analysts may follow when applying the introduced pattern-based approach to derive a DMN decision model related to an existing BPMN process model.

For each step, we discuss the design challenges and limitations, and show its application to a real-world example taken from the clinical domain of Chronic Obstructive Pulmonary Disease (COPD) [99].

11.6.1 The Reference Healthcare Domain

As partly introduced in Sect. 11.1, COPD is an irreversible condition of the lung requiring continuous multidisciplinary care. In the remainder, we focus on the BPMN process depicted in Fig. 11.5, corresponding to the one of Fig. 11.1 where the previously collapsed subprocess *Diagnostic workup* is expanded. The depicted process model focuses on the unplanned presentation of patients in a hospital complaining of acute respiratory symptoms that can be referable to COPD.

The process has been modeled by interviewing the personnel of a German hospital, and by combining the retrieved information with clinical practice guidelines [99] and with results obtained by previous research [59, 58]. Interviews were conducted by adopting the approach introduced in [58] and knowledge about clinical practice was acquired during process design by directly observing how physicians operate in hospital.

In the presented case study, one of the main goals of process design was to support the identification and standardization of critical steps in the treatment of COPD and of the local and global decisions underlying the whole care process [58]. Thus, decisions were also identified by designers during process design and were submitted to stakeholders for approval.

In detail, we asked stakeholders targeted questions, regarding which and how many decisions could be discerned in the processes, whether and how these would influence the process flow, and which were the data requirements for decision-making. In the studied context, hospital physicians play the most important role in the process in terms of decision-making, as the outcome of their decisions is often used as an indicator of process quality [58]. In addition, they are also responsible for maintaining the decisions. For specialized pulmonary assessment and for the related decision-making physicians are helped by pulmonologists.

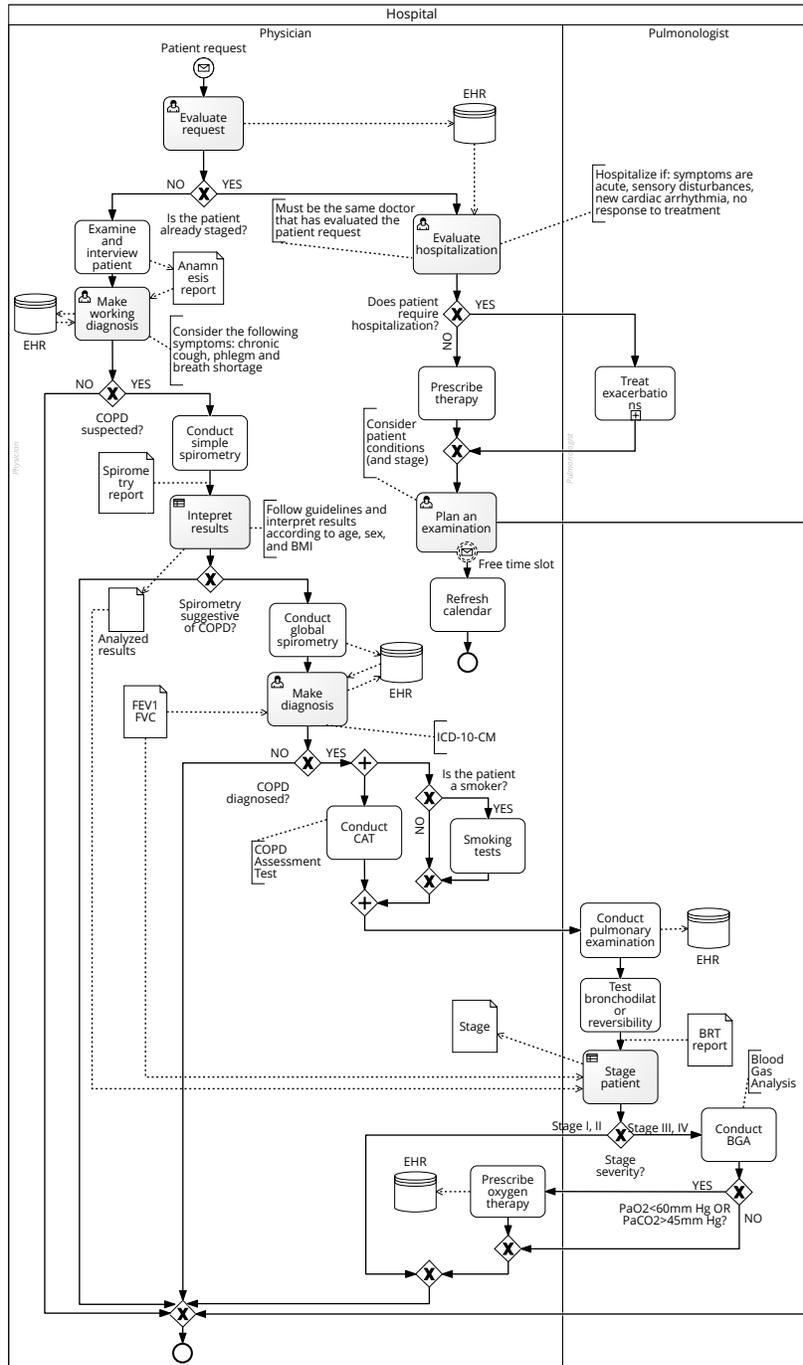


Fig. 11.5: Process for dealing with unplanned presentation of patients possibly affected by Chronic Obstructive Pulmonary Disease (COPD). Decision activities are shaded.

The process of Fig. 11.5 begins when a **Patient request** is received by a physician, who determines whether the patient has already been diagnosed with a certain stage of COPD or a new diagnosis for the present condition is required. This decision is portrayed by user task **Evaluate request**, and it is mostly based on the information included in the request and on the physical conditions of the patient.

If the patient is experiencing an exacerbation causing symptom worsening, hospitalization is evaluated, based on the patient's history recorded in the Electronic Health Record EHR and on current symptoms. If the patient does not require hospitalization, therapy is prescribed, otherwise the worsening is treated in hospital. In either case, a future examination is planned: the chosen date depends on the patient's conditions and on doctors' availability, which is constantly updated as explained in Sect. 11.1.

If the patient has not been previously diagnosed with COPD, respiratory symptoms and risk factors must be collected through an **Examination with Interview** that serves as a basis for medical diagnosis. A diagnosis of COPD consists of two phases: (i) a working diagnosis, which requires the physician to identify signs that may lead to COPD suspicion based on the collected anamnesis data, and (ii) a definitive diagnosis of COPD, which may confirm the working diagnosis based on the results several medical tests.

Spirometry is required by international clinical guidelines to make a diagnosis of COPD. Simple spirometry provides information about the post-bronchodilator forced expiratory volume in one second (FEV1) and the forced vital capacity (FVC) of the patient. When the ratio between these two values is lower than 0.7, the presence of airflow limitation that is not fully reversible is confirmed. Spirometric results are interpreted according to predefined thresholds, but they also have to take the patient age, sex, and Body Mass Index (BMI) into account [99].

Global spirometry provides more accurate and detailed results, and it is used to assess COPD severity. Spirometric outcome interpretation is a delicate decision as it is the main piece of information used by physicians to **Make diagnosis** and by pulmonologists to **Stage patient**.

Once COPD has been diagnosed, additional tests and examinations, such as the COPD Assessment Test (CAT), smoking tests, and a pulmonary examination are conducted to grade the severity of the disease according to specific stages, defined in clinical guidelines [99]. COPD staging is represented as a business rule task, conducted under the supervision of a pulmonologist, who classifies the patient into one of the four admissible stages based on the results of the previously conducted spirometry and of a bronchodilator reversibility test.

Finally, after staging the patient, arterial blood gas analysis is conducted to see if temporary or permanent oxygen therapy must be administered.

In the following sections, we refer to this example and explain how to (i) identify the decision patterns described in Sect. 11.4 in a process model, (ii) apply the mapping described in Sect. 11.5 to derive the corresponding DRD

fragments, (iii) obtain the final DMN model and, where appropriate, consider process model refactoring or decision model post-processing.

11.6.2 Identification of Decision Patterns in a Process Model

First of all, a decision analyst should detect all the decision activities in a process, if this was not already done during process design. To this end, the analyst may rely on his or her domain knowledge, or may consult stakeholders. This step can be supported by the natural language analysis of activity labels [236] that recommends candidate decision activities in a process model. Also decision points used for decision mining typically correspond to decision activities [50, 225].

Another way to facilitate this first identification step is to detect the exclusive split gateways in a given process model and consider the activities immediately preceding the gateways as candidate decision activities. Again, a stakeholder should confirm that the selected activities represent real business decisions [20].

Then, for each identified decision activity, the analyst should check whether it is connected to any events, data nodes, text annotation, or resource in accordance with the patterns *II1–II6* described in Sect. 11.4.

Example 11.2. The analysis of the BPMN process model of Fig. 11.5 was conducted with the help of stakeholders and led to the detection of seven decision activities, listed in Table 11.2.

Then, we extracted the process fragments corresponding to the introduced BPMN decision patterns, as shown in the upper part of the four rows of Fig. 11.6.

11.6.3 Mapping of BPMN patterns to DRD fragments

Finding patterns *II1–II6* in a process model does not always imply that the discovered data are valuable for decision-making. Indeed, patterns capture the representation of data in a process model, but do not provide details about the value of such data. Thus a decision analyst should always refer to the application domain to assess whether the identified pieces of information are actually involved in the decision-making. This also depends on the level of detail of the considered process model: if a process model is designed to include a lot of technical information, some data connected to decision activities may not be an input for decisions, as they may be needed in the process for other purposes such as, for instance, conformance checking or activity execution.

In particular, special care is needed for text annotations (*II2*), boundary non-interrupting events (*II5*), and resources (*II6*).

Text annotations are often informal comments, written in natural language, and no limit is set on their content. For this reason, they may contain useful information as well as additional data that is not useful for making decisions.

DECISION ACTIVITY	IDENTIFICATION OF DECISION PATTERNS
Evaluate request	It uses the information related to the patient and cause of presentation, carried by start message event Patient Request , as input data (<i>II4-a</i>). The physician who evaluates the incoming patient is responsible for making and maintaining the decision (<i>II6</i>).
Make working diagnosis	It is mostly based on data regarding the patient's health condition, gathered from the <i>Anamnesis report</i> (<i>II1</i>), represented as a data object, and from the EHR (<i>II3</i>). All data are interpreted according to clinical guidelines, summarized by the linked text annotation (<i>II2</i>). A physician is the authority responsible for the working diagnosis (<i>II6</i>).
Interpret results	It takes data object Spirometry report as an input (<i>II1</i>) and evaluates its content based on clinical guidelines (<i>II2</i>), but also taking into consideration the patient's age, sex, and BMI (<i>II2</i>). This latter information is used as input data and, thus, it is represented as a DRD input in Fig. 11.6, while guidelines are mapped onto a knowledge source.
Make diagnosis	It relies on the results of global spirometry and on the working diagnosis, both stored in the EHR (<i>II3</i>). The results of global spirometry are compared with reference values FEV1 and FVC, represented as a data object (<i>II1</i>). Text annotation ICD-10-CM denotes diagnosis encoding, thus being irrelevant decision-making. A physician is responsible for making the diagnosis and is also an authority for that decision (<i>II6</i>).
Stage patient	Staging is carried out by a pulmonologist, who classifies the patient into a specific COPD stage, depending on the previously made spirometry interpretation results, on the and on the information contained in the BRT report , both represented as data objects (<i>II1</i>). Process resource Pulmonologist is not responsible for the governance of patient staging and, thus, it is excluded from the selection of the patterns.
Evaluate hospitalization	It must consider the patient's current and past symptoms, and the history of previous exacerbations or treatments, as stated in the attached text annotation "Hospitalize if: symptoms [...]" (<i>II2</i>). Data regarding both previous exacerbations and symptoms are retrieved from the EHR (<i>II3</i>), where the output of activity Evaluate request is also recorded. Physicians are responsible for evaluating hospitalization and for maintaining that decision (<i>II6</i>). Text annotation "Must be the same doctor that has evaluated the patient request" does not contain information valuable for decision-making and, thus, it is discarded.
Plan an Examination	It is executed by a physician (<i>II6</i>), who considers the patient's stage as an input, as described in the attached text annotation (<i>II2</i>). Besides, real-time data about physicians availability must be considered: if a free time slot becomes available it is used during the planning (<i>II5-a</i>).

Table 11.2: Decision activities identified in the process of Fig. 11.5 and related decision patterns.

Domain knowledge expertise, often resulting from the interaction with stakeholders, is necessary to discern useful data in this context.

For example, in the process of Fig. 11.5 decision activity **Evaluate hospitalization** is linked to two different text annotations, one "Hospitalize if: symptoms are acute, sensory disturbances, new cardiac arrhythmia, no response to treatment" provides useful information for deciding whether the patient should be

hospitalized, the other one “Must be the same doctor that has evaluated the patient request” contains information that is needed only for constraining activity execution.

The same considerations may be done for boundary non-interrupting events, as the received trigger may or may not contain data that have the potential to change the decision outcome at run-time. Being an intrinsic part of process models, resources are usually not considered in decision models. However, since process resources may have a decisional role, they should also be represented in decision models, especially when they are also responsible for the governance of a decision. Particularly, if the process resource is also domain experts responsible for defining or maintaining the decision he or she also makes, then the BPMN resource should be mapped towards a DMN knowledge source.

Example 11.3. In the example of Fig. 11.5, physicians have a dual role: in the process they are responsible for taking care of the patient and for making the right clinical decisions, while in the decision model, they are responsible for defining the treatment and diagnostic steps, and for maintaining clinical guidelines.

For the considered scenario, all of the process resources detected by pattern *II6* (i.e., physicians) were mapped to knowledge sources, as process resources where directly responsible of decision governance. Pattern *II2* is mapped onto knowledge sources or input data, depending on the kind of information it includes. Pattern variants *II4-b* and *II5-b* are not found, as the considered process does not have intermediate events nor decision subprocesses. The obtained DRD fragments are shown in the lower part of the four rows of Fig. 11.6.

11.6.4 Post-Processing of Decision and Process Models

Once a set of the DRD fragments is derived, two additional post-processing steps need to be carried to obtain a decision model that represents business decisions previously encoded in the process.

In the first place, a complete decision requirement graph needs to be constructed by combining the obtained fragments into one or more DRDs. This should be done by taking into account the inter-dependencies between different decisions, some of which are dictated by the process control flow or by the use of shared output/input data, and by eliminating repeated elements.

For example, when a decision activity produces data objects as an output that are reused as inputs by another decision activity, an information requirement should be added between the two decisions [20]. Similarly, decision activities may be related by reading and writing operations on a data store. If no connection with other decisions can be determined, the DRD fragment shall remain independent.

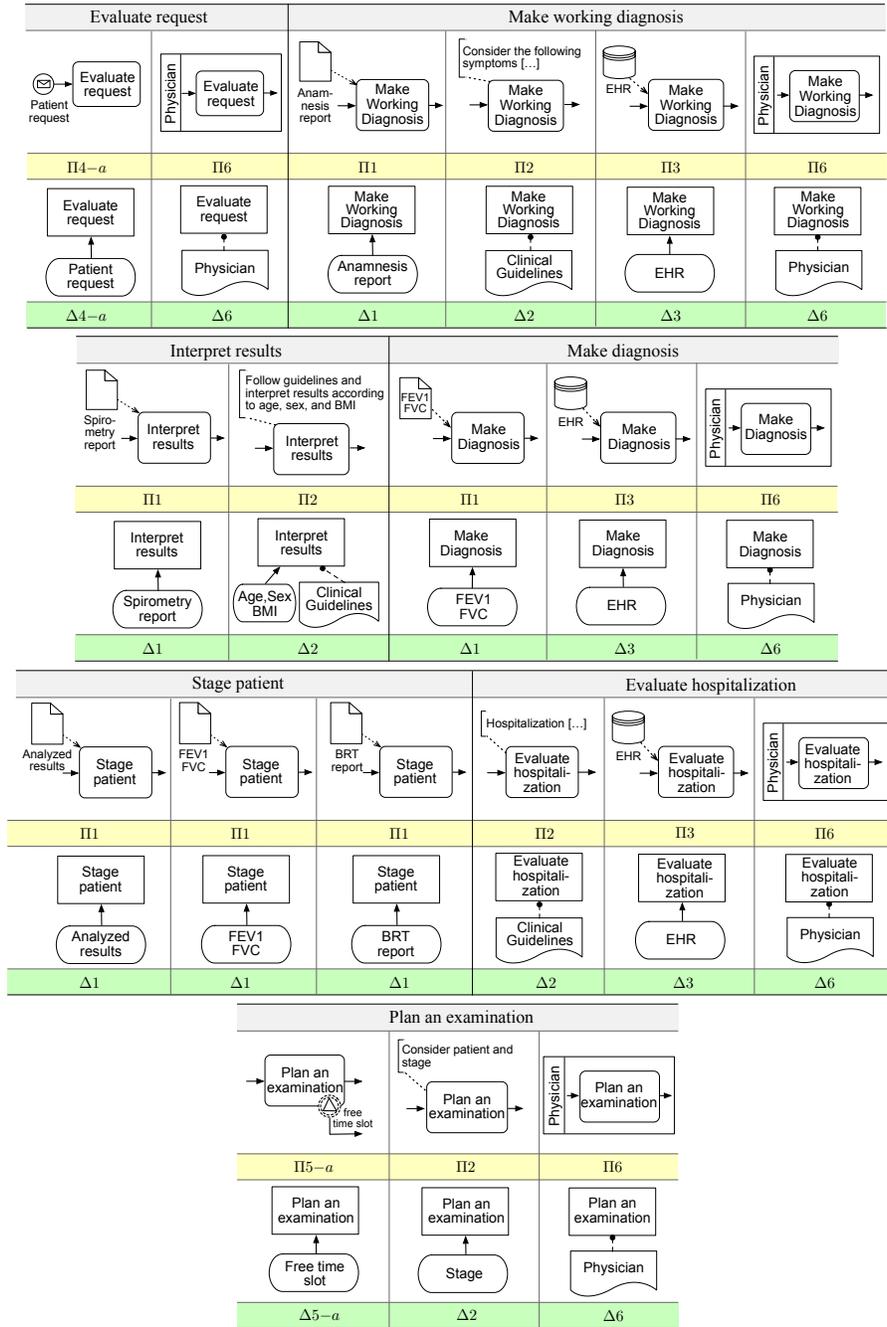


Fig. 11.6: Decision patterns extracted from the decision activities listed in Table 11.2 and corresponding DRD fragments, obtained by applying the mapping explained in Sect. 11.5.

In some cases, multiple combinations of the extracted DRD fragments are suitable to describe one scenario and it is up to decision designers to choose the more appropriate one, by following stakeholders indications.

Once the comprehensive DRD is constructed for a given decision model, the original process model may be adapted to improve the representation of data and decision activities. Usually process adaptation is carried out to reduce inconsistencies and improve the integration of process and decision models [50]. In the considered context, process adaptation involves mostly data representation. On the one hand, if some data are used exclusively for decision-making and are already captured by the newly designed DRD, an analyst may decide to remove them from the process to increase readability. On the other hand, missing data used as an input for decision-making shall be added to the process to ensure correct process and decision enactment [50]. For instance, decision outputs representing intermediate results needed by another decision activity shall be represented as data objects in the process model.

Compared to the misuse of control flow for representing decision-logic [20, 50], the removal of data from the process model is not always mandatory when dealing with data, as process-related data do not directly influence process execution and the same piece of information may represent different concerns in different models. For example, it is absolutely acceptable to have a data object representing both an input for a decision activity in a process model, and an input to a decision in a DRD [47]. Indeed, if not all required data are included in the process model, the decision activities requiring that data will not be executed properly [50]. Of course, designers may consult stakeholders to determine whether it is relevant to keep the data element that carries a decisional value in the process model (e.g., for documentation purposes to allow process users to consult the model and see which data is used there), or eliminate it from the process model (e.g., if a text annotation was misused to describe a decision rule which has become a business rule/knowledge source in the decision model obtained after the mapping). Lastly, process resources being also authorities for decisions should never be removed from process models, as they have a dual role, which is correctly captured by the integrated process and decision models.

Eventually, for the determined decision activities of the process model, undirected association links to the corresponding elements of the extracted DRD model should be added. Once decisions have been extracted from the process model, a decision engine can be employed to evaluate the decisions at runtime. Actually, we do not require a dedicated process engine, but could rely on an arbitrary implementation of the process.

Example 11.4. The DRD fragments discovered from the process of Fig. 11.5 were consequently combined together and compiled into two separate decision requirements diagrams, presented in Fig. 11.7. In order to link different decisions, we considered both the relationships between decision activities dictated by the process control flow [20] and the flow of information (i.e., input/output relations and access to shared data) that links such activities.

pitalization through a decision structure represented by exclusive gateway Is the patient already staged? immediately following decision activity Evaluate request.

The extracted DMN model in Fig. 11.7 serves as an explanatory decision model for the BPMN process model of Fig. 11.5, as it explicitly incorporates the process-related data used by decision activities for decision-making. Herewith, the extracted decision model can be executed complementarily to the process model, and thus, the principle of separation of concerns [224] is observed. However, the original BPMN process model should contain the undirected association links between decision activities and the corresponding elements of the extracted DMN decision model, a step that can be done at the implementation level.

11.6.5 Empirical Findings

To gather empirical evidence supporting the reliability and applicability of the proposed pattern-based approach, we analyzed the relationship between process-related data and decisions in selected real-world process models.

We manually analyzed a repository of 43 process models addressing a complex procedure of liver transplantation [229]. This set of process models resulted from a collaborative design effort, involving both practitioners and process modeling experts, aimed to enable process monitoring and analysis.

The goal of our analysis was to quantify how often in practice the patterns described in Sect. 11.4 are used for capturing or supporting decision-making.

All patterns $\Pi 1$ – $\Pi 6$ were detected in the 43 process models, except for $\Pi 3$. The absence of pattern $\Pi 3$ can be explained by the fact that the domain experts involved in process design were hospital physicians who did not directly interact with IT-systems. Moreover, although a consistent amount of information was recorded in the hospital IT system, the connection between process and data management systems was not made explicit.

The most commonly detected patterns were $\Pi 2$, which was found in 44.19% of the process models, and $\Pi 1$, which was present in 39.53% of them. These results met our expectations, as the use of text annotations ($\Pi 2$) and data objects ($\Pi 1$) to describe input data for decision activities is quite common in health-care processes. Indeed, clinical decision are often based on medical knowledge and evidence, stored in patients health records and interpreted according to professionals' experience and expertise [58]. Having multiple and fragmented information sources to consider makes clinical decisions hard to be represented in process models. As a result, text annotations are often used to make BPMN processes easier to be understood by practitioners, who are used to read and interpret textual documents such as clinical guidelines.

Patterns $\Pi 5-a$ and $\Pi 5-b$ occurred quite frequently in the analyzed process models. In detail, $\Pi 5-a$ was detected in 25.58% of the repository process models, while $\Pi 5-b$ was found in 18.60% of them. However, during our analysis, we observed that boundary events were overused for modeling non-exceptional

control flows, probably instead of exclusive gateways. Therefore, the probability of finding $\Pi 5-b$ in other application domains may be lower.

Patterns $\Pi 4-a$, detected in 11.63% of the process models, and $\Pi 4-b$, found in 16.28% of them, were slightly less common. This can be explained due to the fact that start and intermediate events preceding decision activities, usually serve as triggers for decisions rather than bearing decision-related data.

Pattern $\Pi 6$ representing the involvement of a process resource as an authority for decision-making was detected in 11.63% of the process models. This percentage was lower than we expected. However, we discovered that in the studied context there was no differentiation among the resources entitled to execute each clinical pathway (i.e., process) and to make the related decisions. That is, since the correspondence resource-pathway was almost one-to-one, resources were not explicitly modeled as BPMN swimlanes and this explains why we could not detect $\Pi 6$ as often as expected.

After detecting the decision patterns, we applied the steps described in Sect. 11.6.2 – 11.6.4 to obtain the decision models related to the 43 processes. During this phase, we followed the principles outlined in [50] to gain a deeper understanding of decision model post-processing and process model re-design.

Overall, we discovered that several decisions were hidden in the analyzed process models. This fact may be explained by at least two reasons.

On the one hand, the process models were designed under time pressure in a series of workshops [229] and, thus, multiple aspects were incorporated within one (process) model. On the other hand, designers were not aware of the principle of separation of concerns, especially because process models were designed before the introduction of the DMN standard [47].

In addition, most of the detected decisions were based on process-related data originating from manual activities. Such data had been annotated on the process models to improve understanding and to serve as a basic reference for compliance analysis. As a result, the data-centric decision patterns occurred quite frequently in the considered repository and most of the identified process fragments were suitable to be externalized into dedicated decision models.

Overall, analyzing real-world process models allowed us to shed light on the features of process models that are a prerequisite for successfully applying the proposed pattern-based approach. We discuss them in Sect. 11.6.5, together with the strengths and limitations of our work.

The pattern-based approach proposed in this paper is grounded on the consolidated BPMN and DMN standards and partly complements previous research addressing the discovery of decisions from the process control-flow [20]. Indeed, by considering the data perspective of BPMN process models from a decision modeling standpoint, our approach contributes to providing analysts with complete overview of the decision-making carried out in a process.

The choice of relying on design patterns is convenient for two main reasons. Firstly, BPM researchers and practitioners are familiar with the use of patterns to support process technology [20, 70, 237]. Secondly, experts have noticed that there are often recognizable patterns in the decisions they make. This is partic-

ularly true for clinical and healthcare domains, where decision-making is often standardized for improving compliance with care protocols [58, 229, 238, 239].

In our case, patterns are useful to provide a conceptual basis of process and decision design. The decision patterns described in Section 11.4 capture high-level concepts that are commonly understood in business environments and, thus, they may also serve to ease communication among designers, analysts, and practitioners, and to encourage collaborative process and decision (re-)design. Last but not least, the proposed approach allows designers to focus on the extracted decisions that can be evaluated at run-time by a decision engine.

Being based on the data perspective of BPMN process models, our approach depends on the quality of the process models used as a starting point.

In particular, we assume that (i) process-related data have been modeled *explicitly* (i.e., by using process elements that can be visualized in a process model) and consistently, and that (ii) the decision activities based on such data are somehow distinguishable in the process model. Our first assumption stems from the fact that the use of data objects or text annotations in BPMN processes is considered good practice to improve process model understanding [50, 240, 241] and, thus, process models typically include them. We intentionally leave out non-visible elements and attributes, e.g., `InputSets` and `OutputSets` of activities [11].

The two assumptions mentioned above go hand in hand with the assumption that domain knowledge is available to designers and analysts. Typically, in real-world settings, domain experts have deep understanding of the parts of the process they perform, whereas designers often lack of domain knowledge [240].

However, the gathering of domain knowledge is usually carried out during process discovery [58, 240] and, thus, the knowledge related to existing process models is likely to be already available when our approach is applied. Besides, process and decision analysts may actively interact with stakeholders to gain a clear understanding of the application domain, identify decision activities, and ensure that process-related data are properly and explicitly modeled.

Since the scope of our patterns is centred around decision activities, which typically precede local “decision points” in process models [20], we intentionally leave out decisions spanning multiple processes. This point of view may be seen as a limitation of our approach, as the scope of the discovered decisions is restricted to the considered process model and, particularly, to its data perspective.

Moreover, when combining the obtained DRD fragments into a complete graph, intermediate decision results that were not visualized in the starting process model are likely to be missing in the decision model.

To be able to add them while maintaining process and decision model consistency, we must include all decision activity outputs in the process model. Forcing such outputs to be explicitly represented in process models may as well be perceived as a limitation. Yet, according to the principles for consistent process and decision model integration outlined in [50], all intermediate results necessary for correct process and decision enactment shall be represented in the process model.

All things considered, the proposed pattern-based approach considers a high level of abstraction. We focus on modeling the decision requirements derived from process models to identify the extent of the decision-making and the authorities involved in it. Then, we discuss the re-design of the process to ensure that the decision-making it coordinates is consistently specified.

The full specification of the decision logic is not aimed at. Consequently, we do not address the automation and implementation of the models into software components. For the same reason, we do not consider the decision-service layer, although it seems promising to automate decisions while keeping process and decision logic separate [227].

In light of the assumptions discussed above, the empirical findings presented in Section 11.6.5 have some generalization limitations.

Since the process models were designed for monitoring and analysis purposes, special attention was given to identify the data sources involved in the process at design time. Thus, most of the 43 processes we analyzed were rich in explicitly represented data. In addition, we had easy access to the domain knowledge needed to identify decision activities, thanks to the complete documentation that was gathered during process design, the availability of clinical guidelines, and the direct interaction with practitioners.

Both these aspects, which are prerequisites to apply the proposed pattern-based approach, are not generalizable to every process repository.

11.7 Concluding Remarks

In this chapter, we proposed a pattern-based approach to support process and decision analysts in unbundling decisions hidden in BPMN process models, focusing on the data perspective of the latter ones.

In a nutshell, we distinguished a set of decision patterns that characterize process-related data used for making decisions in existing process models.

The patterns were elicited by conducting a systematic qualitative analysis of the BPMN standard [11] and by considering previous work on the suitability of BPMN for modeling decisions and (related) data [12, 70, 230]. Then, we provided a mapping of such patterns onto corresponding DRD fragments. Finally, we discussed the derivation of a comprehensive DRG from the obtained fragments and considered process model re-design to make an effective use of the data needed for decision-making and to ensure process and decision model consistency [50].

Previous research considered the extraction of decisions from the process control flow [48, 20] or focused on the decision services layer to separate process and decision logics [227, 228]. By focusing on the extraction of DMN models starting from the data perspective of BPMN process models, the presented approach contributes to enrich the stream of proposals aimed to improve the separated yet integrated use of the BPMN and DMN standards.

A Methodological Framework for the Integrated Design of Decision-intensive Care Pathways

This chapter is based on results published in [59, 58].

Healthcare systems are among the most complex and challenging organizational environments [26]. Healthcare organizations are becoming more and more fragmented into specialized and integrated functional units, which cooperate and exchange information to provide primary, secondary, and specialist care.

They operate by enacting a wide range of processes with different characteristics and requirements, spanning from those capturing clinical procedures, such as diagnosis and treatment, to those representing organizational and administrative tasks, such as examination planning and patient transfer [76]. Yet, clinical and organizational processes are not independent, but rather complementary and intertwined.

In the past decades, care pathways have received increasing attention and have often been used as an instrument for the reorganization of clinical processes. Care pathways are perceived as tools for quality assurance, process optimization, benchmarking, and cost analysis [242] and have established themselves as an effective method of reorganization of medical practice in a process-oriented way [113].

Care pathways are developed and utilized by multidisciplinary teams of clinicians, case managers, nurses, pharmacists, and physiotherapists for local use inside organizations, as well as a roadmap by patients and their relatives [113]. Clinical pathways include both organizational and clinical tasks, and address various aims, by capturing different aspects of clinical care.

However, the main challenge related to their creation and maintenance is the selection of an appropriate modeling method. Indeed, the computer-based design and management of care pathways needs to be supported by methodologies and technological tools that enable understanding, standardization, and sharing of the underlying care process.

Similar needs have emerged in the field of healthcare process management [26, 51, 76], where process standardization has proven effective to reduce

the occurrence of exceptional events that would compromise the quality and the efficiency of both organizational and clinical outcomes.

Among others, business process systems emerged as a leading technology also in the healthcare domain [59, 76, 53, 162, 243, 244]. In particular, the BPMN standard meets the requirements of comprehensibility, standardization, and tool support required to design care pathways.

BPMN diagrams are suitable to design, visualize, and document processes at different levels of abstraction and according to different perspectives (process structure, interaction between processes, and so on), thus meeting the needs of various categories of users and fostering both process implementation and re-engineering. BPMN is currently supported by numerous process engines [24, 94, 25] that allow stakeholders to actively monitor executable process at run-time and support the interaction with knowledge workers through domain-specific client and invoked applications.

In addition, BPM systems provide rich development environments that integrate design with process simulation, which encompasses techniques used to walk through a process in a step-by-step manner in order to check whether the process actually behaves as desired [1]. In BPM, simulation is a commonly used approach for quantitative process analysis [240], as it enables the identification of shortcomings and procedural bottlenecks by allowing process managers to assess quality, check conformance, and to progressively tune organizational outcomes.

Recently, the need for a separation of concerns [46] has prompted the introduction of the DMN standard [47]. DMN diagrams are meant to model decisions by representing input information, the used knowledge models, and their requirements for decision logic. Such explanatory diagrams portray the structure of one or more decisions and can be linked to one or more processes or activities, depending on the scope of decision-making. Thus, coupling BPMN processes and DMN diagrams could be beneficial to support the integrated design of decision-intensive care pathways, where complex clinical and organizational knowledge for decision-making needs to be represented.

Despite the presence of such advanced models and software tools, at the best of our knowledge, only few attempts have been made to manage care pathways through the combined use of BPMN and DMN, and no attention has been paid to propose/extend design methodologies for the specific context of care pathways.

In this chapter, we propose a new methodological framework to support the integrated design, implementation, and enactment of decision-intensive care pathways under an organization standpoint. Our goal is to foster the modeling and re-engineering of care pathways and related decision-making through proper information management and data re-use.

The methodology defines both the main phases of process development, which are based on BPM best practices [1], and the widely known and adopted modeling languages and tools to be used. In detail, we show how DMN can complement BPMN with the goal of increasing support towards decision-making, hence improving its suitability for healthcare process modeling and enactment. For each

phase, relevant information sources are also highlighted. Within the proposed methodology, we will mainly focus on the design phase.

The main novelties, specific for the management of care pathways, of the proposed methodological approach are:

- supporting the integrated design of care pathways and decisions, based on conjunct use of the BPMN and DMN standards;
- strongly integrating care pathway simulation within the design phase, to support agile development;
- proposing a classification of the main kinds of information involved in different phases of process and decision development, and investigating the role of data-based indicators;
- using and possibly extending BPMN modeling constructs to represent complex temporal behaviors/ constraints in care pathways;
- illustrating meaningful aspects of clinical decision-making and evaluating their scope and impact on the overall care pathway.

Moreover, as a further added value, we show a real-world application of the proposed approach to the design of care pathways for Chronic Obstructive Pulmonary Disease (COPD), within a regional context.

This chapter illustrates a general methodology that encompasses all the phases of a typical business process life-cycle, thus considering also process implementation and enactment. We highlight possible connection points with existing formalisms (e.g., BPMN, DMN), as the methodology aims to provide a main standard core that can be extended and integrated with context-specific decision support systems and tools, suitable for dealing with specific clinical aspects.

We also include information management in the methodological approach and present a preliminary classification of data relevant for process execution.

As for the application to the clinical domain of COPD, we aim to illustrate an approach for process and decision design tailored for care pathways covering all the aspects related to requirements analysis, modeling, and simulation.

In detail, primary, specialist, and secondary care of COPD have been studied and modeled exhaustively, with the help of stakeholders. The presented BPMN and DMN diagrams describe the initial diagnosis and assessment of COPD patients, ordinary care, and extraordinary clinical interventions. Finally, real data have been obtained from currently enacted healthcare processes and they have been used for process simulation in order to analyze as-is organizational procedures from a re-engineering perspective.

The remainder of the chapter is organized as follows. Sect. 12.1 discusses the proposed methodological framework. Sect. 12.2 explains important aspects of COPD treatment and management and illustrates how the introduced methodology is applied to the design and management of care pathways for COPD in the region of Veneto, in northern Italy.

12.1 The Methodological Framework

In this section, we present a new methodological framework for supporting the seamless design, implementation, and enactment of healthcare processes and decisions. We also tackle a few crucial aspects related to the management of information relevant for the execution of both processes and decisions.

In particular, the methodology provides both a classification of the disparate facets of process-related decision-making and a qualitative analysis of the kinds of data involved in the various phases of process development.

12.1.1 Overview

The proposed methodological framework, which extends the methodology introduced in [59], is a general-purpose approach grounded on standard BPM models and tools, but it is presented with special attention to the requirements related to the management of care pathways. In particular, we present a structured approach to delineate the main phases of care pathway development and, for each phase, we define which are the techniques and tools to be used and how.

In addition, we provide the reader with an overview of which are the most important challenges that can be encountered in each phase. Finally, we suggest possible connection points with existing formalisms and results presented in previous chapters of this thesis suitable for healthcare domains.

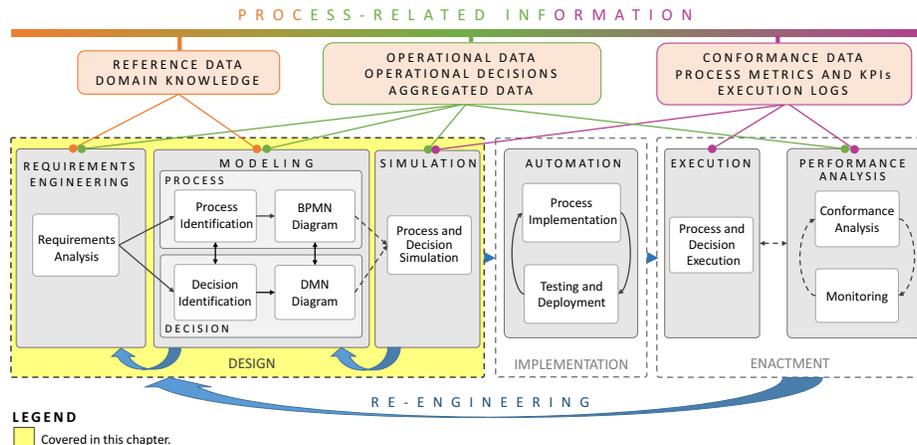


Fig. 12.1: Main steps of the proposed methodological framework. The Design phase has been exhaustively applied to the COPD case study presented in this thesis, as detailed in Sect. 12.2.

The viewpoint of the methodology is intentionally process-oriented, as the macro-steps portrayed in the central part of Fig. 12.1 correspond to specific

phases of the business process life-cycle. Namely, we considered DESIGN, IMPLEMENTATION, and ENACTMENT, the latter one encompassing also performance analysis, which is typically depicted as a separate EVALUATION phase [1].

These phases constitute the core of the approach and are connected to process-related information, which is outlined at the top of Fig. 12.1. The mentioned life-cycle phases are framed by a dashed line to highlight that there is an open interaction and a continuous exchange of knowledge and information between the phases of process development and the organizational environment. The advancement from one phase to the next one is shown in Fig. 12.1 by small light-blue triangles.

In real application environments it is common to deal with incremental process development, followed by iterative refinements, and with agile approaches, i.e., non-completely planned and heavily involving final users in the related software development [245]. For this reason, the depicted life-cycle has to be considered a continuous one. That is, the output of the enactment phase is re-used as an input for the design phase. This iterative re-thinking and organizational change of both the structure and the outcomes of a process is often referred to as process re-engineering.

Business process re-engineering is commonly defined as the “radical re-design of one or more aspects of a business process”, aimed to improve process performance and to implement organizational changes [246]. In Fig. 12.1, the RE-ENGINEERING of both processes and decision-making is represented by thick curved light-blue arrows. A big re-engineering arrow links the enactment and design phases, denoting the need for process and decision re-design, based on information retrieved from process execution and analysis. Similarly, two smaller curved arrows are used to represent local re-engineering steps, which refine the design of process and decision diagrams, based either on insights provided by simulation or on new requirements. In healthcare domains process change is often needed to improve compliance to standards statements or to integrate health information systems [247].

In the remainder of this chapter, we show how the presented methodological approach for care pathways design and enactment, even if applied partially, allows IT experts and care providers to interact and address some complex challenges [26, 51, 75]. The methodology is directed to expert process modelers. That is, we assume that a team of process and IT experts, actively interacts with care providers and clinicians to deeply understand and analyze the healthcare context. Indeed, as process development is a time-consuming task requiring IT and technical expertise, we do not expect clinicians to directly model and maintain the given care pathway.

In Tables 12.1 and 12.2, we exemplify selected important questions that are asked to stakeholders for eliciting the requirements needed for each healthcare process development step.

M. P.	Phase	Step	Design Questions
DESIGN	REQUIREMENTS ENGINEERING	Requirements Elicitation and Analysis	<p>How does the healthcare organization operate? Which are the final goals (clinical ones? related to healthcare policies? related to monitoring quality of care?) and which is their value? Who are the clinical/healthcare final users of the modeled care pathway? Are there healthcare policies or documents to follow?</p>
	MODELING	Process Identification BPMN Diagram	<p>Which and how many processes can be discerned? Are the processes covering clinical or healthcare-organizational aspects? Which is the most suitable modeling granularity for care providers? Which are the main process aspects and how can they be represented (roles, activities, etc.)? Are there recurrent/known patterns? [237] Which are the clinical/healthcare data that need to be available for the process to be executed? Where are they stored? How are they exchanged? Does the model require execution details?</p>
		Decision Identification	<p>Which and how many decisions can be discerned? Are decisions operational or strategic (i.e., strongly based on aggregated medical data)? Do decisions influence the process flow? How? Which data requirements are needed for decision-making? Are these data coming from EHR/EMRs? Are they specific to this activity? Who is responsible for making the decision? Does the decision need to be automated or is it made by humans? Do clinicians require direct support for this decision?</p>
	SIMULATION	Process and Decision Simulation	<p>Is the process behaving as expected? Which are the costs and the duration of the process? Are the bottlenecks related to bad modeling or resource consumption? What aspects of the process need improvement? May novel indicators improve healthcare process monitoring?</p>

Table 12.1: Part I - Summary of important questions that should be answered during each phase of the methodology shown in Fig. 12.1. **M.P.** stands for “Main Phase”.

Main Phase	Phase	Step	Design Questions
IMPLEMENTATION	AUTOMATION	Process Implementation	How is the process enriched with information for execution? How is the graphical model translated/enriched? Are there organization constraints on execution?
		Testing and Deployment	Does the engine have access to all services, client, and invoked application needed by the process? How does the process interact with other systems? How are instances grouped or synchronized (e.g., by patient, physician, and so on)?
ENACTMENT	EXECUTION	Process and Decision Execution	Who executes process tasks? Which execution data are relevant for performance analysis?
	PERFORMANCE ANALYSIS	Conformance Analysis Monitoring	Does the process conform to the expected objectives? Are there process instances that cause bottlenecks? Which is the current status of the process? Are there unexpected shortages of resources? Is there the need of sending alerts or managing exceptions?

Table 12.2: Part II - Summary of important questions that should be answered during each phase of the methodology shown in Fig. 12.1.

12.1.2 Design: modeling processes and decisions

Reasonably, the development of decision-intensive healthcare processes begins with a DESIGN phase. This is organized into three main steps: REQUIREMENTS ENGINEERING, MODELING, and SIMULATION, depicted in Fig. 12.1.

During Requirements Analysis a specific organizational problem is studied and process participants, final objectives, and performance goals are defined. The latter ones are usually determined with help of indicators, established by teams of clinicians and healthcare managers who meet regularly to assess process performance. Among the indicators to consider in care pathways, in recent years healthcare quality indicatorss (HCQIs) have been proposed by the member countries of the Organisation for Economic Co-operation and Development, as a way to assess, monitor, and compare the quality of healthcare systems both at regional and national level [248]. As an example, cancer mortality rates and the number admissions for asthma are quite traditional quality indicators, to consider in related pathways.

Requirements engineering usually comprehends feasibility studies and the analysis, definition, and specification of requirements, aimed to understand the studied domain and the role of the modeled pathway in the organizational context. In the healthcare domain, process and decision requirements related to care pathways stem either from organizational documents, medical literature, clinical guidelines, and local care policies, or they are collected by process modelers through direct interaction with both medical professionals and care managers. Usually interviews with participants are conducted and modelers are invited to observe how the organization operates (see the questions reported in Table 12.1 as a trace of possible interview).

MODELING embraces processes, data, and decisions, based on the previously identified requirements.

Process modeling allows one to filter out the irrelevant complexities of a real scenario, while providing a picture of the considered healthcare environment, tailored to spot improvements needed for re-engineering. Process modeling begins with Process Identification, which focuses on establishing which are the main process activities, their execution order, and the chosen modeling granularity. During process identification, the modeler decides which and how many processes should be outlined, considering final objectives and possible implementation requirements.

Besides, it is crucial to ensure that the outlined processes do not overlap and that the interactions between resources correspond to reality. To this end, BPMN choreography and collaboration diagrams can be employed to explicitly represent message-based communication between multiple processes and resources [11].

Usually, at this point, process elements are defined at a quite abstract level. Step BPMN Diagram deals with the design of a more detailed process diagram. BPMN diagrams are suitable to represent the process control flow with sufficient detail and expressiveness, at different degrees of structural complexity and support for automation [1]. In addition, BPMN partially tackles other aspects of

process modeling, such as the exchange of data produced and consumed during process execution and the management of process roles and their interactions. In particular, BPMN allows the designer to specify different roles within a health-care organization and to outline which activities are performed by different roles through *pools* and *swimlanes*.

During this phase, process designers adapt the process model to better suit the final design goals and to meet the local requirements of the application context. Designers can exploit existing patterns for modeling control flow [237], data [249], and time [17], to capture complex process aspects.

Similarly, decisions play a crucial role in process development and enactment, and, thus, it is important to understand how they affect both the process control flow, the related data, and the organizational environment. Hence, decision modeling is often carried out in conjunction with process modeling.

Decision Identification deals with the discovery of which are the decisions that affect the process and its organizational landscape: During this phase, designers need to understand which are the relationships between decision-making and the process flow, and the repercussions of decision outputs on the process flow.

A decision can be modeled by means of a DMN Diagram, which addresses different aspects of decision-making. In particular, DRDs represent the internal structure of a decision and its relationships with other decisional steps. Decision logic can also be specified through boxed expressions or decision tables.

Whenever decisions are explicitly represented in a BPMN model, a DMN diagram can be associated to those process activities that encompass decision-making. In Fig. 12.1, this connection is represented through a simple double-headed arrow that links BPMN Diagram to DMN Diagram. However, according to the principle of separation of concerns [46, 48], it is beneficial to keep process and decision logics separate, as these are usually supported by different stakeholders, having distinct goals and expertise.

In this chapter, we distinguish two main types of decision.

- *Operational decisions* are usually expressed quantitatively, in terms of process variables that are used by process gateways to determine which path has to be chosen. Usually, operational decisions are represented in BPMN as decision tasks (i.e., a task of type user or business rule), whose result is used by a following gateway [20]. In the healthcare domain, operational decisions concern daily activities, such as drug dosing, identification of a specific vital parameter to measure, and scheduling of examinations.
- *Non-operational decisions* include more complex and strategic decision-making, usually affecting high level process objectives [224]. In healthcare, such decisions are often expressed in the form of natural language statements, based on medical evidence or professionals' knowledge, and rely on qualitative assessment. In a BPMN diagram, such decisions can be represented by a single decisional task, or they can span over more process activities. Examples of complex non-operational decision-making are clinical diagnosis

and therapy planning, taking into account different kinds of knowledge and, possibly, sub-decisions.

Usually DMN is used for supporting operational decisions, and patterns have been proposed to ease the extraction of such decisions from the process control-flow [20] and data-flow [60] (see Chapter 11).

As for decision logic, DMN supports the representation of decision tables and logic formalisms that can be easily mapped to the Friendly Enough Expression Language [47]. Decision tables can be constructed by using almost any principle, for instance by using the approach presented in [250] when decisions need to be extracted from textual guidelines. In addition, algorithms for analyzing the completeness, consistency and non-redundancy of DMN decision tables have been proposed in [251], and tools such as Signavio [24] currently support the verification of completeness for decision tables containing numerical entries.

Moreover, DRDs can be useful to provide a compact and explanatory representation of non-operational decisions. In detail, DRDs can be used for decision compliance and management, as they allow managers to understand which data must be used for decision-making, which are the required sub-decisions to be made, and who is responsible for them.

As third and final step of the design phase, in Fig. 12.1 **Process and Decision Simulation** is connected to process and decision modeling by means of dashed arrows, meaning that, despite being suitable for the design of care pathways, useful, and cost-saving (as discussed in detail in Sect. 12.1.4), it does not constitute a mandatory design step.

12.1.3 Dealing with process-related information and knowledge

PROCESS-RELATED INFORMATION is represented at the top of Fig. 12.1, and the relationships with the introduced phases of process and decision development are highlighted by solid lines, depicted with a rounded arrowhead.

During requirements engineering and modeling, DOMAIN KNOWLEDGE is used to provide a clear description of the relevant aspects of the modeled reality, by considering peculiarities, critical aspects, and goals of a specific health-care organization. For instance, a care pathway for treating patients affected by pneumonia must deal with medical knowledge related to the management and treatment of pulmonary diseases, bearing in mind that we are interested in data regarding the organizational aspects of the treatment rather than in clinical knowledge. Similarly, REFERENCE DATA help to delineate the requirements for process compliance. Examples of such data are reference values for medical parameters, input values for medical instrumentation, as well as administrative data regarding personnel salaries and skills, required duration of certain procedures, and quality standards. Reference data are strongly related to domain knowledge and local organization policies.

Then, OPERATIONAL DATA regarding process activities, roles, and the organizational structure are also gathered during requirements analysis. During

process and decision modeling, input and output data for process activities and decisions must be identified, and their structure, usage, and availability must be defined. In general, operational data produced, consumed and exchanged by process activities must be considered during the design of a BPMN diagram, either to set activity attributes or to define data artifacts.

During process design, data-related aspects are usually expressed through data objects and data stores. As BPMN is closely related to UML, some extensions such as the Activity View introduced in Chapter 7 can be adopted to enrich the process model by defining which classes of a UML data schema are touched by data operations required by process activities. By adopting this solution, it is possible to represent the direct connection that exists between the conceptual model of the process and the conceptual schema of the accessed database [57].

In healthcare, operational data are typically stored within health information systems. Similarly, OPERATIONAL DECISIONS are used in day-to-day operations and must be defined to ensure that their input and output data are used properly within the process. Process and decision modeling deals also with non-operational data that are used for decision support. This information, often referred to as AGGREGATED DATA, can be used to measure the performance of the process within a specific application context and to make strategic decisions during process execution. In healthcare, aggregated data are extracted from data warehouses or decision support systems, to be used during process modeling and re-engineering. A subset of both operational and non-operational data, used to identify which are the aspects of a process model that are interesting to be analyzed during process execution, is represented by CONFORMANCE DATA.

Finally, PROCESS METRICS AND KPIs are used to quantify process and decision quality and productivity. Data-based key performance indicators (KPIs), are measurable values that assess the progress of care pathways towards the achievement of the organizational objectives.

In BPM, Key Performance Indicators (KPIs) are defined to monitor process execution, evaluate the quality of process and decision outcomes, and quantify the overall progress towards organizational goal achievement [240].

In this work, we discerned two kinds of KPIs depending on their scope.

- *Local KPIs* correspond to process and decision metrics and are mostly used to measure activity execution times, costs, and resource consumption. We refer to such indicators as “local”, as they are extracted/used directly from/by multiple instances of one process model. As an example, the expected mean time duration to complete a care pathway related to the management of patients with breast cancer requiring a following chemotherapy is a local KPI, as it is related to the specific pathways and to possible “local” features of patients, reacting in different ways to the given chemotherapies.
- *Global KPIs* are used to evaluate the strategic value of the performed (decision) activities both quantitatively and qualitatively. Global KPIs can be directly re-used in clinical decision modeling to improve decision quality, pa-

tient prognosis, and predictive analytics. Usually, global KPIs rely on aggregated data and they impact several activities and processes. In addition, they can be used for organizational and re-engineering purposes that go beyond the scope of the defined processes. These indicators are rather based on clinical and healthcare data, and provide a summarized and focused overview of the modeled domain. Global KPIs can have both a clinical or organizational character.

Continuing with the example related to the oncological pathway, a global KPI could be the ratio between the number of patients concluding in a successful way the pathway and those who stopped the pathway for some health related issue. Such kind of global KPI could be possibly used inside the pathway to finely tune chemotherapy cycles as well as outside the pathway to activate, for example, suitable prevention actions.

As already underlined pathway KPIs are strictly related to Healthcare Quality Indicators, as we will show in the application of the proposed methodology to COPD patient management. Indeed, HCQIs may correspond to some measure used to structure and evaluate the considered pathway.

As an example, the outcome indicator coded as *13a* “The proportion of patients with estrogen receptor negative invasive carcinoma who received adjuvant chemotherapy, out of the total number of patients with the same diagnosis” is an acknowledged quality measure of the process related to breast cancer treatment [252]. This value could be both a result of a general care pathway representing the overall management of patients with breast cancer and an indicator used inside a more specific pathway for managing chemotherapies, to estimate the percentage of patients that have to be considered for the given pathway.

Although care pathway KPIs are used during process analysis or SIMULATION, they are defined during Requirements Analysis and Process Identification, in order to better tailor the process model to the final goals.

12.1.4 Integrating simulation in the design of care pathways

Process simulation is one of the most popular and widely used technique for quantitatively analyzing process models [240]. Simulation encompasses both the execution of a model under hypothetical conditions in a defined simulation environment and the analysis of the execution output [253].

During process design, SIMULATION can be used to support the quantitative analysis of pathway models. Specifically, Process and Decision Simulation promotes the study of the execution of modeled pathways and decisions in a realistic or ideal scenario, without resorting to pathway and decision implementation.

Being faster and more flexible than process enactment, simulation tools and techniques foster care pathway validation and provide reliable estimates of resource consumption and execution times. Indeed, process configuration and enactment may be highly demanding, while stakeholders are interested in obtaining

prompt and reliable estimates of process behavior, associated costs, required resources, and quality of outcomes.

Simulation allows one to validate a process behavior, by spotting bottlenecks, and by evaluating resources costs and consumption. Besides, the duration of the overall process and of single activities can be estimated and ideal or worst-case scenarios can be studied.

Decision simulation is often used to reproduce the outcomes of decision tables given an arbitrary set of input values and to verify table completeness. As previously mentioned, simulation results can be re-used to promote process adaptation to expected performance requirements. Moreover, simulation can be used for what-if analysis to explore alternatives, eliminate inadequate re-engineering proposals, and refine current as-is procedures.

Beside the continuous support to designers during care pathway modeling, in the proposed methodology we underline the importance of simulation for stakeholders who are responsible for decision-making within the healthcare organization. Indeed, simulation supports healthcare decision-makers in *resource allocation and management*. Through simulation, decision-makers can verify whether the resources associated to a care pathway are properly considered with respect to the estimated number of patients and the specified goals for the care pathway (e.g., reach the end of the treatment within 30 days since the admission of the patient).

Both local and global pathway KPIs could be considered together with HCQIs to see their mutual interactions in a “what-if” analysis context.

As an example, in the chemotherapy domain, the previously mentioned indicator *13a* is expected to be at least 80% [252]. This value could be used to distinguish the percentage of patients following a path of activities related to chemotherapies from other patients receiving different treatments. When simulating the same care pathway, one could change the value of *13a* to the target value of 90% in order to monitor the effects of such change on the local KPIs of the pathway, such as its overall cost or its average duration.

Moreover, simulation can provide healthcare decision makers with some evaluation of the HCQIs related to the considered pathway. As an example, in the design of a care pathway for patients with acute stroke, we could consider whether the HCQI *Acute stroke mortality rate* is maintained low, either according to predicted patients features or to the expected results of the designed process activities.

In this way, simulation can be used to estimate how HCQIs change with respect to different scenarios. Finally, simulation results can also feed some indicators that either are employed in external processes or support strategic decision making, or are used to obtain more complex HCQIs.

12.1.5 Implementation and Enactment

Following design, the methodological framework deals with IMPLEMENTATION, which can be realized in different ways, depending on which IT infrastructures are already available in the organizational environment.

Implementation is realized in different steps, usually starting from the automation of single processes and, then, by realizing the coordination between them. When starting from a single process, AUTOMATION is usually the first step that is applied for implementation. Automation focuses on the selection of software and services for process execution, encompassing both implementation and deployment aspects [1].

In Fig. 12.1, by Process Implementation we intend mostly process configuration, that is, the integration of a process model with technical information that facilitates enactment. Then, the enriched BPMN diagram is mapped into a formalism suitable for execution. Processes are executed by engines, which link client applications, support the distribution of work among different process actors, and coordinate activities. Most existing process engines support the direct deployment of standard XML process descriptions generated by modeling tools, without the need of further mappings.

Process automation is closely related to Testing and Deployment. Traditional testing techniques from software engineering can be used to test the automated process, to see if it behaves as expected. During this phase, integration and performance are tested for detecting potential run-time problems.

Finally, the system is deployed in its target environment and complementary actions, such as data migration and user-training may be executed. In general, one advantage of using standard modeling languages is that process engines can easily interpret them and, within healthcare organizations, such systems may be already used by the administrative staff.

The final phase of the introduced methodological framework deals with ENACTMENT, which consists of EXECUTION and PERFORMANCE ANALYSIS.

In detail, Process and Decision Execution encompasses the actual run time of a decision-intensive process. This is usually the result of the process engine interacting with client applications. Execution constraints must be observed, and execution exceptions arising from unexpected occurrences must be properly handled. In this methodology, we do not explicitly deal with exception-handling, as this topic depends on the kind of system used for implementation and solutions have already been proposed in the BPM community [254, 255].

During and/or following process execution, performance analysis may be carried out to monitor enactment or to analyze how well the process is performing, compared to desired performance goals. In Fig. 12.1, the relationship between process execution and performance analysis is depicted by a double-headed dashed arrow, meaning that the latter may be optionally performed during or after process execution.

Monitoring allows one visualizing the current status of process and activity execution, in order to be able to enact corrective measures if the process is not be-

having as expected. During process monitoring, bottlenecks can also be detected. Analysis tools and algorithms can also be used to verify specific properties, such as done for the satisfaction of temporal constraints in [27].

Conformance Analysis is employed to verify the fitness and appropriateness of a process with a given pattern or organizational model, in order to check whether the process is well-incorporated the organizational practice. Both structural and behavioral aspects can be analyzed and the results of the analysis can be used for re-aligning the process model to reality. In general, multiple cycles of monitoring and conformance analysis are executed. In addition, formal models, such as Petri Nets, can be applied for analyzing process behavior [256].

During process enactment, conformance data and process metrics and KPIs are used to monitor the status of the running process and to analyze different executions. Usually, valuable execution data are gathered in the form of **EXECUTION LOGS**, that is, long files that contain information about events that have occurred during process enactment. Examples of such data are starting and ending times of activities, task duration, role assignments, and message exchanges. Execution logs are analyzed for process re-engineering and alignment, or for discovery and mining purposes.

12.1.6 Managing temporalities of care pathways

Even though a deep discussion related to the management of temporal issues in care pathways is not central to this chapter, we recall the main temporal issues related to the management of care pathways, considering the results presented in Part II of this thesis. Besides, we will show how the proposed methodology can be easily extended to manage such temporal aspects.

Temporal aspects of care pathways are usually complex and need to be suitably modeled, both in **REQUIREMENT**, and **MODELING** phases.

As an example, in modeling preliminary stages for the 6-Minute Walk Test introduced in Sect. 2.1.1, we need to represent that the patient has to rest for at least 10 minutes before executing the test. Further temporal constraints among different activities may also need to be represented. For instance, it could be interesting to show that the efficacy of a therapy must be evaluated two weeks after the treatment has been prescribed, as in the management of COPD that will be later discussed. In addition, periodicities need to be considered, since some care-related activities are inherently periodic, as, for example, the assessment of some chronic pathology or the administration of a given drug.

The representation of temporal constraints in BPM systems has been considered in literature, also considering clinical processes [30, 132, 125, 162, 152, 257]. In general, the focus of such proposals is both to (i) design temporal constraints in BP models and (ii) to verify some basic properties of temporal BP models.

In particular, the concept of *dynamic controllability* (cf. Chapter 5) has been introduced and extensively studied. Suitable algorithms have been proposed to verify at design time whether a process model can be executed while satisfying all the given temporal constraints and allowing activities performers to take any

possible duration within the range of the allowed ones [30, 162]. The proposed solutions can be applied also to suitably extended BPMN models, as described in Chapter 5. Thus, from this point of view, the proposed methodology is well suited to include approaches for checking temporal features of BP models.

In addition, the modular BPMN-based temporal patterns presented in Chapter 3 and Chapter 4 have been elicited from real clinical domains. The advantage of the BPMN patterns presented in Chapter 3 is that temporal aspects are suitably represented through a sub-process that can be edited by modelers and executed directly on a BPMN process engine. Thus, such patterns can be seamlessly and directly specified within BPMN care pathways and suitably handled by the proposed methodology.

Moving to the SIMULATION and EXECUTION of time-aware care pathways, two different aspects arise: The first one is related to updating temporal constraints according to previously executed process tasks; The second one deals with the runtime management of possible constraint violations and of the related reaction policies.

As shown in Sect. 3.6 for some basic patterns, designers can leverage exception handling of BPMN to specify suitable patterns for supporting different kinds of reactions to constraints violation [52, 53].

As an example, if a therapy duration exceeds its maximum duration, the model designer could specify either that such therapy must be immediately interrupted (strong policy) and some other compensation actions could be enacted or that the therapy will continue (weak policy), but some other actions have to be done (sending a warning to the physician) [53].

As a final comment, it is worth noticing that powerful solutions to support time-related aspects of care pathways through BPMN are mainly based on the use of events, event-related activities, and exception-handling mechanisms. As an example, timer events can be used to specify either a specific delay from their activation or a certain time-date or a specific period of time within the process model, to trigger some specific clinical activity or some conditional path in the control flow.

12.2 Design and simulation of care pathways for COPD

In this section, we apply the introduced methodology to a real healthcare scenario, in order to discuss modeling choices and design challenges, and share the benefits brought by the proposed approach in a real clinical context.

The methodology discussed in the previous section has been applied to support the design of diagnostic-therapeutic care pathways for the management of Chronic Obstructive Pulmonary Disease in the region of Veneto, in northern Italy. The process and decision diagrams have been designed to provide a compact representation of healthcare procedures aimed at easing the understanding and improvement of as-is healthcare pathways and of their organizational as-

pects. One of the main modeling goals was the detection of flaws in the care pathways that led to undesired, yet reducible healthcare expenses.

In this chapter we focus on care pathway design and simulation, and intentionally leave out the implementation and enactment phases.

Requirements have been collected by analyzing textual documents authored by the Regional Healthcare System of Veneto, including both regulatory standards and organizational plans, and clinical practice references.

In addition, we directly interacted with the regional board, composed of care managers and clinicians responsible for the actuation of the care pathways, and we were directly observing how care interventions have been carried out in a few regional hospitals. Finally, standard American and European guidelines for COPD management have been studied and used for process compliance [99, 258].

Among existing tools for process design and management, we chose the Signavio Business Transformation Platform [24], which integrates process modeling, decision design, and simulation features for cost analysis. The web-based Signavio Process Editor has been used for process design and simulation, as it guarantees full adherence to the BPMN standard and supports different kinds of process simulation and cost management on several user-defined scenarios. In addition, it provides several functionalities that ease collaborative design, diagram reviewing, and commenting. The Signavio Decision Manager has been chosen to model DMN diagrams and to simulate decision table execution.

In the remainder of this section, we introduce the modeled clinical domain in Sect. 12.2.1 and we explain the design of both processes and decisions with BPMN and DMN. Then, we discuss the re-use of information through indicators in Sect. 12.2.7 and present simulation results in Sect. 12.2.8.

12.2.1 Chronic Obstructive Pulmonary Disease

In this section we recall the main aspects of Chronic Obstructive Pulmonary Disease, which was previously introduced in Sect. 11.6.1.

The World’s Health Organization defines COPD as “a life-threatening lung disease characterized by chronic and not fully reversible obstruction of lung air-flow, which interferes with normal breathing” [259]. Despite being preventable and treatable, COPD global prevalence and socio-economic impact remain significantly high [260], thus making the disease one of the leading causes of disability and death in developed countries. Moreover, factors as general population aging, development of comorbidities, and misdiagnosis contribute to reinforce this trend [261].

Whereas COPD is often underdiagnosed in younger patients, as respiratory abnormalities become noticeable only when internal organ damage is already advanced [262], current diagnostic criteria may lead to over-diagnosis in the elderly [263]. For this reason, it is important to limit the variability of the current diagnostic process.

A major proportion of COPD burden is represented by exacerbations, i.e., acute and unexpected worsening of respiratory symptoms that require a change

in management. Besides, the development of cardiovascular and respiratory comorbidities influences the prognosis of patients with COPD, further worsening the burden of the disease and requiring physicians to base on differential diagnoses for the identification of appropriate treatments [262].

In order to improve the management of COPD, integrated diagnostic and therapeutic care pathways have been proposed at a regional scale, with the intent of improving the effectiveness of COPD treatment, the prevention of exacerbations, and the coordination between different care providers. The development of such care plans aims to standardize the critical steps of long-term patient treatment and the overall decision process. In particular, it is crucial to clarify which is the role of each involved resource, which are the responsibilities, and which is the optimal timing for intervention. Besides, clinical, social, educational, and organizational objectives are set by considering economical expenses and resource availability.

The main goal of COPD diagnostic and therapeutic care pathways is the improvement of primary care quality mostly by reducing costs, intended both as patient lives and economical burden, related to misdiagnosis and to the prescription of unnecessary medical exams and treatments. The definition of such plans also encompasses a set of indicators that are defined to evaluate disparate aspects of the healthcare process, such as its structure, clinical and organizational efficiency, performance, progress towards the set final objectives, and overall economic impact. Prevention measures, educational programs, and counseling services are also included in the care plan, with the aim of training patients to recognize symptoms worsening and to improve the self-management of the disease in home care settings.

12.2.2 BPMN-based Modeling of Care Pathways for COPD

In the assessment and management of patients with COPD, three main intervention steps have been discerned, namely (i) *COPD diagnosis and assessment*, (ii) *ordinary management of stable COPD*, and (iii) *management of COPD exacerbations*. These steps are described in medical literature [99, 258], and are also identified in the as-is processes that are currently applied in Veneto.

For each one of the aforementioned steps, a BPMN process model has been designed, following the principles of well-structured process design (cf. Sect. 2.1.3) to increase process model readability and reduce the probability of committing modeling errors [107, 264].

The three phases included in the studied care pathways for the management of COPD are depicted in the BPMN process of Fig. 12.2, each instance of which represents the treatment of a single patient. Each one of these aspects of COPD management is represented as a collapsed sub-process, which relates to the other process elements as described in the following paragraph.

The overall care process begins in primary care with the COPD Diagnosis and Assessment, during which practitioners have to detect the disease and classify the patient according to the most pertinent severity level. This assessment phase

is carried out only once and, then, any patient diagnosed with COPD enters a multidisciplinary program for monitoring and managing the disease, which is enacted periodically, based on the assessed COPD stage.

The studied healthcare context of Veneto provides a staging system based on three main levels of severity, which is slightly different from the one presented in [258]. In particular, all patients having a spirometry-related value $FEV_1 \leq 50\%$ are considered at high risk and belong to Stage 3 (i.e., there is no Stage 4). The general practitioner is in charge of COPD diagnosis and of monitoring and validating the decisions made by pulmonary specialists.

General and pulmonary examinations considered in the **Management of Stable COPD** are scheduled based on the severity of the disease. For this reason, the sub-process representing the ordinary management of COPD is executed periodically. In BPMN, this periodicity is expressed by setting a looping condition for the whole **Management of Stable COPD** sub-process. In Fig. 12.2, this is highlighted by a standard loop marker [11], depicted at the bottom of the sub-process.

However, during ordinary COPD management, exacerbations can occur. In the BPMN process of Fig. 12.2, the occurrence of an exacerbation is represented by means of non-interrupting boundary conditional event **Exacerbation Episode**, which is triggered when an exacerbation occurs.

Non-interrupting boundary events allow the activity to which they are attached to remain active and a new a token is generated to drive a new flow, parallel to the continuing execution of the activity.

Therefore, as event **Exacerbation Episode** is non-interrupting, **Management of COPD Exacerbations** starts while ordinary **Management of stable COPD** is still being performed. Exacerbations must be promptly treated in order to reduce the impact of the current worsening and prevent the development of future episodes. In general, the treatment of exacerbations does not affect the intervention planned for the ordinary management of COPD.

However, ordinary medical inspections must be re-scheduled whenever the stage of the disease changes, either due to the progression of the disease or to an exacerbation episode. In Fig. 12.2 this is represented by boundary interrupting signal event **Re-schedule**, which is triggered by a corresponding signal event thrown either within sub-process **Management of Stable COPD** or following conditional event **Stage changed**, which captures a change of stage due to an exacerbation. Signal events are used for broadcast communication and, in the depicted process, they represent the interaction between the two sub-processes **Management of Stable COPD** and **Management of COPD Exacerbations**. Throwing signal events are characterized by a filled triangle marker, whereas catching signal events have a unfilled maker of the same sort. Whenever signal event **Re-schedule** is caught, sub-process **Management of Stable COPD** is interrupted and restarted, according to the periodicity defined for the newly assessed stage. The end of the described macro-process is reached once all interventions for COPD management and exacerbations are concluded.

The BPMN diagrams representing the structure of the three sub-processes outlined in Fig. 12.2 are shown Fig. 12.3, 12.4, and 12.5.

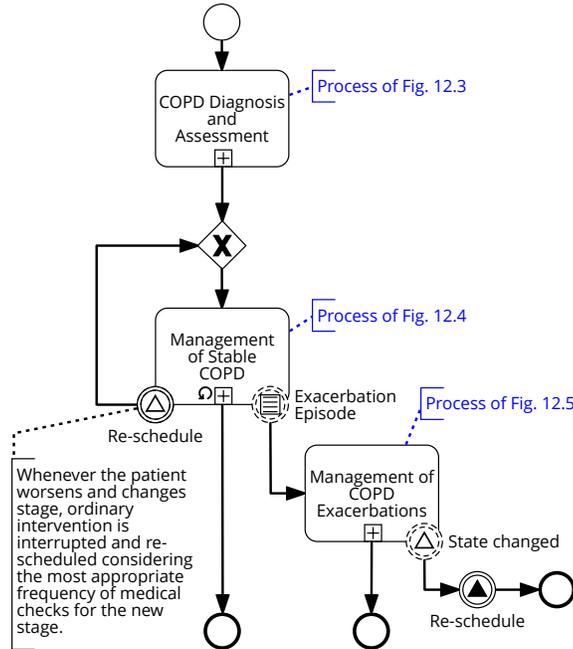


Fig. 12.2: BPMN diagram outlining the three main steps of the studied COPD care pathway, each one represented as a sub-process. Sub-process COPD Diagnosis and Assessment (shown in Fig. 12.3) is always executed first, followed by Management of Stable COPD (shown Fig. 12.4), which is performed periodically, according to the assessed stage. Ordinary examinations can be rescheduled whenever the COPD stage assigned to the patient changes due to the progressive worsening of the disease or following an exacerbation episode, the latter handled by sub-process Management of COPD Exacerbations (shown in Fig. 12.5).

In these process diagrams, different task types are used to denote decision tasks, i.e., process activities incorporating decision-making, which can be represented through a linked DMN decision model, at different levels of detail. We employ *user* tasks to denote activities that encompass human-decision making, whereas *business rule* tasks are used to highlight partially automated decision-making [47], as previously done in Chapter 5 and Chapter 11. All the other process activities are represented as *abstract* tasks or sub-processes, that is, generic activities whose behavior is not further specified [11].

In the BPMN diagrams, the recording of data to persistent databases is represented through data stores. For readability reasons, we used data objects to show only the exchange of information between different resources, while omitting details related to data flowing through activities executed by the same resource.

In particular, the presented processes need to communicate with the following information systems: The primary care information system, represented by data store **PCU db**, is the system used by general practitioner to store patient data – This system is usually shared among the practitioners that belong to the same PCU; The hospital information system, represented by data store **Hospital db**, stores different patient medical records; Finally, the health district authority information system, represented by data store **HC district db** is mostly used to record data regarding vaccinations and related campaigns. Nowadays, the three information systems are only partially connected, since data about drug prescriptions and examination scheduling are shared between PCUs and hospitals, but most data are still exchanged between professionals through reporting documents. The region of Veneto is currently working on developing a unified electronic health record that will allow practitioners, hospital personnel, health district authorities, and pharmacies to access the same data repositories.

12.2.3 COPD Diagnosis and Assessment

Diagnosis of COPD has to take into account a multiplicity of signs and symptoms. A clinical diagnosis of COPD should be considered in any patient who has dyspnea, chronic cough or sputum production, and presents a history of exposure to risk factors for the disease [258]. Among genetic disorders that must be considered in a diagnosis of COPD, the best documented is α_1 -antitrypsin deficiency, which leads to an increased predisposition to obstructive pulmonary disease [265].

Spirometry is required by international clinical guidelines to make a diagnosis of COPD [258, 99]. Practically, a spirometer measures how quickly full lungs can be emptied and the total volume of air expired. Spirometric measurements used in a diagnosis of COPD include:

- Forced Vital Capacity (FVC), which is the maximum volume of air that can be exhaled during a forced maneuver.
- Forced Expired Volume in one second (FEV_1), which is volume expired in the first second of maximal expiration, during a forced maneuver. This value measures how quickly lungs can be emptied.
- FEV_1/FVC , which is the ratio between the two previously calculated values. FEV_1 is expressed as a percentage of the FVC and gives an indication of airflow limitation, called Tiffenau index [258].

Spirometry results are evaluated by comparison with reference values calculated on healthy subjects and depending on age, height, gender, and ethnicity.

Currently, the spirometric criterion used for diagnosing COPD is a post-bronchodilator FEV_1/FVC ratio below 0.7. Bronchodilators are medications that increase the FEV_1 or change the other spirometric variables, by altering airway smooth muscle tone. Performing spirometry after the administration of a short-acting inhaled bronchodilator helps to differentiate between an acute or persistent obstructive defect and minimizes outcome variability [258]. However,

the aforementioned threshold is age-dependent and can lead to a significant degree of overdiagnosis in the elderly and of underdiagnosis in younger patients. Besides, as multiple respiratory diseases present a FEV_1/FVC ratio < 0.70 , differential diagnoses must be considered in order to determine an appropriate treatment [266]. Chest X-rays may be prescribed to confirm a working diagnosis of COPD and questionnaires such as the COPD Assessment Test (CAT) are administered for assessing symptoms. Indeed, when used in conjunction with spirometry results and demographic data, these tools have proven to enhance diagnostic accuracy [260].

For grading COPD severity, spirometric classification remains the preferred standard method [258]. Post-bronchodilator lung function, expressed in terms of FEV_1 and FEV_1/FVC ratio, is used to classify COPD patients into four levels of severity [258], summarized in Table 12.3. Patients showing a post-bronchodilator Tiffenau index greater than 0.70 are not considered in COPD management, despite those who smoke or have exposure to pollutants are considered at risk and may be targeted by prevention programs.

The BPMN process corresponding to *COPD diagnosis and assessment*, depicted in Fig. 12.3, consists of a main pool, which stands for the Regional Healthcare System. This is partitioned into two lanes, one representing a Primary Care Unit (PCU) and, specifically, a General Practitioner, the other one representing a Pulmonologist working in a Regional Hospital.

Since one of the goals of the COPD process models is to define the respective responsibilities of general practitioners and pulmonary specialists, only these two roles have been represented explicitly. Indeed, practitioners and pulmonologists have to supervise the performance of nurses, technicians and auxiliary personnel who collaborate in the process.

The process begins when the patient consults the general practitioner, typically after having experienced some sort of respiratory discomfort, such as productive cough, acute chest pain or breathlessness. *Case history and working diagnosis* is the first decision activity encountered, modeled as a user task, which represents the gathering of useful observations during medical inspection that are analyzed in conjunction with the patient clinical history and exposure to risk factors in order to formulate a working diagnosis of COPD.

Severity	FEV_1/FVC	FEV_1
Stage 1 (Mild)	< 0.70	$\geq 80\%$
Stage 2 (Moderate)	< 0.70	[50%, 80%)
Stage 3 (Severe)	< 0.70	[30%, 50%)
Stage 4 (Very Severe)	< 0.70	$< 30\%$

Table 12.3: Levels of COPD severity, based on spirometric assessment, adapted from [258].

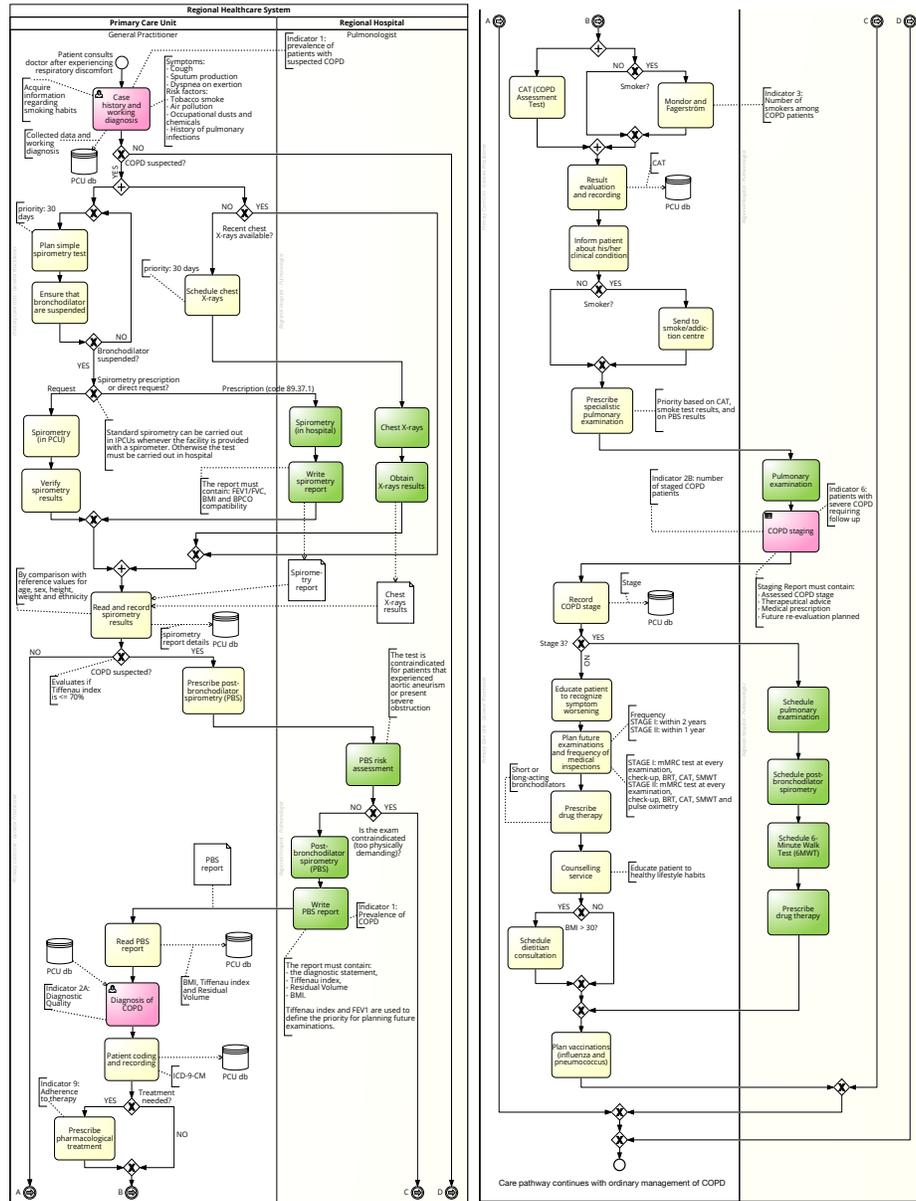


Fig. 12.3: BPMN process diagram for COPD diagnosis and assessment. The main decision activities identified with the help of experts are Case history and working diagnosis, which is a sub-step of decision Diagnosis of COPD, and COPD staging.

All the collected information regarding symptoms, smoking habits and risk factors is recorded in the local PCU database.

The following exclusive gateway, **COPD Suspected?** directs the flow towards the prescription of spirometry and further examinations, if a diagnosis of COPD is considered, or towards the process end event, otherwise. **Spirometry (in PCU)/Spirometry (in hospital)** is indicated to support COPD diagnosis, to assess the severity of airflow obstruction and, in some cases, it can be used as a prognostic indicator. If patients in primary care have direct access to spirometers, a simple spirometry test is requested and performed by a general practitioner directly within a primary care facility. Otherwise patients need a prescription for the spirometry to be performed in hospital.

Under an organization standpoint, the biggest difference between the performance of a spirometry in PCU or in hospital is waiting time. Whereas in PCU spirometric assessment can be executed during a routine examination, spirometers in hospitals are used for assessing various respiratory problems and patients are prioritized according to the severity of their conditions.

Chest X-rays may also be carried out in hospital, if believed necessary to support the diagnostic process. Spirometry allows measuring FEV_1 , FVC and the ratio between such values. The physician must **Read and record spirometry results** received with the corresponding reports, and compare them with reference values for age, weight, height, gender, and ethnicity. If airflow obstruction is detected, the practitioner must **Prescribe post-bronchodilator spirometry (PBS)** to confirm the diagnosis of COPD and to measure the degree of airflow obstruction and its reversibility.

Once the patient has been informed and advised about the conduct of the test, **Post-bronchodilator spirometry** is performed by a pulmonologist in hospital. Subsequently, the pulmonologist must **Write PBS report**, containing the diagnostic statement and the spirometric results. This is sent to the general practitioner, who must confirm the **diagnosis of COPD** and record the patient data in the primary care unit database, following the ICD-9-CM coding rules [267].

The general practitioner can also **Prescribe pharmacological treatment**, whenever considered necessary to relieve the patient from symptoms. Then, the **COPD Assessment Test (CAT)** is administered to gather information about the impact of COPD on the patient's quality of life and to measure general health status impairment. If the patient is a smoker, additional tests, i.e., **Mondor and Fagerstör**m, are administered to reveal addiction to tobacco consumption and nicotine [268].

Once the test-based assessment is concluded, the physician needs to **Inform the patient about his/her clinical condition** and can suggest a detoxing treatment in specialized facilities for smoking cessation. Then, the patient undergoes a **Pulmonary examination** in hospital, which results in **COPD staging**, based on the outcomes of the previously executed spirometric assessment. The staging report is authored by the pulmonologist and must contain the assessed COPD stage, the value of the Tiffenau index, an indication for medical examinations and future re-evaluations, and therapeutic advice. The general practitioner receiving the report, must verify that the assessed stage is correct before proceeding with

COPD stage recording as he/she is responsible for signaling possible incongruities. Indeed, the classification of COPD patients is a crucial decision as the planning of the future inspections, both in terms of the kind of medical analyses required and their frequency, depends on the severity of the assessed grade.

If the patient is found to be in Stage 3, a pulmonary specialist in hospital is put in charge of monitoring the patient. In this case, the pulmonologist must Schedule pulmonary examination, Schedule post-bronchodilator spirometry, Schedule 6-Minute Walk Test (6MWT), and Prescribe drug therapy to monitor the disease, and to evaluate the patient's functional exercise capacity.

Instead, if the patient is assessed with Stage 1 or Stage 2, PCUs remain in charge of COPD management. PCUs must also provide a Counseling service in order to educate patients to avoid exposure to behaviors that can further compromise their health condition. The progression of the disease can be limited by efficient monitoring and changes in lifestyle and smoking habits. In the final stage of COPD assessment, the general practitioner needs to Schedule dietitian consultation if the patient presents a BMI greater than 30. Finally, he/she has to Plan vaccinations (influenza and pneumococcus), as they appear to be effective in older and severe patients [258]. Once the main steps of the care intervention have been scheduled, the COPD diagnosis and assessment process terminates.

To wrap up, it is worth noticing that the process of Fig. 12.3 as some similarities with the one shown in Fig. 11.5. However, beside being less detailed, the process in Fig. 11.5 includes also the (collapsed) management of COPD exacerbations, and deals with presentation in hospital of patients experiencing acute symptoms, possibly due to an exacerbation.

The differences between the two processes are due to the fact that they represent care pathways carried out in different nations (i.e., Germany and Italy), thus being subjected to disparate regulations due to local arrangements and policies.

12.2.4 Ordinary Management of Stable COPD

Life-long and continuous management of stable COPD can become quite complex especially in advanced stages of the disease, as multiple complications have to be considered beside compromised pulmonary function. In particular, patients affected by COPD suffer from impaired gas exchange and are subjected to develop cardiovascular, metabolic, and neoplastic comorbidities [258].

Therefore, COPD treatment aims to prevent the progression of the disease, relieve patient symptoms, improve exercise tolerance and general health status, and prevent and treat correlated complications. The type of care workers in charge of COPD management and the frequency of medical inspections depend on the severity of the disease, on the patient individual response to pharmacological treatment, and on the healthcare system.

Ordinary management of COPD encompasses the following important aspects: smoking cessation, optimization of pulmonary status by pharmacological therapy, improvement of exercise tolerance, nutritional care, and possible sup-

port in terms of oxygen therapy or pulmonary ventilation. Besides, education, social, and behavioral aspects must be considered to improve compliance [99].

Vaccinations can be required, based on local policies, availability, and affordability [258]. Influenza and pneumococcal vaccinations are planned yearly, to prevent the insurgence of serious illnesses that may lead to COPD exacerbations.

Ordinary management of COPD is based on individualized assessment and aims to reduce both current symptoms and risk of sudden worsening [258]. As previously mentioned, the BPMN diagram representing the main steps of ordinary management of COPD, depicted in Fig. 12.4, is executed with a periodicity that depends on assessed COPD stage. This periodicity is captured by a start timer event. Specifically, the process is expected to start every two years for patients in Stage 1, every year for patients in Stage 2 and every six months or less for patients in Stage 3.

The first steps of the care process are executed either by the general practitioner, for patients in Stage 1 and 2, or by the pulmonologist, for patients in Stage 3, but the conduct of the explained actions coincides. Firstly, it is necessary to **Verify administered vaccinations**. In the studied context, vaccinations are managed by district health authorities. If vaccinations have not been administered, the physician must **Re-schedule vaccinations**. If the patient refuses to be vaccinated, the motivation behind the refusal must be recorded in the health authority information system HC district db. In the studied context, data suggest that elderly patients often forget to attend vaccination appointments or they have troubles reaching care facilities, due to mobility limitations. In this latter case, vaccinations can be provided at home, depending on local service availability.

To monitor the disease, care providers are required to **Administer COPD Assessment Test (CAT)** and to **Adjust current bronchodilator therapy**, if needed.

Pharmacological therapy is prescribed according to the severity of the illness, the availability of the drugs, and the presence of comorbidities. The treatment regimen should be patient-specific as the relationship between the severity of symptoms and airflow obstruction tends to vary from patient to patient. Moreover, COPD severity is also dependent on the geographical distribution of the population. Usually, patients in Stage 1 are treated with short-acting inhaled bronchodilators, whilst long-acting bronchodilators are prescribed to patients in Stage 2 or 3, in different dosage and combinations.

Then, different inspections are carried out in parallel, to evaluate the patient's health status, and re-assess BMI and respiratory function.

Mondor and Fagerstorm tests are administered to **Evaluate lifestyle and smoking habits** and to assess the patient's progress towards smoking cessation.

Physicians must also **Evaluate dyspnea and exercise tolerance**. The objective measurement of exercise impairment is obtained with the help of the **6-Minute Walk Test** [98]. This is used in conjunction with the Modified British Medical Research Council (mMRC) dyspnea questionnaire to quantify the degree of disability due to breathlessness during daily activities.

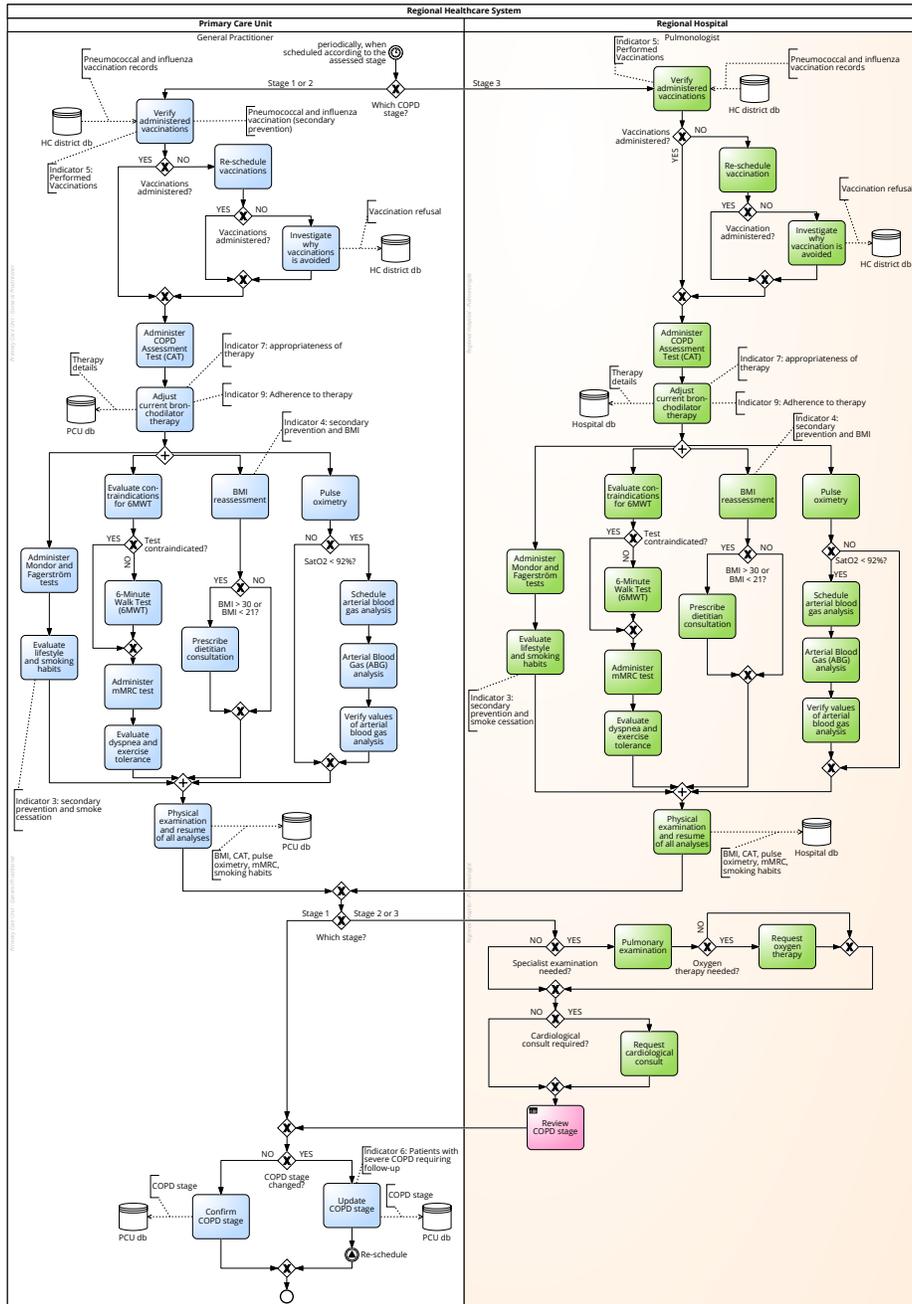


Fig. 12.4: BPMN process diagram for the ordinary management of stable COPD.

Body Mass Index (BMI) must also be re-assessed, in order to decide if it is necessary to **Prescribe a dietitian consultation**. Obesity and weight loss, when combined with muscle wasting, contribute significantly to morbidity, disability, and handicap in COPD patients, who must receive nutritional therapy.

Finally, **Pulse oximetry** is used to evaluate oxygen saturation. If the measured oxygen saturation is less than 92%, blood gas assessment is scheduled. **Arterial Blood Gas Analysis (ABG)** is the preferred method for determining the need of oxygen therapy [99]. Finally, the patient is examined and all analyses results are resumed and recorded in the dedicated information system PCU db.

If the patient is assessed with Stage 2 or Stage 3, he or she might necessitate an additional pulmonary inspection. In this case, a pulmonologist can **Request Oxygen Therapy** according to the considered clinical picture. Long-term oxygen therapy is usually administered continuously and in a home-care setting to improve survival, exercise, sleep, and cognitive performance. In addition, a cardiological consult may be requested, to exclude cardiovascular instability. Finally, the pulmonologist must **Review COPD stage** in order to confirm the stage after treatment or to update it.

At the end of each instance of the COPD ordinary management process, the general practitioner must either **Confirm COPD stage** or **Update COPD stage**, if this has changed. In case of stage change, signal event **Re-schedule** is thrown to be caught by the corresponding interrupting boundary event, depicted in Fig. 12.2 and attached to the border of sub-process **Management of COPD Exacerbations**.

The introduced healthcare process representing the ordinary management of stable COPD is periodically re-enacted and the patient is also involved in educational programs to improve recognition of symptoms worsening and self-care.

12.2.5 Management of COPD Exacerbations

COPD exacerbations are “acute episodes characterized by a sudden worsening of the patient respiratory symptoms that is beyond expected day-to-day variations” [269]. The main drawback associated with COPD exacerbations is represented by hospital admissions, that negatively impact the disease evolution, economical costs, and quality of life [270].

COPD exacerbations seem to be triggered by viral or bacterial infections of the upper respiratory tract, as well as by air pollution. Comorbidities such as pneumonia, congestive heart failure, pulmonary embolism and pneumotorax are possible triggers of exacerbations.

Several clinical findings must be considered when evaluating patients with exacerbations. Among these COPD severity, the presence of comorbidities, and the history of previous exacerbations are the most important. Indeed, patients that are subjected to two or more exacerbations of COPD per year seem to maintain this worsening trend over time and, thus, must be constantly monitored. A diagnosis of exacerbation requires a physical examination to evaluate the effects of the episode on both cardiovascular and respiratory systems [258].

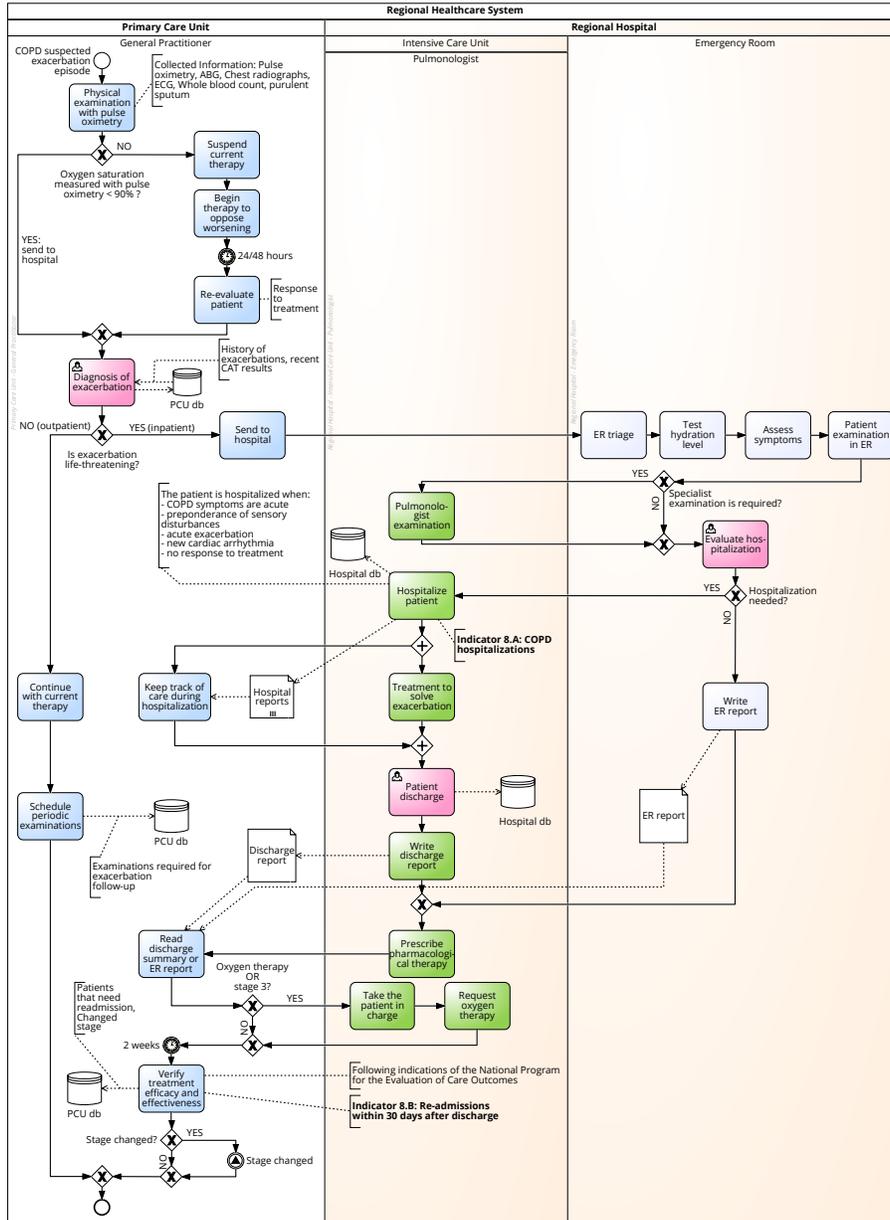


Fig. 12.5: BPMN process diagram for the extraordinary management of COPD exacerbations.

COPD exacerbations are often treated with temporary short-acting inhaled β_2 -agonists, which can be combined with corticosteroids and antibiotics. However, if the patient does not respond to outpatient pharmacological treatment, hospitalization is required. When a patient is admitted to hospital, supplementary oxygen is administered, short-acting bronchodilators are prescribed, and pulmonary rehabilitation may also be indicated [99].

COPD hospital admissions due to exacerbations negatively affect the patient quality of life and the progression of the illness, while substantially increasing the costs of patient management [270]. However, a standardized and integrated care intervention, supported by proper data collection and resource coordination, seems to effectively prevent hospitalizations due to exacerbations in COPD patients [270, 271].

The goal of exacerbation treatment is to minimize the effects of the on-going worsening and prevent future episodes. The process enacted for treating COPD exacerbations in Veneto is depicted in Fig. 12.5.

An exacerbation episode is usually detected by the General Practitioner, following the clinical presentation of the patient, who complains of an acute change of symptoms. Diagnosis of exacerbation relies on the current symptoms, on the evidence gathered during the Physical examination with pulse oximetry, and on the history of previous worsening episodes. If more than two exacerbations occur within one year, the stage of the disease is expected to vary, as the patient respiratory function is compromised more rapidly [99].

In general, diagnosing an exacerbation can become quite cumbersome, as multiple physical findings and diagnostic procedures must be evaluated, with the main aim of excluding the development of comorbidities. If oxygen saturation measured with pulse oximetry is $<90\%$, the exacerbation is likely to be life-threatening and, thus, hospital and intensive care treatment is indicated. Otherwise, the general practitioner can Suspend current therapy in order to Begin therapy to oppose worsening. Usually, short acting beta agonists (SABA), short acting muscarinic antagonists (SAMA), inhaled corticosteroids and antibiotics are prescribed, according to the considered clinical picture. Then, after 24/48 hours the patient is re-evaluated. Following clinical diagnosis, gateway Is exacerbation life-threatening? determines if the patient needs to be treated in inpatient or outpatient settings. If the patient has improved, the exacerbation can be managed in an outpatient setting. In these circumstances, it is indicated to Continue with current therapy and to Schedule periodic examinations to ensure proper patient monitoring.

If the patient has not improved after treatment or oxygen saturation is assessed $<90\%$, then her or she must be sent to the hospital Emergency Room. Hospitalization must be considered when the patient cannot be treated adequately in home care settings, due to the severity of the respiratory dysfunction [99, 258].

Following ER triage, ER physicians must Test hydration level to ensure proper fluid balance and they need to Assess symptoms. Then, the patient is physically examined and continuously monitored. Physicians in ER must exclude other clinical complications, such as heart failure, bronchial pneumonia, pul-

monary embolism, and pleural effusion. A **Pulmonologist** examination in ICU can be requested to assess arterial blood gases and execute chest radiography. If the severity of respiratory dysfunction is high, the patient must be admitted to an intensive care unit and treated under the responsibility of a pulmonologist. Different factors must be considered when evaluating hospitalization. Typically, patients are admitted in ICU when COPD symptoms are acute, there is a preponderance of sensory disturbances, there is evidence of new cardiac arrhythmia, or no response to exacerbation treatment is observed [258]. The **Treatment to solve exacerbation** is mostly based on oxygen therapy, which aims to prevent tissue hypoxaemia, i.e., reduced oxygen availability, and at preserving cellular oxygenation. During hospitalization, the **General Practitioner** is informed about the administered care. This is necessary to provide practitioners with all the information, needed to optimally treat the patient after hospital discharge.

Patient discharge is decided after evaluating several criteria, which consider also local policies and availability. The discharge report must contain information regarding the possibly changed COPD stage, respiratory function, comorbidities, outpatient suggested therapy, and clinical follow-up. Patients dismissed with hypoxaemia or in Stage 3 may require short-term or long-term oxygen therapy. In this case, a pulmonologist must **Take the patient in charge** and a home-care oxygen therapy protocol is started. Oxygen therapy is monitored by specialized centers in the region and its efficacy must be re-evaluated yearly.

As required by the national program for the evaluation of care outcomes, **two weeks** after patient discharge, the general practitioner must **Verify treatment efficacy and effectiveness**. The aforementioned evaluation program aims to estimate the overall quality of the provided care, in terms of treatment costs, appropriateness, and efficacy, in order to compare the outcomes of the overall care intervention with audit and bench-marking data.

Hospital re-admission of COPD patients within 30 days from discharge is probably the most important indicator used for assessing the quality of the overall care plan [270, 272]. In the studied context, around 15% of discharged patients is re-hospitalized within 30 days from discharge.

During this final verification phase, the stage of COPD is also reviewed and, if it has changed as a consequence of the exacerbation, signal **State changed** is thrown. This is caught by the corresponding non-interrupting boundary event attached to the border of sub-process **Management of COPD Exacerbation** of Fig. 12.2, which enables an exception flow devoted to the rescheduling ordinary examinations. Likewise ordinary management of COPD, exacerbations require follow-up. This includes reassessment within 4 weeks, evaluation of improvement in symptoms, physical examination, assessment of need for oxygen therapy, and re-adjustment of current treatment regimen.

Wrapping up, the modeled BPMN processes have provided healthcare professionals with a comprehensive and accurate overview of the COPD care process. After applying the proposed methodology, improvements in common understanding and cooperation have been noticed. In earlier times, each resource used to work independently and this fragmentation resulted in longer waiting times for

patients, miscommunication, and information overload. The standard definition of the care processes has served to delineate everybody's responsibility, while improving information exchange, cooperation between resources, and coordination between several processes. Besides, from a technical point of view, the process model has been used for validating the correctness and efficiency of the enacted measures, thus constituting a sound trace for implementation.

12.2.6 Modeling Clinical Decisions with DMN

Process-related decisions have been modeled in detail by using DMN diagrams and, sometimes, by specifying more detailed decision logic. Decision models based on process-related data and regarding the diagnosis of patients assessed with COPD or being re-admitted to hospital after an exacerbation have been presented in Chapter 11 for a slightly different process.

In this section, we focus on describing the decision modeling for the management of COPD exacerbations process of Fig. 12.5.

In the management of COPD exacerbations, a crucial step is represented by the diagnosis and assessment of an exacerbation, which is evaluated by combining physical findings with the clinical history of the patient [99, 270].

In general, clinical diagnosis is probably the most important example of complex, decision-intensive medical task. Indeed, diagnosis encompasses aspects of human decision-making that rely on medical knowledge, professional experience, and sound sources of clinical information. For this reason, modeling clinical diagnosis by means of formal approaches can become quite cumbersome, as it involves complex (temporal) reasoning and uncertain and incomplete knowledge [112].

Nevertheless, the representation of structure of a decision and of its meaning is interesting for the organization and can be realized by using DMN DRDs, which allow one to describe decision-making at quite a high level.

As an example, consider the DMN diagram representing the diagnosis of a COPD exacerbation, depicted in Fig. 12.6. The diagnosis of an exacerbation relies on the clinical presentation of the patient, who consults the general practitioner complaining of substantial symptomatic and physiologic deterioration [258].

In the DRD of Fig. 12.6, Diagnosis of COPD Exacerbation is portrayed as the main decision, which ideally responds to the question "Is the patient experiencing an exacerbation of COPD?". The decision is made by a **General Practitioner**, represented as a knowledge source, who is responsible for assessing the degree of symptom worsening and for evaluating if the exacerbation is life-threatening.

Several clinical findings have to be considered when assessing an exacerbation. During a physical examination, **Pulse oximetry** is performed to assess the need for supplementary oxygen therapy. Besides, **Hemodynamic stability**, **Use of accessory respiratory muscles**, and **Response to treatment** are evaluated.

The general practitioner must also consider the number of previous exacerbation episodes and the **Presence of comorbidities**, which are combined with information about COPD severity to predict the patient risk of exacerbations. COPD severity corresponds to the output of sub-decision **Evaluate Severity of**

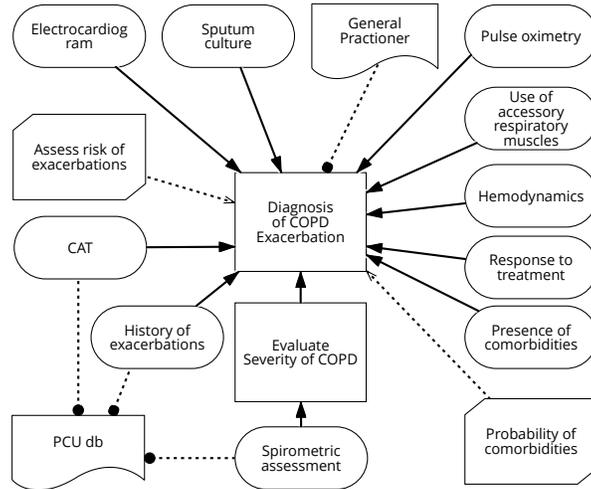


Fig. 12.6: Decision Requirement Diagram representing the main decision **Diagnosis of COPD Exacerbation**, together with the input data needed for making the decision, the used business knowledge, and the sources responsible for the decision. The decision is linked to task **Diagnosis of exacerbation** of Fig. 12.5.

COPD, which is based on **Spirometric assessment**. Results of previously administered **CAT**, **Spirometric assessment**, and **History of exacerbations** are retrieved from the **PCU db**, depicted as knowledge source.

Probability of comorbidities is a business knowledge model that describes the kind of comorbidities that are commonly found during an exacerbation and the probability of finding more of them. Business knowledge models describe a reusable fragment of decision logic or know-how, which can be retrieved by guidelines or by local organizational knowledge. In the presented context, the knowledge is based on data collected in the region, since the probability of comorbidities changes according to the distribution of the population in the territory.

Similarly, business knowledge model **Assess risk of exacerbations** encloses the knowledge required for assessing exacerbation risk, based on the combination of COPD severity, symptomatic assessment and history of previous exacerbations [258].

The diagram of Fig. 12.6 is linked to the user task **Diagnosis of exacerbation** of the process shown in Fig. 12.5. The result of the decision is used by gateway **Is exacerbation life threatening** to determine if the patient must be treated in outpatient settings or hospitalized.

The DRD introduced in Fig. 12.6 encompasses both human decision-making and partially automated decisions, the latter ones reviewed and validated by the general practitioner. For the main decision **Diagnosis of Exacerbation** we decided

not to specify decision logic, as the presented DRD suffices in explaining the expected structure and input data needed to make this human decision.

However, a few elements in the considered diagram have a decision logic specified at different levels of abstraction, by using the various expressions provided by the DMN standard [47]. In detail, decision logic for business knowledge model **Probability of comorbidities** is represented by a boxed a literal expression, shown in Fig. 12.7, written as plain English text and, thus, not suitable for automation.

Probability of Comorbidities
<p>Common comorbidities associated with poor prognosis of COPD exacerbations are: congestive heart failure, coronary artery disease, renal and liver failure.</p> <p>According to data gathered in the Region of Veneto, the percentage of patients presenting comorbidities is as follows:</p> <p>12% of COPD patients has only one comorbid condition;</p> <p>21,1% of COPD patients has two comorbid conditions;</p> <p>22,6% of COPD patients has three comorbid conditions;</p> <p>16,2% of COPD patients has four comorbid conditions;</p> <p>The rest of patients can have from 5 to 10 comorbid conditions.</p>

Fig. 12.7: Boxed literal expression, written in plain English, and representing the decision logic of the **Probability of Comorbidities** business knowledge model.

Conversely, sub-decision **Evaluate Severity of COPD** is based on well-defined rules that determine how the amount airflow limitation should be associated to a specific COPD stage. Hence, decision logic can be easily represented by means of a decision table that relates spirometric values FEV1 and FEV1/FVC to the corresponding COPD stage.

Fig. 12.8 shows the decision logic of decision **Evaluate Severity of COPD**, which invokes decision table **Evaluate Severity of COPD Table**, passing **Spirometric assessment.FEV1** as **FEV1** parameter and **Spirometric assessment.FEV1/FVC** as **FEV1/FVC** parameter. Decision **Evaluate COPD Severity**, invokes the corresponding decision table. The presented table has been verified with Signavio [24] and, as expected, it is not complete as no rule exist for ($\leq 70\%$, *any*). However, this omission was intentional, since this latter scenario is not considered in COPD treatment.

Finally, as an example of formal expression allowed in DMN, consider the decision logic for business knowledge model **Assess risk of exacerbations**, represented in Fig. 12.9 through a boxed FEEL expression which describes how an output value is derived from its input values. The presented formal expression is used for combined COPD assessment in order to understand which is the impact of COPD on an individual patient, in terms of future exacerbations, based on current symptoms, spirometric assessment, and exacerbation history [258].

Patients are classified into 4 main categories, which summarize the risk of exacerbations and expected health status. For instance, a patient with a CAT

Evaluate Severity of COPD			
Evaluate Severity of COPD Table			
FEV1	Spirometric assessment.FEV1		
FEV1/FVC	Spirometric assessment.FEV1/FVC		

U	Evaluate Severity of COPD Table		
	FEV 1/FVC (percentage)	FEV1 (percentage)	COPD Severity
			{“Mild”, “Moderate”, “Severe”}
1	< 70 %	>= 80 %	“Mild”
2	< 70 %	[50%..80%]	“Moderate”
3	< 70 %	< 50%	“Severe”

Fig. 12.8: Decision table representing the decision logic for decision Evaluate severity of COPD, based on spirometric data.

score of 16, assessed with “Moderate” COPD, and a history of 3 exacerbations within the last 12 months has a risk of exacerbations corresponding to “High, more symptoms”.

12.2.7 Dealing with Data and Process Indicators

The modeling of the described COPD processes and decisions underlined the strong connection that exists between processes and related clinical and organizational information, throughout all the design phase.

On the one hand, data needed for process and decision execution must be identified during process design, to define which is the information needed during process enactment. On the other hand, new informational value generated during process simulation and execution must be properly managed and used for re-engineering as-is processes and decisions.

Risk of Exacerbations
If COPD Severity IN (“Mild”, “Moderate”) or ExacerbationsPerYear IN ([0..1]) and CAT <= 10 then Risk of Exacerbations = “LOW, LESS SYMPTOMS” else if COPD Severity IN (“Mild”, “Moderate”) or ExacerbationsPerYear IN ([0..1]) and CAT >= 10 then Risk of Exacerbations = “LOW, MORE SYMPTOMS” else if COPD Severity = “Severe” or ExacerbationsPerYear IN ([0..1]) and CAT <= 10 then Risk of Exacerbations = “HIGH, LESS SYMPTOMS” else Risk of Exacerbations = “HIGH, MORE SYMPTOMS”

Fig. 12.9: Decision Logic for the Assess risk of exacerbations business knowledge model, adapted from guidelines on combined COPD assessment [258].

In the management of chronic diseases, most clinical knowledge and conformance data stem from standard guidelines, whereas operational data are retrieved from process logs or health information systems [273]. However, health information systems are usually meant to serve multiple and disparate care processes and, thus, data availability, standardization, and maintainability may become a complex issue.

In the studied context, a first step towards the identification of a minimum set of data that are relevant for process execution and audit is represented by the definition of process indicators, used to assess the quality of the provided care from different viewpoints and levels of abstraction.

The definition of indicators is part of the care process specification and is done by a “focus group” of clinical experts, who identify clinical, organizational, and social criteria for evaluating process outcomes [274]. Indicators are used to validate the structure of the process and to evaluate the organization potential in terms of resource distribution and consumption. In addition, they are useful to estimate process compliance with respect to both clinical and organizational outcomes, and cost sustainability.

In BPM, Key Performance Indicators (KPIs) are defined to monitor process execution, evaluate the quality of process and decision outcomes, and quantify the overall progress towards organizational goal achievement [240]. In this work, depending on their scope, we discerned two kinds of KPIs.

Local KPIs correspond to process and decision metrics and are mostly used to measure activity execution times, costs, and resource consumption.

We refer to such indicators as “local”, as they are extracted/used directly from/by multiple instances of one process model. An example of local KPI related to process metrics is the average total cost spent by the Regional Healthcare System for treating COPD patients assessed with Stage 3. Local KPIs allow one to quantify the probability that a process execution path is preferred over another, within the context of a data-based process gateway. For instance, the percentage of smoking patients determines how many times the care providers involved in the process of COPD treatment have to deal with smoke cessation.

In the region of Veneto, it has been estimated that the probability of contracting COPD for smokers 25% higher than for non-smoking patients, and can increase up to 30% in patients that smoke more than 30 cigarettes per day. In the process of Fig. 12.3, this probability value influences the execution of all tasks that concern the administration of smoking questionnaires, as Mondor and Fagerst orm tests can be skipped if the patient is not a smoker. Local KPIs can also be derived and evaluated with process simulation techniques, as explained in Sect. 12.2.8.

Conversely, *global KPIs* are used to evaluate the strategic value of the performed (decision) activities both quantitatively and qualitatively. Global KPIs can be directly re-used in clinical decision modeling, to improve decision quality, patient prognosis, and predictive analytics. Usually, global KPIs rely on aggregated data and their impact can affect several activities and processes. In addition, they can be used for organizational and re-engineering purposes that

go beyond the scope of the defined processes. These indicators are rather based on clinical and healthcare data, and provide a summarized and focused overview of the modeled domain. Global KPIs can have both a clinical or organizational character.

For instance, the appropriateness of drug prescription is a clinical indicator, whereas the time needed for treating a patient in Stage 2 is considered to be an organizational indicator. Of course, nothing prevents the same indicator to be used for both clinical and organizational goals. The indicators defined by the Regional Healthcare System of Veneto are global KPIs used to give a high level estimate of the impact of care standardization in terms of costs, patients involvement, and medical personnel competence. Such indicators have been highlighted as text annotations in the process diagrams of Sect. 12.2.2.

For example, **Indicator 5: Performed Vaccinations** is defined to quantify the number of performed influenza vaccinations, in order to assess the quality of secondary prevention of COPD. It is defined as the proportion between patients diagnosed with COPD (i.e., ICD-9-CM 491.2x e 496.x) aged at least 40, and having received a vaccination against influenza in the last 12 months and the total of COPD patients aged at least 40. This indicator is associated to task **Verify administered vaccinations** of the process shown in Fig. 12.4 and experts expect the value of this indicator to remain greater than the 40% of all the patients diagnosed with COPD (and quantified by **Indicator 2A**). Any value lower than the expected threshold should be considered an index of poor process performance, to be further investigated by considering re-engineering.

With process management tools, **Indicator 5** can be calculated by counting how many times the process branch labeled with YES is followed after each exclusive gateway labeled with question **Vaccination administrated?**.

As for data requirements, this indicator suggests that the following data need to be collected during the care process: A patient identifying code, the diagnosis of COPD registered according to the ICD-9-CM coding systems, the influenza vaccination, and the date of the vaccination.

As for the global scope of the indicator, data about the performed vaccinations gathered during the management of stable COPD in Veneto, are combined with data collected at a national level to evaluate the efficacy of routine vaccination programs. In the years 2002-2009, 32% of Italian population received vaccination against influenza, whereas in the years 2010-2016 the same value dropped to 18%. Among patients with COPD, around 75% of them received a vaccination against influenza in the same time-span. This drop in the number of vaccinations, highlighted by **Indicator 5**, can be explained by the recent spread of several anti-vaccination movements that developed in the considered territory in the last years. As a consequence, new vaccination programs were introduced, patients were actively informed about the risks of non-vaccination, and a service for delivering vaccinations at home was provided. This latter intervention was directed not only to COPD patients, but to all those considered at risk by the National Healthcare System, whence the global scope.

The remaining indicators defined within the processes can be used as **Indicator 5** to define (clinical) data requirements and measure clinical and organizational outcomes.

12.2.8 Simulation of COPD Processes

In the application of the proposed methodology, we used simulation for validating the behavior of the modeled COPD processes by spotting pitfalls related to erroneous control flow. Then, we evaluated the performance of the processes based on current data collected from the Regional Healthcare System, and we studied the behavior of the processes when dealing with hypothetical scenarios. The latter ones have been defined according to the indications of experts, who were interested in gathering insights regarding both the quality of the interactions between process resources and the performance in case of increased amount of patients in severe conditions.

We exploited the simulation functionalities provided by the Signavio Process Editor [24], which allows one to directly open the process model in the simulation environment, without the need for further configuration.

Signavio supports three kinds of simulation of syntactically sound BPMN processes (verification of the syntax is done by the tool prior to simulation), namely (i) step-by-step, (ii) one-case, and (iii) multiple case simulation.

(i) Step-by-step simulation is guided by the modeler, who chooses which process flows are enabled, and it is mainly used for verifying that the process captures all the possible desired execution behaviors. This is optimal for comprehending the process flow and, therefore, it can be used when interacting with clinicians for improving common understanding. (ii) One-case simulation allows one to reproduce the execution of a single process instance, according to specific conditions defined on process execution. Conversely, (iii) multiple-case simulation supports the run of multiple instances of the same process model, thus allowing one to aggregate results and analyze the overall workload of the process.

In all circumstances, Signavio allows the token flow to be graphically visualized, so that partial results can be visually monitored by a process modelers and stakeholders.

The simulation tool requires the following input information:

1. the probability associated to each outcome of an exclusive gateway (e.g., the proportion of COPD patients in Stage 3);
2. the fixed costs associated with each activity (e.g., spirometry performed in hospital costs around 25€);
3. the probability distribution that describes the processing time for each activity (e.g., arterial blood gas analysis lasts 10 minutes, on average);
4. the hourly cost of each resource and the corresponding working schedule.

Data regarding the costs and duration of examinations have been collected by regional hospitals, by analyzing scheduled events and medical reports.

On the contrary, the number of primary and hospital care resources involved in the management of COPD is managed directly by the Regional Healthcare

System. To obtain a reliable estimate of the amount of time that each resource devotes to COPD care, we considered that COPD patients in Veneto correspond to the 7,9% of the overall regional population. Accordingly, we assumed that the same proportion of patients is assigned to each practitioner, since in Italy each general doctor is responsible for the same amount of patients. Of course, we had to take territorial variance into account, as the percentage of ill people increases up to 11,8% in polluted areas. The same strategy has been adopted for estimating the time that pulmonary specialists working in hospitals dedicate to COPD patients.

We created multiple simulation scenarios, starting from those that reproduce current reality to interesting what-if scenarios, suggested by experts.

In this regard, we analyzed the behavior of the process in critical scenarios, such as the increased proportion of severe patients, longer hospitalization following exacerbations episodes, and increase in the number of COPD sufferers during winter.

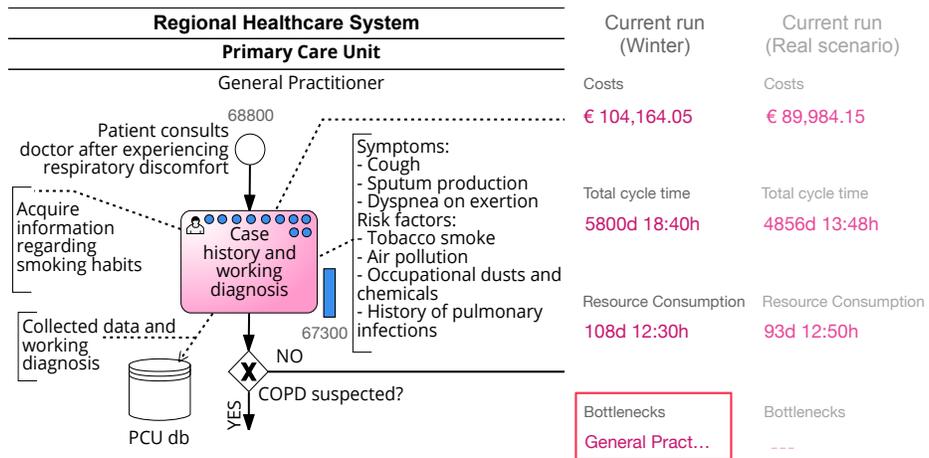


Fig. 12.10: Simulation results shown in Signavio for scenario Winter, run on 68800 instances over 120 days. The tool highlights a bottleneck corresponding to the general practitioner, who is responsible for diagnosing COPD.

In Winter, the conditions of patients tend to worsen due to the cold weather and evidence suggests that in Veneto there are 15% more cases of COPD in winter than on average during the year. We simulated the process *COPD Diagnosis and Assessment* for 120 days, considering that 185.000 patients are diagnosed every year. As for resources, we considered having 3000 pulmonary specialists, working 15000 hours per week with COPD patients, and 3500 general practitioners, working 58000 hours per week with COPD patients.

The simulation interface, shown in Fig. 12.10, suggests a bottleneck in correspondence of the general practitioner, who is overloaded with the first diagnostic assessment. When visualizing the details of the simulation, the workload for the general practitioner is estimated to be around 184 days full-time, which is an excessive amount compared to the considered working schedule for 120 days.

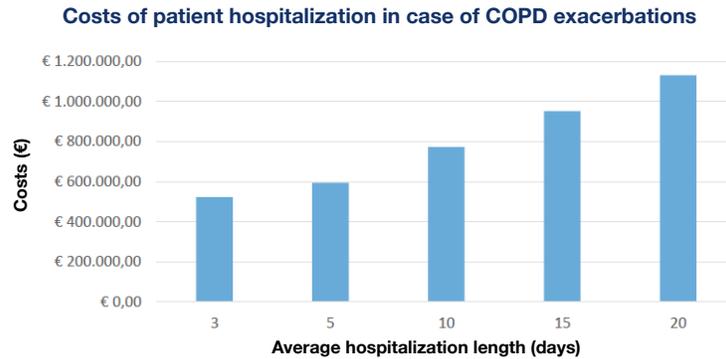


Fig. 12.11: Simulation results of process *Management of COPD Exacerbations* considering different hospitalization lengths, run on 72000 instances over 100 days, with 23800 patients hospitalized.

Simulation results of process *Management of COPD Exacerbations*, obtained for different lengths of hospitalization, are outlined in Fig. 12.11.

The costs shown in the histogram are calculated by summing the fixed costs of hospitalization with the costs associated to the resources. In detail, we simulated the case of having 72000 patients experiencing COPD exacerbation and considered that the 34% of them is hospitalized. As expected, the costs of the process increases drastically whenever patients are hospitalized for a longer time. The distribution of the total cost of process *Management of COPD Exacerbations* considering the most relevant tasks for expense calculation is shown in the pie chart of Fig. 12.12. The depicted results correspond to the execution of the process by setting an average hospitalization length of 15 days.

In Fig. 12.13, the difference between the costs associated to the performance of spirometry are shown. As discussed in [59], spirometry should be preferably performed in PCUs, as the costs of hospital spirometers is higher and waiting lists should be taken into account.

Finally, we analyzed how the duration of the process changes in relation to waiting times and examination priorities. One-case simulations were performed based on real data about waiting times for medical examinations, which can either be “urgent”, “non urgent”, or “scheduled”. Severe patients, classified as “urgent”, are treated within 18 days on average, whereas patients that are scheduled according to the lowest priority need around 120 days to go through the

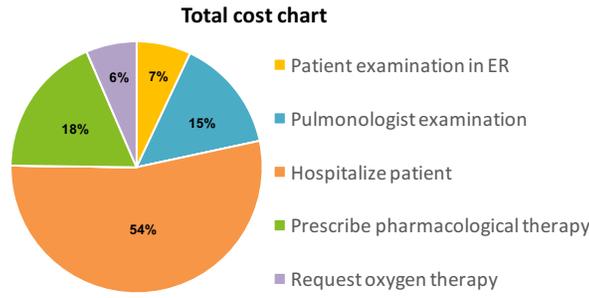


Fig. 12.12: Distribution of the total cost of executing process *Management of COPD Exacerbations*, considering a hospitalization length of 15 days.

whole *COPD Diagnosis and Assessment* process. These values have been obtained by analyzing the waiting times associated to examination prescriptions for COPD patients.

The real results obtained by simulation are shown in Fig. 12.14 and are in accordance with the expected values. In addition, a visual comparison with the maximum waiting required by the Italian law are shown.

Overall, these graphical summaries of simulation data have proven useful to describe the obtained results to stakeholders and to underline the connection between process performance and the defined clinical indicators.

For instance, let us consider examination priority, described in Fig. 12.14. Despite the great difference in waiting times, which can be harmful in some cases, results have shown that the costs associated to the aforementioned process double if 50% of the patient is assigned priority “urgent”. Moreover, bottlenecks

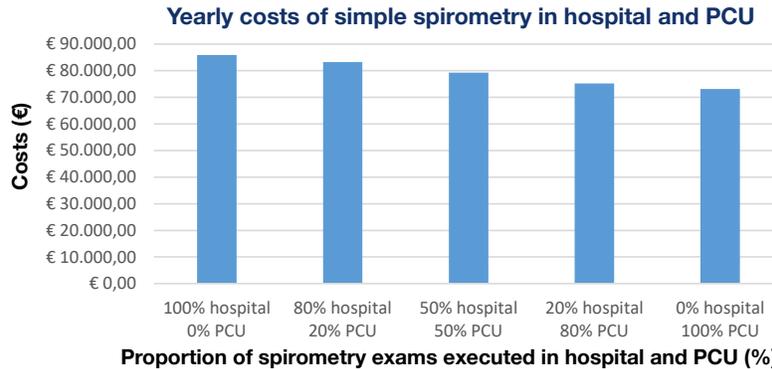


Fig. 12.13: Costs of the process *COPD Diagnosis and Assessment* based on changing proportions of spirometry examinations carried out in PCUs or in hospital.

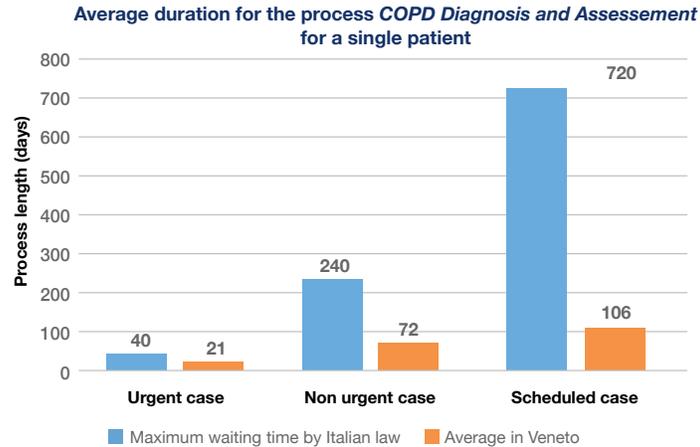


Fig. 12.14: Average length of the process *COPD Diagnosis and Assessment* based on different waiting times, depending on the examination priority assigned to “urgent”, “non urgent”, or “scheduled” patients. In orange (right column), the average results are shown for Veneto, whereas in blue (left column) the national threshold are shown for comparison.

related to resource consumption arise soon and concern pulmonary specialists, who are in charge of patients in Stage 3. Since both inappropriate diagnosis and the assignment of wrong priority to patients lead to increased expenses and process bottlenecks, Indicator 1 and Indicator 2A are defined in the process of Fig. 12.3 to evaluate the accuracy and quality of diagnosis. In this specific case, experts had already defined such indicators prior to simulation, based on data obtained from the as-is process. However, in other circumstances new indicators can be proposed and tested, according to simulation results.

Decision logic has been simulated with the Camunda DMN Simulator [275], which allows one to verify the output of a decision table, given custom input values.

Overall, simulation results confirmed the concerns of experts, regarding increased waiting times for patients in Stage 1, and a worrying increase in costs and resource consumption related to an augmented proportion of severe patients. Surprisingly, data regarding resource consumption reveal that pulmonary specialists in Veneto tend to be more burdened than general practitioners, despite the central role played by the latter ones in the process.

12.3 Conclusions

In this chapter, we introduced a methodological framework for supporting the design, implementation and enactment of decision-intensive healthcare process,

by exploiting standard process design techniques and by integrating them with proper decision modeling.

We advocate that combining the advantages derived from the design of process and decision models, each one stressing on a different care perspective, increases the overall value that such a synergistic design approach can bring to care process development. In addition, the proposed methodology provides a preliminary classification of the kinds of data involved during the different phases of process development. Proper information and knowledge management allows stakeholders to better support iterative process design, execution, and re-engineering, thus leading to more flexible and information-aware processes.

To conclude, the advantages brought by the proposed methodology for organizational process design are compared to those achieved by computer-interpretable guidelines for clinical process modeling in Chapter 13.

Related Work

In this chapter, we discuss related work conducted in the field of integrated process and decision modeling, focusing on applications in healthcare domains.

In Sect. 13.1 we begin with reviewing general approaches combining process and decision modeling with BPMN and DMN. Then, in Sect. 13.2 we focus on research efforts conducted in the field of healthcare process modeling. Throughout the chapter, we discuss and compare the contributions of Chapter 11 and Chapter 12 with selected relevant approaches.

13.1 Process and Decision Modeling

Despite having existed as an independent and evolving discipline [224], decision management is increasingly being used in conjunction with business process management to improve business outcomes and competitiveness. Organizations need to extract information and knowledge useful for decision-making and process improvement out of data being collected by their business processes and other (big) data sources [?, 235, 276]. Besides, the increasing interest in knowledge-intensive processes demands proper support to ensure that the best decisions possible are made and that the collaboration between knowledge workers is fruitful [75].

However, since both decision and process management have long existed without proper integration, process modeling languages have been misused to encode decision logic [20, 50, 219]. As a result, existing process models often incorporate decisions that are encoded into control flow structures [20], hidden within process activities [219], or implicitly contained in process execution logs [222, 223]. This improper integration trend leads to maintainability, flexibility, re-usability, and scalability issues in both process and decision models [50, 226].

Therefore, separation of concerns has drawn increasing interest in the BPM field [109, 218, 221, 224], especially since the introduction of the DMN standard [47], designed to elicit and represent decision models that can possibly be used to complement BPMN process models, thus keeping concerns separate.

Several research approaches have addressed the design or derivation of DMN decision models that are complementary to BPMN process models with the aim of separating concerns, yet integrating the modeling of decisions and processes [20, 48, 50, 58, 59, 220, 235].

In [48], DMN is used in the context of collaborative networks to discern decisions that are incorporated in BPMN process models from those that can be modeled through more appropriate DMN diagrams. In [220], the authors propose a methodology for automatically deriving process models from Product Data Models that capture the complex data dependencies underlying a workflow. The approach allows one to obtain a BPMN process model emphasizing the most important decisions, while detailed decision logic is outsourced to a dedicated DMN model. The extraction of decision logic from the process control flow is considered in [20]. The authors identify a set of control flow patterns often misused to capture decisions and show how DMN models succeed in reducing the complexity of a process model that embeds decision logic. In [235], the authors adopt event processing techniques to address the re-evaluation of decision based on updated and relevant real-time data represented by events, that may change the outcome of a decision being made during process execution. In [59, 58], the authors discuss how to jointly use the BPMN and DMN standards to represent the structured organizational aspects of real-world decision-intensive healthcare processes that include complex clinical and organizational decision-making.

Some of the introduced approaches focus on control flow aspects [20, 48]. Probably, this trend originates from the role of primary importance given in the BPMN standard to the control flow perspective. However, as motivated in Chapter 7, BPMN allows designers to represent data through data artifacts, events or text annotations [11, 70], thus providing some open contact points to model the data perspective of business processes.

In Chapter 11, we provided decision designers and analysts with an approach to ease the derivation of DMN models from data that are *explicitly* represented in BPMN process models and that provide input information for decision activities.

In line with previous research [20], we assumed that stakeholders play an active role in approving whether the identified decision activities are real business decisions. However, dealing with data rather than control flow is more challenging for two main reasons. On the one hand, data contribute to build both explicit and implicit process knowledge [232] and, thus, they may contribute to decision-making in multiple ways beside being an input for decisions. On the other hand, the same data may be shared between process and decisions models, yet addressing different concerns.

As a result, consistent process and decision models integration is of prime importance [50], as the same piece of information may be used in both models for different purposes and managed by different stakeholders. Lying at a conceptual level, our approach supports decision analysts and stakeholders in the identification of data relevant for decision making, but leaves the freedom to choose the most appropriate level of process and decision model integration.

The consistent integration of the BPMN and DMN standards has been the focus of some relevant contributions [49, 50, 226, 219]. In [219] the authors frame the role of a decision model within the context of a related business process and examine execution mechanisms for different availability of input data. Five novel principles for integrated process and decision modeling, called 5PDM, are proposed in [50] to guide designers in avoiding and solving inconsistencies between process and decision models, such as the unsound ordering of decision activities or the absence of input data. Consistency requirements between DMN decision models and BPMN process models are defined in [226]. The authors analyze multiple kinds of process models and, based on their structure and characteristics, provide a set of requirements for integrating DMN decisions. Finally, in [49] the authors identify common challenges related to the refactoring of process models that arise when integrating decision models.

Both the proposals presented in [49, 226], rely on the definition of relevant integration scenarios for process and decision models.

In Chapter 11, we took the view of having both decisions and processes intertwined within the same model, as we allowed the possibility of having hidden decisions be hard-coded within the process [49, 50]. Despite considering process models having decisions as a local or global concern [49, 226], we focused on discovering decisions having a scope restricted to the considered process.

For example, let us consider the process of patient discharge. The information related to the patient's mobility and nutritional intake is used by care givers to plan how the patient should be treated in hospital and at home once discharged. This information affects daily decisions related to transport arrangements (e.g., decide whether requiring night assistance for a patient that is not able to walk for more than 25 minutes unaided) and meal plans (e.g., decide whether precluding discharge if the patient does not tolerate solid meals), as well as higher-level managerial decisions related to re-admissions, long waiting lists, and bed-blocking (e.g., if there are, which are the criteria related to the patients' mobility that cause delayed discharge?).

In Chapter 11, we focused on capturing and representing the first kind of exemplified decisions, that is, those made within a specific process and based on process-related data. These are sometimes referred to as local decisions [50]. The discovery of higher level, strategic organizational decisions that are not explicitly made in business processes or span over multiple processes was not discussed in Chapter 11, but a higher level idea of *global decisions* was provided in Chapter 12.

As argued in [49, 50], the main obstacle to consistent process and decision model integration appears to be the declarative nature of the DMN standard, which clashes with the dependency of process-related decisions from the invoking context. A solution to support separation of concerns while ensuring consistency between process and decision models is the representation of decisions as externalised services, following the principles of Service-Oriented Architecture (SOA) paradigm [227, 228]. According to this approach, business rules are grouped into a decision services that are incorporated within a web services layer to be consumed by the business process layer. For example, SOA^{+d} defines fine and

coarse-grained services that are necessary for modeling the business, information system, and decisions of an organization [228].

More general approaches have investigated the integrated modeling and externalization of decision rules [231, 234]. In [234], the authors empirically evaluate a set of factors that affect the choice of whether business rules shall be incorporated into process models and propose a set of guidelines for improving the modeling of business rules. Similarly, in [231] the effects of business rule integration on business process model understanding is discussed and evaluated empirically.

In the field of decision mining, a framework to classify activities in a process model based on how they contribute to the overall decision dimension of a process is presented in [277]. The introduced approach enables an in-depth analysis of every activity in order to establish whether it entails a decision, and how it is related to other activities. Last but not least, other approaches presented in [222, 223, 225, 278] aim to semi-automatically extract complex decision logic from process event logs using decision trees and other conventional process mining techniques.

13.2 BPM and Healthcare Process Modeling

In this section, we consider the methodological framework presented in Chapter 12 and provide an overview of existing approaches that are relevant for the discussed methodological steps and tools used.

Business process management has been employed in healthcare to support the modeling and enactment of organizational interdisciplinary processes [51]. However, applications of process modeling approaches in healthcare are still modest, despite the suitability of BPM techniques for representing complex and multidisciplinary organizational procedures. In addition, existing approaches tend to target specific clinical domains rather than proposing general methodologies, as exhaustively surveyed in [76].

Process Modeling in Healthcare

Healthcare is considered a challenging domain for BPM approaches, as flexibility is often required and processes heavily depend on both information and knowledge [13, 26, 51]. The role of human knowledge is considered central in knowledge-intensive processes and, due to the variability in their structure, they cannot be directly mapped to business process models [75].

The *organizational* aspects of knowledge-intensive processes can be captured by BPM techniques [51, 76], whereas *clinical* or *medical treatment* processes can suitably be handled by approaches based on case management or business artifacts [62, 91, 279, 280].

For instance, the ADEPT framework [279] allows one to revise running process instances and schemata in order to support the management of clinical guidelines. Among the interesting features of ADEPT, modularity, ease of use and management of temporal aspects stand out.

In the context of case management, a clinical pathway for living donor liver transplantation was firstly modeled using BPMN in [280]. Then, in order to facilitate transferability of the modeled pathway to other hospitals and to include flexibility caused by differences in treatments due to local policies, the processes have been modeled by using the OMG standard Case Management Model and Notation (CMMN)[62]. This standard can be used to model specific cases, together with the data that drive their execution, in order to simulate the decisions taken by a knowledge worker during process run-time.

However, traditional BPM approaches are activity-centric and, thus, they succeed in supporting structured and repetitive organizational processes. For example, BPMN is used for modeling surgical pathology processes in [243]. The authors claim to have chosen BPMN because of its widespread use and tool support, and because they were looking for a language that allowed them to model “transparent” processes, easy to be understood by care providers.

Similarly to the temporal perspective (cf. Chapter 6), most research proposals focus on extending the core of the BPMN notation to improve its expressiveness with respect to the design of healthcare processes, often introducing new elements that are tailored for modeling specific clinical circumstances.

In [117], the authors propose a conservative extension of BPMN for the healthcare domain, with the aim of incorporating role information and facilitate the assignment of tasks to performing resources.

A major extension to BPMN, proposed to systematically support the design of clinical pathways, is developed in [281]. This extension, named BPMN4CP, incorporates concepts retrieved from the context of clinical pathways, with the aim of enhancing evidence-based decision activities related to the considered domain. New tasks types are added to model medical diagnosis and therapies, whilst different data objects types are used for representing documents related to both medical and administrative procedures.

Conceptual modeling of care pathways is addressed under different viewpoints in [204] and [147]. Specifically, in [204] a novel methodology for connecting processes and data belonging to a healthcare information system is proposed. In [147], a data-aware representation of BPMN processes is presented as a means to support clinicians working in ICU to understand their responsibilities in the care process, how therapies are temporally constrained and which are the critical decisions that affect the quality of the provided care.

The processes designed in [147], have been further refined and used in [149] as a blueprint for guiding the development of a rule-based decision-support system for antibiotic stewardship.

Finally, in [242] two clinical pathways for colon and rectum carcinoma are designed as part of a pilot project, with the aim of evaluating the benefits of using BPMN for modeling structured medical procedures.

A general methodology for the design of medical processes was recently presented in [282]. The authors present an approach based on three main phases for healthcare process modeling, starting from context analysis and addressing the design of conceptual and logical process models. UML Activity Diagrams

are chosen to represent processes, in order to ease integration with other UML diagrams, suitable for representing other aspects of the clinical domain.

Despite the motivations described in [282] are similar to those addressed in Chapter 12, the two methodologies have different focuses. In [282] the integration of medical processes and health information systems plays a central role, whereas the modeling and management of decisions is not specifically addressed. Moreover, authors provide a mapping towards logical formalisms, while we focused mostly on conceptual design, yet completing it with organizational analysis based on KPIs and process simulation.

In general, all the presented approaches are directed to expert modelers and aim at improving understanding and, in some cases, standardization of care procedures under an organizational standpoint.

Modeling Decisions related to Healthcare Processes

In healthcare, user decisions and unpredictable events contribute to the continuous generation of new clinical knowledge, thus making the structure of the processes less rigid. The previously introduced BPMN-based approaches, do not explicitly deal with the design of process-related decisions, as these either tend to remain hidden within other process elements or are managed by dedicated decision support systems.

In general, clinical decision support is often directed to medical personnel, whereas decision management is useful to serve organizational purposes.

DMN has been proposed to model different aspects of decision-making and to complement BPMN for decision design, following the well-known “separation of concerns” principle [46], as discussed in Sect. 13.1.

At the best of our knowledge, DMN has not yet been applied for modeling clinical decisions, despite the suitability of DRDs for the design and sharing of human decision-making. Indeed, decisions in healthcare are complex, shared among different professionals, and often rely on medical evidence and expertise. Despite the formalization of these aspects is quite complex, explanatory decision requirements diagrams can be used for compliance, standardization, and training and educational purposes. Last but not least, approaches such as the one presented in [283] seem promising for improving the representation and formalization of domain-knowledge underlying DMN decision tables.

Healthcare Process Re-engineering

Business process re-engineering requires a clear understanding and a radical redesign of organizational procedures, in order to overcome critical steps and to improve process performance, flexibility, and adaptability to change. Most re-engineering approaches focus on re-thinking the process control flow, in order to allow parallel execution of activities and resource re-allocation with the aim of reducing execution times and costs. In this scenario, data and decisions are often managed and re-engineered independently and only information related to process metrics is employed for this scope.

Examples of canonical applications of process and knowledge re-engineering in healthcare are discussed in [244, 284, 285]. In detail, in [244] a framework based on event driven process chains, the entity-relationship model, and discrete event simulation is introduced in order to define and analyze the current state of a surgical ward and design an improved system.

A more general study addressing the identification of those business process re-engineering principles that can improve the application of E-Health technologies within a healthcare environment is conducted in [284].

Modeling Computer Interpretable Guidelines

A last important group of approaches that must be considered in comparison with the methodology presented in Chapter 12 is the one dealing with the modeling of clinical practice guidelines.

For more than two decades, the community of medical informatics has been actively investigating and developing IT systems for the modeling and management of clinical practice guidelines [76]. The so-called computer-interpretable guidelines (CIGs) are formalizations of clinical practice guidelines aimed at fostering the development of guideline-based decision support systems [114].

In general, the implementation of guideline-based support systems requires timely data sharing, to ensure communication between care providers, data completeness, and correct formalization of guideline rules and statements [116]. In addition, proper clinical knowledge and decision management features must be provided for dealing with knowledge-intensive care processes.

According to [286], the following areas of development must be considered when modeling clinical guidelines: (i) modeling and representation, (ii) acquisition, (iii) verification and testing, and (iv) execution and monitoring. The most relevant CIGs formalisms are exhaustively reviewed in [114].

In the remainder, we focus on discussing workflow-based approaches and task-network models, as the structure of the latter ones can be compared to the concept of process control flow.

Workflow and process modeling techniques have been applied for specifying, designing and implementing procedural aspects of clinical guidelines. One of the most complete projects addressing the integration of clinical guidelines into organizational workflows is the GUIDE project, developed within the Careflow framework [287]. GUIDE is a multi-level architecture designed to integrate a formalized model of the medical knowledge retrieved from clinical guidelines with workflow management systems. The presented approach is based on computational formalisms that facilitate the work of healthcare stakeholders by suggesting them the medical task to execute and the resources to allocate.

In [125], the authors introduce a temporal model for fostering the conceptual design of clinical workflows for stroke prevention and management, by addressing activity duration, delays, periodic and absolute constraints and inter-activity constraints.

As for CIGs formalisms, several languages for clinical guideline modeling have been proposed, such as GLIF [288], Asbru [289], PROforma [290], and SAGE [291].

The Guideline Interchange Format (GLIF) was proposed to ease guideline sharing [288]. The core language includes several ontological entities that allow the representation of guidelines as multi-step flowcharts that support branching logic. The current core, GLIF3, supports the modeling of guidelines at different abstraction levels, namely conceptual, computable, and implementable. An interesting aspect of GLIF3 is the hierarchical modeling of decisions, which distinguishes between automated decision steps and those that are made by a health worker. Asbru [289] enables designers to represent clinical guidelines, by considering continuous prescribed actions, and by supporting the parallel, in sequence, or ordered execution of care plans. Sub-plans and their outcomes can also be specified. PROforma [290] allows one to represent guidelines as directed graphs of tasks, which can be either plans, decisions, actions, and enquiries. Scheduling and temporal constraints define task ordering, and task execution is based on the triggering of pre-/post-conditions. Finally, SAGE [291] builds upon previously introduced work [288, 289, 290] and incorporates workflow representation, information and terminology standards, and control flow standards. Likewise GUIDE [287], it fosters the definition of an organizational model, with resources and roles. Such model and the guideline model are kept separate, to ease the adaptation of the guideline to local organizational realities.

A recent architectural framework for promoting the design, implementation, and evaluation of continuous guideline-based decision support is presented in [292]. A decision support engine, called PICARD, interacts with a guideline knowledge library and a temporal reasoning service in order to apply medical knowledge to patient data during the execution of clinical tasks. This framework provides alerts and recommendations at the point of care to health workers, patients, and knowledge engineers. In detail, patients treated in home-care settings can connect via a mobile client application to obtain personalized recommendations. Care providers such as physicians or nurses receive alerts, or therapy recommendations based on telemedicine data. Knowledge engineers can perform simulations on different guideline-related scenarios. In addition, the architecture allows accesses for the retrieval of procedural knowledge, usually expressed in terms of workflows. Currently, PICARD is the backbone for MobiGuide [293], an intelligent decision-support system for patients affected by chronic diseases. Briefly, patients are provided with sensors, which collect real-time information that is integrated with clinical history data and medical knowledge in order for supporting better decision-making. Alerts and personalized guideline-based recommendations are sent to patients, through their mobile phones.

Generally speaking, despite both CIGs and BPM approaches aim to improve healthcare processes and decision-making, we discerned a few crucial differences.

CIGs are meant to be used by clinicians and, thus, they focus on clinical decision support and on the development of solutions tailored for dealing with a specific clinical process, enacted in a well-defined context. On the contrary,

BPM approaches in healthcare aim to foster standard-based tools for supporting broader organizational aspects in healthcare, handled by IT and organization experts.

In Chapter 12, we proposed an agile design methodology, based on the well-known BPMN and DMN standards, addressing the integrated yet complimentary design, implementation, and enactment of care processes and related decisions, considering the viewpoint of an expert modeler.

The introduced step-wise approach promotes the iterative and agile design of organizational aspects of care pathways, mainly by focusing on the standardization of processes and decisions. The methodology supports incremental refinements, as the modeler can choose how and how many steps to apply, depending on the final objectives. Procedural, decisional, and informational aspects of care pathways are represented in an integrated and scalable format, thus supporting the re-use of clinical data and knowledge for process and decision re-engineering.

Compared to the discussed contributions belonging to the BPM field, the methodological approach introduced in Chapter 12 utilizes conceptual modeling techniques for purposes that go beyond clinical process documentation and information sharing. Indeed, simulation and indicator-based process analysis are discussed, and the role of data in process management is promoted to first citizen in the methodology.

The integration of standard process modeling techniques with systematic decision and information design, possibly used for re-engineering purposes, is a novelty of the introduced methodological framework. In addition, the methodology is presented at a quite high level, thus allowing the integration of other approaches, suitable to meet specific requirements. For instance, the use of BPMN and DMN facilitates the integration of CMMN models, in case there is the need to deal with more flexible clinical aspects. Similarly, standard ontologies and terminologies can be integrated in BPMN and DMN, for defining domain-specific concepts [283].

For instance, we suggest that standards such as HL7 [294], ICD9-CM [295], or SNOMED CT [296] can be used to encode knowledge in DMN requirement diagrams. In addition, they could be used as values for FEEL expressions, as FEEL supports several kinds of data structures and its semantics draws inspiration from Java, JavaScript, XPath, SQL, PMML, Lisp, and many others [47]. In particular, FEEL is thought to be tool-independent, executable, and expressive enough to support all kinds of decision logic.

Whereas traditional process management techniques are suitable for representing *organizational* aspects and procedural process knowledge, computer-interpretable guidelines approaches aim to support and facilitate appropriate decision-making at the point of care. Usually, the main goal of such systems is to provide direct decision support to clinicians, by fostering clinical knowledge representation. For this reason, CIGs are suitable to capture complex knowledge-based aspects of *clinical* and medical treatment processes. However, the main drawback of CIGs approaches is their lack of standardization [76, 114], which motivated the introduction of workflow-based approaches for better modeling

organizational aspects [287, 291]. Indeed, despite the abundance of language proposals, no specific proposal emerges over the others, and each approach focuses on a selected set of modeling aspects.

The goal of the methodology proposed in Chapter 12 is to start from a standard, modular, and extensible core, that allows one to exploit existing standard tools for healthcare process modeling.

The benefits of using standard tools are manifold. First of all, process and decision modeling are based on graphical and understandable notations, that can be conservatively extended and integrated with other (standard) approaches to suit the need for flexibility required in the healthcare domain. For instance, new types of tasks can be created in conformance to the standard, as done in [281], and process cases can be added to handle run-time variability [280]. Secondly, process exchange, deployment, and verification are supported by several tools, which may already be used by the administrative counterpart of healthcare organizations. Besides, modelers are facilitated by the existence of numerous control-flow, data, temporal, and resource design patterns [17, 52, 249, 297].

In general, BPMN is a powerful language for capturing various procedural aspects, as it provides a wide range of constructs suitable for advanced event and exception handling. Especially in the context of temporal constraints, researchers are actively proposing approaches for the verification of a process temporal properties and for the resolution of related violations. Motivated by processes taken from the domain of emergency medicine, in [27] the authors propose an approach for representing and verifying temporal constraints. These are expressed by extending a basic BPM modeling notation and suitable algorithms have been proposed to verify dynamic properties of temporally-enhanced business processes. TNest is a structured and modular workflow modeling language providing full support of temporal constraint specification during process design, presented in [162]. In particular, TNest allows one to define two main kinds of temporal constraints, that is, activity duration and relative constraints. Finally, in [30] temporal constraints in the context of modularized processes are investigated. In this setting, the re-use of process knowledge must be enabled and, thus, authors consider the issue of representing the overall temporal behavior of a sub-process. This way, temporally enhanced sub-processes can be used within other processes, hiding the internal temporal features of such subprocesses.

Yet, born to capture process control-flow, BPMN does not fully support other perspectives, such as the the representation of domain knowledge and the integration of complex, structured data [51, 74]. Despite research efforts are following this direction considering clinical domains [54, 53, 279, 280] the expressiveness of CIGs is still more suitable for dealing with complex clinical aspects, especially when developing tools that require clinical knowledge representation and the direct involvement of physicians and patients [293].

Concluding Remarks and Future Work

In this chapter, we briefly summarize the main contributions of this work, by discussing limitations and outlining possible future research directions.

A business process model is the result of mapping a business process with the aim to conceptualize and explain how a certain procedure is enacted within an organization. Process modeling is an essential step of the BPM life-cycle [1] as it allows sorting out issues that otherwise may go undiscovered until run-time.

Process models span different perspectives and levels of abstraction, depending on their target audience, final objectives, and degree of automation [10]. Usually, the modeling of a process requires the gathering of different forms of information that contribute to build the so-called perspectives of a process model.

Process modeling languages differ in the extent to which their constructs highlight the information that defines a certain process modeling perspective. The perspectives that a process model is able to capture are bounded by the expressiveness of the language used for modeling [9].

Traditional activity-centric process modeling languages are centred on the control-flow perspective, yet leaving other important aspects only partially supported [12, 22, 23]. Being the reference language for activity-centric process modeling, BPMN is no exception to these limitations [70].

The research presented in this thesis is focused on improving the modeling of different process perspectives in BPMN process models. Motivated by real-world healthcare scenarios, we chose to address the modeling of the temporal, informational, and decision perspectives among others.

We intentionally leave out important perspectives, such as the organizational one (i.e., the representation of which agents in the organization perform which process activities and of the communication and transfer mechanisms included in the process), as it would be impractical to address all of them exhaustively. Indeed, improving the modeling of process perspectives requires choosing which aspects of one perspective to address, how to deal with them, and at which level of abstraction. Last but not least, prior research must be considered and the latter varies depending on how much attention one perspective and the related open issues have gathered within the research community.

In relation to Part II – Part IV, we partition the contributions of this thesis according to the process modeling perspective they concern.

The *temporal perspective*, considered in Part II, deals with the specification and verification of temporal constraints during business process design and management. The lack of direct support of BPMN for the representation of time constraints is remarked by recent research approaches [17, 34, 35, 36, 119], some of which propose to extend the notation with graphical constructs and attributes to capture temporal constraints. Extending BPMN to increase support for the managing temporal aspects is one way towards improving the modeling of the temporal perspective. Another way is to investigate whether BPMN and the advanced modeling constructs it includes can be used to capture such time aspects, without the need of introducing novel extensions.

In this thesis, we followed the first approach in Chapter 5 and the second one in Chapter 3 and Chapter 4. Then, we consider mapping BPMN onto formal models, i.e., time Petri nets [65], timed automata [66], and CSTNUDs [67] to either validate the semantics of the obtained process models or to verify their dynamic controllability.

In Chapter 3 we present a pattern-based approach to model different nuances of duration constraints in BPMN. Our proposal relies on the modularity of well-structured process models [108] and on event handling mechanisms, which are already available in BPMN.

The main advantage of our proposal is the full compliance with existing BPMN modeling and simulation tools, while the major drawback is probably the intricacy of the obtained models, which are not easy to read for non-expert process designers. To overcome this limitation, a possible future step is the extension of BPMN by adding novel duration-aware task types, having the advantage of being semantically BPMN-compliant. In addition, we aim to investigate whether other kinds of temporal constraints, such as restrictions of process execution points [17], may be suitably modeled with the same method.

In Chapter 4, we followed an approach similar to the one presented in Chapter 3 for specifying different kinds of duration constraints. The approach introduced in Chapter 4 makes an even more sophisticated use of BPMN signal events to synchronize the main process with the constructs added to specify temporal aspects. Again, the price to pay for using BPMN in such a technical way is the size of the obtained process models, which nonetheless rely on a well-specified semantics. The main BPMN constructs are mapped onto networks of timed automata to formally validate the execution behavior of the obtained process models.

The proposed mapping sets also the bases for design-time verification of temporal constraints. Indeed, for future work we could adapt the proposed mapping to conform with the UPPAAL modeling language [123] and exploit the simulation and model checking features of the tool.

Instead, in Chapter 5 we start by extending BPMN with graphical symbols to capture temporal constraints, such as duration of activities, waiting times of

events, and inter-activity constraints, and to distinguish two kinds of decision activities based on how their outcomes are managed at run-time. This novel characterization of decision activities into *observations* and *guided decisions* shows the relationship between the temporal and the decision perspectives of a business process, and affects the way time aspects are managed at run-time.

The formal mapping onto CSTNUDs is provided to allow one checking whether the designed time-aware BPMN process is dynamically controllable, that is, whether there exist a way to satisfy all relevant time constraints regardless of which observation outcomes and activity durations are revealed over time.

Part III deals with the *informational perspective* of business processes, which is probably one of the “hot topics” in BPM [43, 45, 19, 74, 83, 177]. Since data affect each phase of the business process life cycle, existing approaches have focused on improving the connection of process and data models to foster process design, automated execution, monitoring, and formal verification.

In the context of process design, organizations have acknowledged the need of representing and making explicit the contents of process and database models. However, at the conceptual level, these two important assets are often conceived and managed independently [45].

This thesis contributes to shrink the conceptual gap between BPMN process models and UML class diagrams representing the conceptual schema of a database whose data are manipulated by one or more processes.

The Activity View, introduced in Chapter 7, serves this purpose and allows one to visualize the data operations that are performed by process activities on a portion of a database schema, at the conceptual level. By properly combining the information included in the Activity Views of one process, designers are provided with an overview of how data are used within a process.

This basic information may be further analyzed to obtain interesting insights, useful for improving communication with stakeholders, data support for existing processes, and process re-engineering. To this end, in Chapter 7 we describe a relational framework that shows how relational calculus and SQL may be exploited to discover complex relationships of connected process and data models.

The Activity View paves the way for multiple future work directions.

First of all, being tailored to process activities, the Activity View may be adapted to capture different levels of process abstractions, such as subprocesses and complete processes. However, going from an Activity View to a “Subprocess View” is not straightforward, as it requires dealing with the inner structure of the sub-process for determining access-times and number of accessed data instances.

Another desirable future direction is the integration of the Activity View into business process editors that support the design or importation of database schemata, such as the Gryphon tool [298]. In order to implement the Activity View in process modeling tools, we refine its graphical representation in such a way to facilitate the communication with stakeholders and data managers. The prototypical implementation discussed at the end of Chapter 7 and the results of

the experimental evaluation described in Chapter 9 are a starting point in this direction, but further requirements analyses must still be conducted.

Finally, it is worth noticing that the Activity View in its current form addresses the conceptual modeling of persistent data sources that are *external* to the process itself. That is, volatile data that are part of the process, such as for instance process variables, are not considered in this thesis. In this scenario, finding a way to model inherent process data and to relate the latter with the Activity View is an interesting prospective direction.

In Chapter 8 we make use of a simplified version of the Activity View to discover inconsistencies arising between data access operations performed by different process activities. Starting from potential inconsistencies, we presented an algorithm that works on process traces to detect erratic sequences of data operations performed by the process on the same data instance.

Also this approach uncovers possible future improvements. First of all, considering the complete version of the Activity View presented in Chapter 7 increases the number of ingredients that should be considered when defining inconsistencies. Secondly, the repair of data inconsistencies is yet to be addressed.

Finally, the *decision perspective* of process models is considered in Part IV, giving particular attention to the interplay between the BPMN and DMN standards, and to requirements coming from real-world healthcare domains.

In Chapter 11 we propose a pattern-based approach to ease the derivation of DMN decision models from the data perspective of BPMN process models. The proposed method complements the one introduced in [20] for the derivation of decisions from the process control-flow. Our proposal could benefit from a semi-automatic way to demarcate which among user and business rule tasks are decision activities. Besides, taking a look at domain knowledge underlying BPMN processes, further connections with DMN decision requirements diagrams could be provided.

Chapter 12 centres on a methodology for the combined use of BPMN and DMN for decision-intensive care pathways. Being inspired by the traditional BPM life-cycle, the proposed methodology may be easily extended to include several other aspects beside the integrated modeling of processes and decisions, thus providing an overview of how different modeling perspectives may be intertwined to deal with complex healthcare domains.

Indeed, in Chapter 12 we mention contact points with some existing approaches for the modeling of temporal and data aspects, but the design phase may be further detailed to span the management of organizational aspects and constraints.

To conclude, healthcare and clinical working environments are the preferred application domains for inspiring and exemplifying the concepts developed in this thesis. The reasons behind this choice lie in the inherent complexity of clinical and healthcare processes [26, 51], that makes the representation of the temporal and decision perspectives especially difficult.

In accordance with [13], we think that general purpose approaches able to deal with the complexity of healthcare processes may be easily applied to other process scenarios.

Publications

The ideas presented in this thesis are based on the following publications. Alphabetical order of authors is the policy we adopted for papers co-authored by my advisors at the University of Verona. All other papers follow a contribution order.

Peer-reviewed Journals

- Carlo Combi, Barbara Oliboni, Francesca Zerbato. “A Modular Approach to the Specification of Time Duration Constraints in BPMN” *Information Systems*. Accepted for publication.
- Ekaterina Bazhenova, Francesca Zerbato, Barbara Oliboni, Mathias Weske. “From BPMN process models to DMN decision models” *Information Systems* 83 (2019): 69–88. doi:[10.1016/j.is.2019.02.001](https://doi.org/10.1016/j.is.2019.02.001)
- Carlo Combi, Barbara Oliboni, Alessandro Zardini, Francesca Zerbato. “A Methodological Framework for the Integrated Design of Decision-Intensive Care Pathways—an Application to the Management of COPD Patients”. *Journal of Healthcare Informatics Research* 1.2 (2017): 157–217. doi:[10.1007/s41666-017-0007-4](https://doi.org/10.1007/s41666-017-0007-4)

Peer-reviewed Conferences

- Combi Carlo, Barbara Oliboni, Mathias Weske, Francesca Zerbato. “Conceptual Modeling of Processes and Data: Connecting Different Perspectives.” In *International Conference on Conceptual Modeling (ER)*, Vol. 11157 of LNCS, pages 236–250. Springer, Cham, 2018. doi:[10.1007/978-3-030-00847-5_18](https://doi.org/10.1007/978-3-030-00847-5_18)
- Roberto Posenato, Francesca Zerbato, Carlo Combi. “Managing decision tasks and events in time-aware business process models.” In *Business Process Management (BPM)*, Vol. 11080 of LNCS, pages 102–118. Springer, Cham, 2018. doi:[0.1007/978-3-319-98648-7_7](https://doi.org/10.1007/978-3-319-98648-7_7)

- Carlo Combi, Barbara Oliboni, Mathias Weske, Francesca Zerbato. “Conceptual modeling of inter-dependencies between processes and data.” In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, pages 110–119. ACM, 2018. doi:[10.1145/3167132.3167141](https://doi.org/10.1145/3167132.3167141)
- Carlo Combi, Barbara Oliboni, Francesca Zerbato. “Towards dynamic duration constraints for therapy and monitoring tasks.” In *Artificial Intelligence in Medicine (AIME 2017)*, Vol. 10259 of LNCS, pages 223–233. Springer International Publishing, 2017. doi:[10.1007/978-3-319-59758-4_25](https://doi.org/10.1007/978-3-319-59758-4_25)
- Carlo Combi, Barbara Oliboni, Francesca Zerbato. “Modeling and handling duration constraints in BPMN 2.0.” In *Proceedings of the 32nd Annual ACM Symposium on Applied Computing*, SAC '17, pages 727 – 734. ACM, 2017. doi:[10.1145/3019612.3019618](https://doi.org/10.1145/3019612.3019618)
- Carlo Combi, Pietro Sala, Francesca Zerbato. “Driving time-dependent paths in clinical BPMN processes.” In *Proceedings of the 32nd Annual ACM Symposium on Applied Computing*, SAC '17, pages 743 – 750. ACM, 2017. doi:[10.1145/3019612.3019620](https://doi.org/10.1145/3019612.3019620)
- Carlo Combi, Barbara Oliboni, Alessandro Zardini, Francesca Zerbato. “Seamless design of decision-intensive care pathways.” In *IEEE International Conference on Healthcare Informatics (ICHI 2016)*, pages 35–45. IEEE, 2016. **Best Paper Award Winner**. doi:[10.1109/ICHI.2016.9](https://doi.org/10.1109/ICHI.2016.9)

Peer-reviewed Workshops

- Ekaterina Bazhenova, Francesca Zerbato, Mathias Weske. “Data-Centric Extraction of DMN Decision Models from BPMN Process Models.” In *Business Process Management Workshops*, Vol. 308 of LNBIP, pages 542–555. Springer International Publishing, 2018. doi:[10.1007/978-3-319-74030-0_43](https://doi.org/10.1007/978-3-319-74030-0_43)

In addition to the above publications presented in this thesis, I was also involved in the following research as part of my doctoral studies:

Book Chapters

- Carlo Combi, Barbara Oliboni, Giuseppe Pozzi and Francesca Zerbato. “Models and Architectures for the Enactment of Healthcare Processes.” In *Process Modelling and Management for Healthcare*. Edited by Carlo Combi, Giuseppe Pozzi, Pierangelo Veltri, 3–33. CRC Press – Taylor and Francis Group, 2017.

Peer-reviewed Conferences

- Carlo Combi, Pietro Sala, and Francesca Zerbato. “A logical formalization of time-critical processes with resources.” In *Business Process Management Forum (BPM Forum)*, Vol. 329 of LNBIP, pages 20–36. Springer, 2018. doi:[10.1007/978-3-319-98651-7_2](https://doi.org/10.1007/978-3-319-98651-7_2)
- Bernardo Canovas-Segura, Francesca Zerbato, Barbara Oliboni, Carlo Combi, Manuel Campos, Antonio Morales, Jose M. Juarez, Roque Marin, and Francisco Palacios. “A process-oriented approach for supporting clinical decisions for infection management.” In *IEEE International Conference on Healthcare Informatics (ICHI 2017)*, pages 91–100. IEEE, 2017. doi:[10.1109/ICHI.2017.73](https://doi.org/10.1109/ICHI.2017.73)
- Francesca Zerbato, Barbara Oliboni, Carlo Combi, Manuel Campos, and Jose M Juarez. “BPMN-based representation and comparison of clinical pathways for catheter-related bloodstream infections.” In *IEEE International Conference on Healthcare Informatics (ICHI 2015)*, pages 346–355. IEEE, 2015. doi:[10.1109/ICHI.2015.49](https://doi.org/10.1109/ICHI.2015.49)

List of Acronyms

BPM	business process management
BPMN	Business Process Model and Notation
CIG	computer-interpretable guideline
CMMN	Case Management Model and Notation
COPD	chronic obstructive pulmonary disease
CSTNUD	Conditional Simple Temporal Network with Uncertainty and Decisions
DCDS	data-centric dynamic system
DMN	Decision Model and Notation
DRD	Decision Requirement Diagram
DRG	Decision Requirement Graph
EPC	event-driven process chains
FEEL	Friendly Enough Expression Language
GSM	Guard-Stage-Milestone
HCQI	healthcare quality indicators
IT	information technology
KPI	key performance indicator
OMG	Object Management Group
PAIS	process-aware information systems
SESE	Single-Entry-Single-Exit
SQL	Structured Query Language
TA	timed automata
TGA	timed game automata
TPN	time Petri nets
UML	Unified Modeling Language

YAWL Yet Another Workflow Language

References

1. M. Weske, *Business Process Management: Concepts, Languages, Architectures*, Springer-Verlag, Berlin, Heidelberg, 2012. doi:[10.1007/978-3-642-28616-2](https://doi.org/10.1007/978-3-642-28616-2).
2. W. M. Van Der Aalst, *Business process management: a comprehensive survey*, ISRN Software Engineering 2013. doi:[10.1155/2013/507984](https://doi.org/10.1155/2013/507984).
3. J. Mendling, *Metrics for process models: empirical foundations of verification, error prediction, and guidelines for correctness*, Vol. 6, Springer Science & Business Media, 2008.
4. M. Dumas, W. M. Van der Aalst, A. H. Ter Hofstede, *Process-aware information systems: bridging people and software through process technology*, Vol. 1, John Wiley & Sons, New York, NY, USA, 2005.
5. G. M. Giaglis, *A taxonomy of business process modeling and information systems modeling techniques*, *International Journal of Flexible Manufacturing Systems* 13 (2) (2001) 209–228. doi:[10.1023/A:1011139719773](https://doi.org/10.1023/A:1011139719773).
6. W. M. Van Der Aalst, A. H. Ter Hofstede, M. Weske, *Business process management: A survey*, in: *International Conference on Business Process Management (BPM)*, Vol. 2678 of LNCS, Springer, 2003, pp. 1–12. doi:[10.1007/3-540-44895-0_1](https://doi.org/10.1007/3-540-44895-0_1).
7. H. Mili, G. Tremblay, G. B. Jaoude, É. Lefebvre, L. Elabed, G. E. Boussaidi, *Business process modeling languages: Sorting through the alphabet soup*, *ACM Computing Surveys (CSUR)* 43 (1) (2010) 4. doi:[10.1145/1824795.1824799](https://doi.org/10.1145/1824795.1824799).
8. J. L. Pereira, D. Silva, *Business process modeling languages: A comparative framework*, in: Á. Rocha, A. M. Correia, H. Adeli, L. P. Reis, M. Mendonça Teixeira (Eds.), *New Advances in Information Systems and Technologies*, Springer International Publishing, Cham, 2016, pp. 619–628. doi:[10.1007/978-3-319-31232-3_58](https://doi.org/10.1007/978-3-319-31232-3_58).
9. B. Curtis, M. I. Kellner, J. Over, *Process modeling*, *Commun. ACM* 35 (9) (1992) 75–90. doi:[10.1145/130994.130998](https://doi.org/10.1145/130994.130998).
10. J. Krogstie, *Perspectives to process modeling*, in: M. Glykas (Ed.), *Business Process Management: Theory and Applications*, Springer, Berlin, Heidelberg, 2013, pp. 1–39. doi:[10.1007/978-3-642-28409-0_1](https://doi.org/10.1007/978-3-642-28409-0_1).

11. Object Management Group, Business Process Model and Notation (BPMN), v2.0.2, available at: <http://www.omg.org/spec/BPMN/2.0.2/> (December 2013).
12. M. Zur Muehlen, J. Recker, How much language is enough? Theoretical and practical use of the business process modeling notation, in: International Conference on Advanced Information Systems Engineering (CAiSE), Springer, 2008, pp. 465–479. doi:[10.1007/978-3-540-69534-9_35](https://doi.org/10.1007/978-3-540-69534-9_35).
13. A. Jiménez-Ramírez, I. Barba, M. Reichert, B. Weber, C. Del Valle, Clinical processes - the killer application for constraint-based process interactions, in: Proceedings of 30th International Conference on Advanced Information Systems Engineering (CAiSE), LNCS, Springer International Publishing, Cham, 2018, pp. 374–390. doi:[10.1007/978-3-319-91563-0_23](https://doi.org/10.1007/978-3-319-91563-0_23).
14. M. Reichert, B. Weber, Enabling flexibility in process-aware information systems: challenges, methods, technologies, Springer-Verlag Berlin Heidelberg, 2012. doi:[10.1007/978-3-642-30409-5](https://doi.org/10.1007/978-3-642-30409-5).
15. M. L. Rosa, M. Dumas, A. H. ter Hofstede, J. Mendling, Configurable multi-perspective business process models, *Information Systems* 36 (2) (2011) 313 – 340, special Issue: Semantic Integration of Data, Multimedia, and Services. doi:<https://doi.org/10.1016/j.is.2010.07.001>.
16. J. Eder, E. Panagos, M. Rabinovich, Time constraints in workflow systems, in: International Conference on Advanced Information Systems Engineering (CAiSE), Springer, 1999, pp. 286–300. doi:[10.1007/3-540-48738-7_22](https://doi.org/10.1007/3-540-48738-7_22).
17. A. Lanz, B. Weber, M. Reichert, Workflow time patterns for process-aware information systems, in: Enterprise, Business-Process and Information Systems Modeling, Vol. 50 of LNBIP, Springer, Berlin, Heidelberg, 2010, pp. 94–107. doi:[10.1007/978-3-642-13051-9_9](https://doi.org/10.1007/978-3-642-13051-9_9).
18. S. Cheikhrouhou, S. Kallel, N. Guermouche, M. Jmaiel, The temporal perspective in business process modeling: a survey and research challenges, *Service Oriented Computing and Applications* 9 (1) (2015) 75–85. doi:[10.1007/s11761-014-0170-x](https://doi.org/10.1007/s11761-014-0170-x).
19. A. Meyer, L. Pufahl, D. Fahland, M. Weske, Modeling and enacting complex data dependencies in business processes, in: International Conference on Business Process Management (BPM), Vol. 8094 of LNCS, Springer, 2013, pp. 171–186. doi:[10.1007/978-3-642-40176-3_14](https://doi.org/10.1007/978-3-642-40176-3_14).
20. K. Batoulis, A. Meyer, E. Bazhenova, G. Decker, M. Weske, Extracting decision logic from process models, in: Proceedings of the 27th International Conference on Advanced Information Systems Engineering (CAiSE), Vol. 9097 of LNCS, Springer, 2015, pp. 349–366. doi:[10.1007/978-3-319-19069-3_22](https://doi.org/10.1007/978-3-319-19069-3_22).
21. B. Von Halle, L. Goldberg, The decision model: a business logic framework linking business and technology, CRC Press, 2009.
22. P. Wohed, W. M. van der Aalst, M. Dumas, A. H. ter Hofstede, N. Russell, Pattern-based analysis of BPMN – an extensive evaluation of the control-flow, the data and the resource perspectives, Tech. rep., BPM Center Report (2006).
23. A. Lanz, B. Weber, M. Reichert, Time patterns for process-aware information systems, *Requirements Engineering* 19 (2) (2014) 113–141. doi:[10.1007/s00766-012-0162-3](https://doi.org/10.1007/s00766-012-0162-3).

24. Signavio GmbH, Signavio Business Transformation Suite, available at: www.signavio.com.
25. Camunda services GmbH, Camunda BPM Platform, available at: <https://camunda.org>.
26. P. Dadam, M. Reichert, K. Kuhn, Clinical Workflows — The Killer Application for Process-oriented Information Systems?, Springer London, London, 2000. doi: [10.1007/978-1-4471-0761-3_3](https://doi.org/10.1007/978-1-4471-0761-3_3).
27. A. Kumar, R. R. Barton, Controlled violation of temporal process constraints – models, algorithms and results, *Information Systems* 64 (2017) 410 – 424. doi: [10.1016/j.is.2016.06.003](https://doi.org/10.1016/j.is.2016.06.003).
28. W. M. van der Aalst, M. Rosemann, M. Dumas, Deadline-based escalation in process-aware information systems, *Decision Support Systems* 43 (2) (2007) 492–511. doi: [10.1016/j.dss.2006.11.005](https://doi.org/10.1016/j.dss.2006.11.005).
29. C. Combi, R. Posenato, Controllability in temporal conceptual workflow schemata, in: *International Conference on Business Process Management (BPM)*, 2009, pp. 64–79. doi: [10.1007/978-3-642-03848-8_6](https://doi.org/10.1007/978-3-642-03848-8_6).
30. A. Lanz, R. Posenato, C. Combi, M. Reichert, Controlling time-awareness in modularized processes, in: R. Schmidt, W. Guédria, I. Bider, S. Guerreiro (Eds.), *Enterprise, Business-Process and Information Systems Modeling*, Springer International Publishing, Cham, 2016, pp. 157–172. doi: [10.1007/978-3-319-39429-9_11](https://doi.org/10.1007/978-3-319-39429-9_11), .
31. S. Cheikhrouhou, S. Kallel, N. Guermouche, M. Jmaiel, Enhancing formal specification and verification of temporal constraints in business processes, in: *IEEE Int. Conf. Serv. Comput. (SCC'14)*, 2014, pp. 701–708. doi: [10.1109/SCC.2014.97](https://doi.org/10.1109/SCC.2014.97).
32. K. Watahiki, F. Ishikawa, K. Hiraishi, Formal verification of business processes with temporal and resource constraints, in: *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2011, IEEE, 2011, pp. 1173–1180. doi: [10.1109/ICSMC.2011.6083857](https://doi.org/10.1109/ICSMC.2011.6083857).
33. A. Lanz, M. Reichert, B. Weber, Process time patterns: A formal foundation, *Information Systems* 57 (C) (2016) 38–68. doi: [10.1016/j.is.2015.10.002](https://doi.org/10.1016/j.is.2015.10.002).
34. D. Gagne, A. Trudel, Time-BPMN, in: *IEEE Conference on Commerce and Enterprise Computing*, 2009, IEEE, 2009, pp. 361–367. doi: [10.1109/CEC.2009.71](https://doi.org/10.1109/CEC.2009.71).
35. S. Cheikhrouhou, S. Kallel, N. Guermouche, M. Jmaiel, Toward a time-centric modeling of business processes in BPMN 2.0, in: *Proceedings of International Conference on Information Integration and Web-based Applications & Services, II-WAS '13*, ACM, New York, NY, USA, 2013, pp. 154–163. doi: [10.1145/2539150.2539182](https://doi.org/10.1145/2539150.2539182).
36. C. Arevalo, M. J. Escalona, I. Ramos, M. Domínguez-Muñoz, A metamodel to integrate business processes time perspective in BPMN 2.0, *Information and Software Technology* 77 (Supplement C) (2016) 17–33. doi: [10.1016/j.infsof.2016.05.004](https://doi.org/10.1016/j.infsof.2016.05.004).
37. M. Dumas, On the convergence of data and process engineering, in: J. Eder, M. Bielikova, A. M. Tjoa (Eds.), *Advances in Databases and Information Systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 19–26. doi: [10.1007/978-3-642-23737-9_2](https://doi.org/10.1007/978-3-642-23737-9_2).

38. M. Reichert, Process and Data: Two Sides of the Same Coin?, in: 20th International Conference on Cooperative Information Systems, Springer, 2012, pp. 2–19. doi:10.1007/978-3-642-33606-5_2.
39. R. Hull, Artifact-centric business process models: Brief survey of research results and challenges, in: OTM Confederated International Conferences “On the Move to Meaningful Internet Systems”, Springer, 2008, pp. 1152–1163. doi:10.1007/978-3-540-88873-4_17.
40. B. Bagheri Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, M. Montali, Verification of relational data-centric dynamic systems with external services, in: Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems, ACM, 2013, pp. 163–174. doi:10.1145/2463664.2465221.
41. R. Hull, E. Damaggio, F. Fournier, M. Gupta, F. H. III, S. Hobson, M. H. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, R. Vaculín, Introducing the guard-stage-milestone approach for specifying business entity lifecycles, in: Web Services and Formal Methods - 7th International Workshop, WS-FM 2010, Hoboken, NJ, USA, September 16-17, 2010. Revised Selected Papers, 2010, pp. 1–24. doi:10.1007/978-3-642-19589-1_1.
42. Y. Sun, J. Su, B. Wu, J. Yang, Modeling data for business processes, in: 30th International Conference on Data Engineering (ICDE), IEEE, 2014, pp. 1048–1059. doi:10.1109/ICDE.2014.6816722.
43. A. Meyer, S. Smirnov, M. Weske, Data in business processes, Tech. rep. (2011).
44. A. Koutsos, V. Vianu, Process-centric views of data-driven business artifacts, Journal of Computer and System Sciences 86 (2017) 82 – 107. doi:10.1016/j.jcss.2016.11.012.
45. G. De Giacomo, X. Oriol, M. Estañol, E. Teniente, Linking Data and BPMN Processes to Achieve Executable Models, in: 29th International Conference on Advanced Information Systems Engineering (CAiSE), Vol. 10253 of LNCS, Springer, 2017, pp. 612–628. doi:10.1007/978-3-319-59536-8_38.
46. W. L. Hürsch, C. V. Lopes, Separation of concerns, Tech. rep. (1995).
47. Object Management Group, Decision Model and Notation (DMN), v1.1, available at: <http://www.omg.org/spec/DMN/1.1/> (December 2016).
48. T. Biard, A. Le Mauff, M. Bigand, J.-P. Bourey, Separation of decision modeling from business process modeling using new Decision Model and Notation (DMN) for automating operational decision-making, in: Working Conference on Virtual Enterprises, Springer, 2015, pp. 489–496. doi:10.1007/978-3-319-24141-8_45.
49. F. Hasić, L. Devadder, M. Dochez, J. Hanot, J. De Smedt, J. Vanthienen, Challenges in refactoring processes to include decision modelling, in: Business Process Management Workshops, Springer International Publishing, Cham, 2018, pp. 529–541. doi:10.1007/978-3-319-74030-0_42.
50. F. Hasic, J. D. Smedt, J. Vanthienen, Augmenting processes with decision intelligence: Principles for integrated modelling, Decision Support Systems 107 (2018) 1–12. doi:https://doi.org/10.1016/j.dss.2017.12.008.
51. R. Lenz, M. Reichert, IT Support for Healthcare Processes - Premises, Challenges, Perspectives, Data & Knowledge Engineering 61 (1) (2007) 39–58. doi:10.1016/j.datak.2006.04.007.

52. C. Combi, B. Oliboni, F. Zerbato, Modeling and handling duration constraints in BPMN 2.0, in: Proceedings of the ACM Symposium on Applied Computing, SAC '17, ACM, Marrakech, Morocco, 2017, pp. 727–734. doi:10.1145/3019612.3019618.
53. C. Combi, B. Oliboni, F. Zerbato, Towards dynamic duration constraints for therapy and monitoring tasks, in: Proceedings of the 16th Conference on Artificial Intelligence in Medicine (AIME 2017), Springer International Publishing, Vienna, Austria, 2017, pp. 223–233. doi:10.1007/978-3-319-59758-4_25.
54. C. Combi, P. Sala, F. Zerbato, Driving time-dependent paths in clinical BPMN processes, in: Proceedings of the ACM Symposium on Applied Computing, SAC '17, ACM, Marrakech, Morocco, 2017, pp. 743–750. doi:10.1145/3019612.3019620.
55. R. Posenato, F. Zerbato, C. Combi, Managing decision tasks and events in time-aware business process models, in: International Conference on Business Process Management (BPM), 2018, pp. 102–118. doi:10.1007/978-3-319-98648-7_7.
56. C. Combi, B. Oliboni, M. Weske, F. Zerbato, Conceptual modeling of interdependencies between processes and data, in: Proceedings of the ACM Symposium on Applied Computing (SAC), ACM, 2018, pp. 110–119. doi:10.1145/3167132.3167141.
57. C. Combi, B. Oliboni, M. Weske, F. Zerbato, Conceptual modeling of processes and data: Connecting different perspectives, in: International Conference on Conceptual Modeling (ER), Vol. 11157 of LNCS, Springer International Publishing, 2018, pp. 236–250. doi:10.1007/978-3-030-00847-5_18.
58. C. Combi, B. Oliboni, A. Zardini, F. Zerbato, A methodological framework for the integrated design of decision-intensive care pathways—an application to the management of COPD patients, *Journal of Healthcare Informatics Research* 1 (2) (2017) 157–217. doi:10.1007/s41666-017-0007-4.
59. C. Combi, B. Oliboni, A. Zardini, F. Zerbato, Seamless design of decision-intensive care pathways, in: IEEE International Conference on Healthcare Informatics (ICHI 2016), IEEE, 2016, pp. 35–45. doi:10.1109/ICHI.2016.9.
60. E. Bazhenova, F. Zerbato, M. Weske, Data-centric extraction of DMN decision models from processes, in: Business Process Management Workshops, Vol. 308 of LNBIP, Springer International Publishing, 2018, pp. 542–555. doi:10.1007/978-3-319-74030-0_43.
61. E. Bazhenova, F. Zerbato, B. Oliboni, M. Weske, From BPMN process models to DMN decision models, *Information Systems* 83 (2019) 69–88. doi:10.1016/j.is.2019.02.001.
62. Object Management Group, Case Management Modeling Notation (CMMN), v1.1, available at: <http://www.omg.org/spec/CMMN/1.1/> (December 2016).
63. Object Management Group, Unified Modeling Language, v2.5, Available at: <http://www.omg.org/spec/UML/2.5/>.
64. P. H. Morris, N. Muscettola, T. Vidal, Dynamic control of plans with temporal uncertainty, in: International Joint Conference on Artificial Intelligence (IJCAI'01), 2001, pp. 494–502.

65. B. Berthomieu, M. Diaz, Modeling and verification of time dependent systems using time Petri nets, *IEEE Transactions on Software Engineering* 17(3) (1991) 259–273. doi:10.1109/32.75415.
66. R. Alur, D. L. Dill, A theory of timed automata, *Theoretical Computer Science* 126 (2) (1994) 183 – 235. doi:10.1016/0304-3975(94)90010-8.
67. M. Zavatteri, Conditional simple temporal networks with uncertainty and decisions, in: 24th International Symposium on Temporal Representation and Reasoning (TIME'17), 2017, pp. 23:1–23:17. doi:10.4230/LIPIcs.TIME.2017.23.
68. H. Campbell, R. Hotchkiss, N. Bradshaw, M. Porteous, Integrated care pathways, *BMJ: British Medical Journal* 316 (7125) (1998) 133. doi:10.1136/bmj.316.7125.133.
69. M. Cairo, C. Combi, C. Comin, L. Hunsberger, R. Posenato, R. Rizzi, M. Zavatteri, Incorporating decision nodes into conditional simple temporal networks, in: 24th International Symposium on Temporal Representation and Reasoning (TIME'17), 2017, pp. 9:1–9:18. doi:10.4230/LIPIcs.TIME.2017.9.
70. P. Wohed, W. M. Van der Aalst, M. Dumas, A. H. ter Hofstede, N. Russell, On the suitability of BPMN for business process modelling, in: International conference on business process management, Springer, 2006, pp. 161–176. doi:10.1007/11841760_12.
71. C. A. Petri, Kommunikation mit automaten, *Schriften des IIM Nr. 2*, Institut für Instrumentelle Mathematik, Bonn.
72. W. Reisig, Petri nets: an introduction, Vol. 4, Springer Science & Business Media, 2012. doi:10.1007/978-3-642-69968-9.
73. R. M. Dijkman, M. Dumas, C. Ouyang, Semantics and analysis of business process models in BPMN, *Information and Software Technology* 50 (2008) 1281–1294. doi:10.1016/j.infsof.2008.02.006.
74. D. Calvanese, G. De Giacomo, M. Montali, Foundations of data-aware process analysis: A database theory perspective, in: Proceedings of the 32 ACM SIGMOD Symposium on Principles of Database Systems, ACM, 2013, pp. 1–12. doi:10.1145/2463664.2467796.
75. C. Di Ciccio, A. Marrella, A. Russo, Knowledge-intensive processes: Characteristics, requirements and analysis of contemporary approaches, *Journal on Data Semantics* 4 (1) (2015) 29–57. doi:10.1007/s13740-014-0038-4.
76. A. Russo, M. Mecella, On the evolution of process-oriented approaches for healthcare workflows, *International Journal of Business Process Integration and Management* 6 (3) (2013) 224–246. doi:10.1504/IJBPIIM.2013.056962.
77. W. M. van der Aalst, Verification of workflow nets, in: Application and Theory of Petri Nets 1997: 18th International Conference, (ICATPN'97), Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 407–426. doi:10.1007/3-540-63139-9_48.
78. W. M. Van der Aalst, The application of Petri nets to workflow management, *Journal of circuits, systems, and computers* 8(1) (1998) 21–66. doi:10.1142/S0218126698000043.
79. M. Kunze, M. Weske, Business Process Models, 1st Edition, Springer International Publishing, Cham, 2016. doi:10.1007/978-3-319-44960-9.

80. P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, H. A. Reijers, Imperative versus declarative process modeling languages: An empirical investigation, in: F. Daniel, K. Barkaoui, S. Dustdar (Eds.), *Business Process Management Workshops*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 383–394. doi:10.1007/978-3-642-28108-2_37.
81. G. De Giacomo, M. Dumas, F. M. Maggi, M. Montali, Declarative Process Modeling in BPMN, in: J. Zdravkovic, M. Kirikova, P. Johannesson (Eds.), *Advanced Information Systems Engineering*, Springer International Publishing, Cham, 2015, pp. 84–100. doi:10.1007/978-3-319-19069-3_6.
82. K. Jensen, G. Rozenberg, *High-level Petri nets: theory and application*, Springer Science & Business Media, 2012.
83. M. Montali, A. Rivkin, Db-nets: On the marriage of colored Petri nets and relational databases, in: *Transactions on Petri Nets and Other Models of Concurrency XII*, Vol. 10470 of LNCS, Springer Berlin Heidelberg, 2017, pp. 91–118. doi:10.1007/978-3-662-55862-1_5.
84. W. M. Van Der Aalst, A. H. Ter Hofstede, YAWL: yet another workflow language, *Information systems* 30 (4) (2005) 245–275. doi:10.1016/j.is.2004.02.002.
85. A. Scheer, O. Thomas, O. Adam, *Process Modeling Using Event-Driven Process Chains*, Wiley, 2005. doi:10.1002/0471741442.ch6.
86. J. Mendling, *Event-driven process chains (EPC)*, LNBIP, Springer-Verlag, Berlin Heidelberg, 2008. doi:10.1007/978-3-540-89224-3.
87. D. Fahland, D. Lübke, J. Mendling, H. Reijers, B. Weber, M. Weidlich, S. Zugal, Declarative versus imperative process modeling languages: The issue of understandability, in: T. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, P. Soffer, R. Ukor (Eds.), *Enterprise, Business-Process and Information Systems Modeling*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 353–366. doi:10.1007/978-3-642-01862-6_29.
88. M. Pesic, H. Schonenberg, W. M. P. van der Aalst, Declare: Full support for loosely-structured processes, in: *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference, EDOC '07*, 2007, pp. 287–298. doi:10.1109/EDOC.2007.4384001.
89. T. T. Hildebrandt, R. R. Mukkamala, Declarative event-based workflow as distributed dynamic condition response graphs, in: *Proceedings Third Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software, PLACES 2010*, Paphos, Cyprus, 21st March 2010., 2010, pp. 59–73. doi:10.4204/EPTCS.69.5.
90. M. Pesic, W. M. P. van der Aalst, A declarative approach for flexible business processes management, in: *Business Process Management Workshops*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 169–180. doi:10.1007/11837862_18.
91. D. Cohn, R. Hull, Business artifacts: A data-centric approach to modeling business operations and processes, *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 32 (3) (2009) 3–9.
92. K. Bhattacharya, R. Hull, J. Su, et al., A data-centric design methodology for business processes, *Handbook of Research on Business Process Modeling* (2009) 503–531 doi:10.4018/978-1-60566-288-6.ch023.

93. B. Bagheri Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, M. Montali, Verification of relational data-centric dynamic systems with external services, in: Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS '13, ACM, New York, NY, USA, 2013, pp. 163–174. [doi:10.1145/2463664.2465221](https://doi.org/10.1145/2463664.2465221).
94. Alfresco Software, Inc., Activiti BPM Software Home, available at: <https://www.activiti.org>.
95. Bizagi, Bizagi – The Digital Business Platform, available at: <http://www.bizagi.com/en>.
96. A. Meyer, M. Weske, Activity-centric and artifact-centric process model roundtrip, in: N. Lohmann, M. Song, P. Wohed (Eds.), Business Process Management Workshops, Springer International Publishing, Cham, 2014, pp. 167–181. [doi:10.1007/978-3-319-06257-0_14](https://doi.org/10.1007/978-3-319-06257-0_14).
97. B. List, B. Korherr, An evaluation of conceptual business process modelling languages, in: Proceedings of the 2006 ACM Symposium on Applied Computing (SAC '06), ACM, New York, NY, USA, 2006, pp. 1532–1539. [doi:10.1145/1141277.1141633](https://doi.org/10.1145/1141277.1141633).
98. ATS Committee on Proficiency Standards for Clinical Pulmonary Function Laboratories, ATS statement: Guidelines for the Six-Minute walk test, American Journal of Respiratory and Critical Care Medicine 166 (1) (2002) 111–117. [doi:10.1164/ajrccm.166.1.at1102](https://doi.org/10.1164/ajrccm.166.1.at1102).
99. B. R. Celli, et al., Standards for the diagnosis and treatment of patients with COPD: a summary of the ATS/ERS position paper, European Respiratory Journal 23 (6) (2004) 932–946. [doi:10.1183/09031936.04.00014304](https://doi.org/10.1183/09031936.04.00014304).
100. N. Lohmann, E. Verbeek, R. Dijkman, Petri Net Transformations for Business Processes — A Survey, in: K. Jensen, W. M. van der Aalst (Eds.), Transactions on Petri Nets and Other Models of Concurrency II, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 46–63. [doi:10.1007/978-3-642-00899-3_3](https://doi.org/10.1007/978-3-642-00899-3_3).
101. F. Kossak, et al., A Rigorous Semantics for BPMN 2.0 Process Diagrams, Springer, 2014. [doi:10.1007/978-3-319-09931-6](https://doi.org/10.1007/978-3-319-09931-6).
102. J. Desel, J. Esparza, Free choice Petri nets, Vol. 40, Cambridge university press, 2005.
103. T. Murata, Petri nets: Properties, analysis and applications, Proceedings of the IEEE 77 (4) (1989) 541–580. [doi:10.1109/5.24143](https://doi.org/10.1109/5.24143).
104. W. M. van der Aalst, K. M. van Hee, Workflow management: models, methods, and systems, MIT press, 2004.
105. J. Dehnert, A. Zimmermann, On the suitability of correctness criteria for business process models, in: International Conference on Business Process Management (BPM), Springer, 2005, pp. 386–391. [doi:10.1007/11538394_28](https://doi.org/10.1007/11538394_28).
106. J. Mendling, H. A. Reijers, W. M. van der Aalst, Seven process modeling guidelines (7pmg), Information and Software Technology 52 (2) (2010) 127–136. [doi:10.1016/j.infsof.2009.08.004](https://doi.org/10.1016/j.infsof.2009.08.004).
107. M. Dumas, L. García-Bañuelos, A. Polyvyanyy, Unraveling unstructured process models, in: Business Process Modeling Notation, Springer, 2010, pp. 1–7. [doi:10.1007/978-3-642-16298-5_1](https://doi.org/10.1007/978-3-642-16298-5_1).

108. J. Vanhatalo, H. Völzer, F. Leymann, Faster and more focused control-flow analysis for business process models through sese decomposition, in: *Service-Oriented Computing (ICSOC)*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 43–55. doi:[10.1007/978-3-540-74974-5_4](https://doi.org/10.1007/978-3-540-74974-5_4).
109. T. Debevoise, J. Taylor, J. Sinur, R. Geneva, *The MicroGuide to Process and Decision Modeling in BPMN/DMN: Building More Effective Processes by Integrating Process Modeling with Decision Modeling*, CreateSpace Independent Publishing Platform, 2014.
110. E. Bazhenova, M. Weske, A data-centric approach for business process improvement based on decision theory, in: *Proceedings of the 15th International Conference on Business Process Modeling, Development and Support*, Vol. 175, Springer, 2014, pp. 242–256. doi:[10.1007/978-3-662-43745-2_17](https://doi.org/10.1007/978-3-662-43745-2_17).
111. R. Vaculin, R. Hull, T. Heath, C. Cochran, A. Nigam, P. Sukaviriya, Declarative business artifact centric modeling of decision and knowledge intensive business processes, in: *Enterprise Distributed Object Computing Conference (EDOC)*, 2011 15th IEEE International, IEEE, 2011, pp. 151–160. doi:[10.1109/EDOC.2011.36](https://doi.org/10.1109/EDOC.2011.36).
112. C. Combi, E. Keravnou-Papailiou, Y. Shahar, *Temporal information systems in medicine*, Springer Science & Business Media, 2010.
113. M. Shitkova, V. Taratukhin, J. Becker, Towards a methodology and a tool for modeling clinical pathways, *Procedia Computer Science* 63 (2015) 205–212. doi:[10.1016/j.procs.2015.08.335](https://doi.org/10.1016/j.procs.2015.08.335).
114. M. Peleg, Computer-interpretable clinical guidelines: a methodological review, *Journal of biomedical informatics* 46 (4) (2013) 744–763. doi:[10.1016/j.jbi.2013.06.009](https://doi.org/10.1016/j.jbi.2013.06.009).
115. G. Schrijvers, A. van Hoorn, N. Huiskes, The care pathway: Concepts and theories: An introduction, *International Journal of Integrated Care* 12 (SPECIAL EDITION I). doi:<http://doi.org/10.5334/ijic.812>.
116. S. Panzarasa, S. Quaglini, A. Cavallini, S. Marcheselli, M. Stefanelli, G. Micieli, Computerised guidelines implementation: Obtaining feedback for revision of guidelines, clinical data model and data flow, in: *11th Conference on Artificial Intelligence in Medicine, (AIME 2011)*, Springer Berlin Heidelberg, Amsterdam, 2007, pp. 461–466. doi:[10.1007/978-3-540-73599-1_62](https://doi.org/10.1007/978-3-540-73599-1_62).
117. R. Müller, A. Rogge-Solti, BPMN for healthcare processes, in: D. Eichhorn, A. Koschmider, H. Zhang (Eds.), *Proceedings of the 3rd Central-European Workshop on Services and their Composition (ZEUS 2011)*, Vol. 705 of *CEUR Workshop Proceedings*, CEUR-WS.org, Karlsruhe, Germany, 2011, pp. 65–72.
118. M. Stefanelli, Knowledge and process management in health care organizations, *Methods of Information in Medicine* 43 (5) (2004) 525–535. doi:[10.1055/s-0038-1633911](https://doi.org/10.1055/s-0038-1633911).
119. H. Pichler, J. Eder, M. Ciglic, Modelling processes with time-dependent control structures, in: *International Conference on Conceptual Modeling (ER)*, LNCS, Springer, Cham, 2017, pp. 50–58. doi:[10.1007/978-3-319-69904-2_4](https://doi.org/10.1007/978-3-319-69904-2_4).
120. P. Merlin, D. Farber, Recoverability of communication protocols—implications of a theoretical study, *IEEE transactions on Communications* 24 (9) (1976) 1036–1043. doi:[10.1109/TCOM.1976.1093424](https://doi.org/10.1109/TCOM.1976.1093424).

121. D. Lime, O. H. Roux, C. Seidner, L.-M. Traonouez, Romeo: A parametric model-checker for Petri nets with stopwatches, in: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, Berlin, Heidelberg, 2009, pp. 54–57. doi:10.1007/978-3-642-00768-2_6.
122. B. Berthomieu, F. Vernadat, Time Petri nets analysis with TINA, in: *Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems, QEST '06*, IEEE Computer Society, Washington, DC, USA, 2006, pp. 123–124. doi:10.1109/QEST.2006.56.
123. G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, M. Hendriks, Uppaal 4.0 doi:10.1109/QEST.2006.59.
124. J. Eder, E. Panagos, M. Rabinovich, Workflow time management revisited, in: *Seminal Contributions to Information Systems Engineering*, Springer, Berlin, Heidelberg, 2013, pp. 207–213. doi:10.1007/978-3-642-36926-1_16.
125. C. Combi, M. Gozzi, J. Juarez, B. Oliboni, G. Pozzi, Conceptual modeling of temporal clinical workflows, in: *IEEE 14th International Symposium on Temporal Representation and Reasoning (TIME)*, IEEE, 2007, pp. 70–81. doi:10.1109/TIME.2007.45.
126. M. Makni, S. Tata, M. Yeddes, N. Ben Hadj-Alouane, Satisfaction and coherence of deadline constraints in inter-organizational workflows, in: *On the Move to Meaningful Internet Systems: OTM 2010*, Springer Berlin Heidelberg, 2010, pp. 523–539. doi:10.1007/978-3-642-16934-2_39.
127. A. Lanz, M. Reichert, Dealing with changes of time-aware processes, in: S. Sadiq, P. Soffer, H. Völzer (Eds.), *Business Process Management*, Springer International Publishing, Cham, 2014, pp. 217–233. doi:10.1007/978-3-319-10172-9_14.
128. J. T. Daugirdas, Dialysis time, survival, and dose-targeting bias, *Kidney international* 83 (1) (2013) 9. doi:10.1038/ki.2012.365.
129. M. z. Muehlen, J. Recker, How much language is enough? theoretical and practical use of the business process modeling notation, in: *Seminal Contributions to Information Systems Engineering*, Springer Berlin Heidelberg, 2013, pp. 429–443. doi:10.1007/978-3-642-36926-1_35.
130. C. Combi, R. Posenato, Towards temporal controllabilities for workflow schemata, in: *17th International Symposium on Temporal Representation and Reasoning (TIME)*, 2010, pp. 129–136. doi:10.1109/TIME.2010.17.
131. J. F. Allen, Maintaining knowledge about temporal intervals, in: *Readings in qualitative reasoning about physical systems*, Elsevier, 1990, pp. 361–372. doi:10.1145/182.358434.
132. R. Posenato, A. Lanz, C. Combi, M. Reichert, Managing time-awareness in modularized processes, in: *Software & System Modeling*, 2018, pp. 1–20. doi:10.1007/s10270-017-0643-4.
133. R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, *Artificial Intelligence* 49 (1–3) (1991) 61–95. doi:10.1016/0004-3702(91)90006-6.
134. L. Hunsberger, R. Posenato, C. Combi, A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks, in: *22nd International Symposium on Temporal Representation and Reasoning (TIME)*, IEEE Press, 2015, pp. 4–18. doi:10.1109/TIME.2015.26.

135. R. R. Gorter, H. H. Eker, M. A. Gorter-Stam, G. S. Abis, A. Acharya, M. Ankersmit, S. A. Antoniou, S. Arolfo, B. Babic, L. Boni, et al., Diagnosis and management of acute appendicitis. EAES consensus development conference 2015, *Surgical endoscopy* 30 (11) (2016) 4668–4690. doi:10.1007/s00464-016-5245-7.
136. C. S. Jensen, J. Clifford, S. K. Gadia, A. Segev, R. T. Snodgrass, A glossary of temporal database concepts, *ACM Sigmod Record* 21 (3) (1992) 35–43. doi:10.1145/140979.140996.
137. B. F. van Dongen, J. Mendling, W. M. van der Aalst, Structural patterns for soundness of business process models, in: 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06), IEEE, 2006, pp. 116–128. doi:10.1109/EDOC.2006.56.
138. L. Toscani, H. D. Aya, D. Antonakaki, D. Bastoni, X. Watson, N. Arulkumaran, A. Rhodes, M. Cecconi, What is the impact of the fluid challenge technique on diagnosis of fluid responsiveness? a systematic review and meta-analysis, *Critical Care* 21 (1) (2017) 207.
139. J.-L. Vincent, M. H. Weil, Fluid challenge revisited, *Critical care medicine* 34 (5) (2006) 1333–1337. doi:10.1097/01.CCM.0000214677.76535.A5.
140. R. Johnson, D. Pearson, K. Pingali, The program structure tree: Computing control regions in linear time, in: *ACM SigPlan Notices*, ACM, New York, NY, USA, 1994, pp. 171–185. doi:10.1145/773473.178258.
141. M. Ciglic, Time management in workflows with loops, in: *On the Move to Meaningful Internet Systems: OTM 2015 Workshops*, Vol. 9416 of LNCS, Springer International Publishing, Cham, 2015, pp. 5–9. doi:10.1007/978-3-319-26138-6_2.
142. L. Rangel-Castillo, S. Gopinath, C. S. Robertson, Management of intracranial hypertension, *Neurologic clinics* 26 (2) (2008) 521–541. doi:10.1016/j.ncl.2008.02.003.
143. A. P. G., *Staphylococcus aureus* (2019).
144. S. Bassil, S. Rinderle, R. Keller, P. Kropf, M. Reichert, Preserving the context of interrupted business process activities, in: *Enterprise Information Systems VII*, Springer Netherlands, Dordrecht, 2006, pp. 149–156. doi:10.1007/978-1-4020-5347-4_17.
145. O. M. Group, *BPMN 2.0 by example, version 1.0 non-normative* (2010).
146. L. Pufahl, M. Weske, Extensible BPMN process simulator, in: *Proceedings of the BPM Demo Track and BPM Dissertation Award co-located with 15th International Conference on Business Process Modeling BPM*, 2017.
147. F. Zerbato, B. Oliboni, C. Combi, M. Campos, J. M. Juarez, BPMN-based representation and comparison of clinical pathways for catheter-related bloodstream infections, in: *IEEE International Conference on Healthcare Informatics (ICHI 2015)*, IEEE, 2015, pp. 346–355. doi:10.1109/ICHI.2015.49.
148. L. A. Mermel, et al., Guidelines for the management of intravascular catheter-related infections, *Clinical infectious diseases* 49 (1) (2009) 1–45. doi:10.1086/599376.
149. B. Cánovas-Segura, M. Campos, A. Morales, J. M. Juarez, F. Palacios, Development of a clinical decision support system for antibiotic management in a hospital environment, *Progress in Artificial Intelligence* 5 (3) (2016) 181–197. doi:10.1007/s13748-016-0089-x.

150. C. Combi, M. Gambini, S. Migliorini, R. Posenato, Representing business processes through a temporal data-centric workflow modeling language: An application to the management of clinical pathways, *IEEE Transactions on Systems Man Cybernetics: Systems* 44 (9) (2014) 1182–1203. doi:10.1109/TSMC.2014.2300055.
151. J. Eder, E. Panagos, M. Rabinovich, Workflow Time Management Revisited, in: *Seminal Contributions to Information Systems Engineering: 25 Years CAiSE*, Springer, 2013, pp. 207–213. doi:10.1007/978-3-642-36926-1_16.
152. A. Lanz, M. Reichert, B. Weber, Process time patterns: A formal foundation, *Information Systems* 57 (2016) 38–68. doi:10.1016/j.is.2015.10.002.
153. F. Durán, G. Salaün, Verifying Timed BPMN Processes Using Maude, in: *Coord. Models Lang.*, Springer, 2017, pp. 219–236. doi:10.1007/978-3-319-59746-1_12.
154. O. Bruyère, et al., An algorithm recommendation for the management of knee osteoarthritis in europe and internationally, *Seminars in Arthritis and Rheumatism* 44 (3) (2014) 253–263. doi:10.1016/j.semarthrit.2014.05.014.
155. E. Kon, et al., Platelet-rich plasma intra-articular injection versus hyaluronic acid viscosupplementation as treatments for cartilage pathology, *Arthrosc. Relat. Surg.* 27 (11) (2011) 1490–1501. doi:10.1016/j.arthro.2011.05.011.
156. K. Batoulis, M. Weske, Soundness of decision-aware business processes, in: *Business Process Management Forum (BPM Forum)*, Vol. 297 of LNBIP, Springer, 2017, pp. 106–124. doi:10.1007/978-3-319-65015-9_7.
157. L. Hunsberger, R. Posenato, C. Combi, The Dynamic Controllability of Conditional STNs with Uncertainty, in: *Workshop Plan. Plan Exec. Real-World Syst.: Princ. Pract. (PlanEx)*, 2012, pp. 1–8.
158. L. Hunsberger, R. Posenato, C. Combi, A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks, in: *22nd International Symposium on Temporal Representation and Reasoning (TIME'15)*, 2015, pp. 4–18. doi:10.1109/TIME.2015.26.
159. C. Combi, L. Hunsberger, R. Posenato, An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty - revisited, in: *5th International Conference on Agents and Artificial Intelligence (ICAART 2013). Revised Selected Papers*, Vol. 449 of CCIS, 2014, pp. 314–331. doi:10.1007/978-3-662-44440-5_19.
160. M. Cairo, L. Hunsberger, R. Posenato, R. Rizzi, A Streamlined Model of Conditional Simple Temporal Networks, in: *24th International Symposium on Temporal Representation and Reasoning (TIME'17)*, Vol. 90 of LIPIcs, 2017, pp. 10:1–10:19. doi:10.4230/LIPIcs.TIME.2017.10.
161. A. Cimatti, L. Hunsberger, A. Micheli, R. Posenato, M. Roveri, Dynamic controllability via timed game automata, *Acta Informatica* 53 (6) (2016) 681–722. doi:10.1007/s00236-016-0257-2.
162. C. Combi, M. Gambini, S. Migliorini, R. Posenato, Representing business processes through a temporal data-centric workflow modeling language: An application to the management of clinical pathways, *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 44 (9) (2014) 1182–1203. doi:10.1109/TSMC.2014.2300055.

163. H. Zhuge, T. yat Cheung, H. keng Pung, A timed workflow process model, *Journal of Systems and Software* 55 (3) (2001) 231 – 243. doi:10.1016/S0164-1212(00)00073-X.
164. E. De Angelis, F. Fioravanti, M. C. Meo, A. Pettorossi, M. Proietti, Verifying controllability of time-aware business processes, in: *International Joint Conference on Rules and Reasoning*, Springer, 2017, pp. 103–118. doi:10.1007/978-3-319-61252-2_8.
165. D. E. Saidouni, N. Belala, R. Boukharrou, A. C. Chaouche, A. Seraoui, A. Chachoua, Time Petri nets with action duration: a true concurrency real-time model, *International Journal of Embedded and Real-Time Communication Systems* 4 (2) (2013) 62–83. doi:10.4018/jertcs.2013040104.
166. C. Flores, M. Sepúlveda, Temporal specification of business processes through project planning tools, in: *Business Process Management Workshops: BPM 2010 International Workshops and Education Track*, Springer, Berlin, Heidelberg, 2011, pp. 85–96. doi:10.1007/978-3-642-20511-8_8.
167. J.-P. Friedenstab, C. Janiesch, M. Matzner, O. Muller, Extending BPMN for business activity monitoring, in: *2012 45th Hawaii International Conference on System Sciences*, IEEE, 2012, pp. 4158–4167. doi:10.1109/HICSS.2012.276.
168. Object Management Group, *Business Process Model and Notation (BPMN)*, v1.2, available at: <http://www.omg.org/spec/BPMN/1.2/PDF> (2009).
169. F. Kossak, C. Illibauer, V. Geist, Event-based gateways: Open questions and inconsistencies, in: *Business Process Model and Notation: Proceedings of the 4th International Workshop*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 53–67. doi:10.1007/978-3-642-33155-8_5.
170. S. Morimoto, A survey of formal verification for business process modeling, in: *Computational Science (ICCS 2008)*, Springer, Berlin, Heidelberg, 2008, pp. 514–522. doi:10.1007/978-3-540-69387-1_58.
171. F. Cassez, O. H. Roux, From time petri nets to timed automata, Tech. rep. (2008).
172. J. Srba, Comparing the expressiveness of timed automata and timed extensions of petri nets, in: *International Conference on Formal Modeling and Analysis of Timed Systems*, Springer, 2008, pp. 15–32. doi:10.1007/978-3-540-85778-5_3.
173. O. Maler, A. Pnueli, J. Sifakis, On the synthesis of discrete controllers for timed systems, in: *Annual Symposium on Theoretical Aspects of Computer Science*, Springer, 1995, pp. 229–242. doi:10.1007/3-540-59042-0_76.
174. G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, D. Lime, Uppaal-tiga: Time for playing games!, in: *International Conference on Computer Aided Verification*, Springer, 2007, pp. 121–125. doi:10.1007/978-3-540-73368-3_14.
175. M. Jurdziński, A. Trivedi, Reachability-time games on timed automata, in: *Automata, Languages and Programming*, Springer, 2007, pp. 838–849. doi:10.1007/978-3-540-73420-8_72.
176. C. Combi, R. Posenato, L. Viganò, M. Zavatteri, Conditional simple temporal networks with uncertainty and resources, *Journal of Artificial Intelligence Research* 64. doi:10.1613/jair.1.11453.

177. R. De Masellis, C. D. Francescomarino, C. Ghidini, M. Montali, S. Tessaris, Add data into business process verification: Bridging the gap between theory and practice, in: AAAI, AAAI Press, 2017, pp. 1091–1099.
178. G. Bruno, A dataflow-oriented modeling approach to business processes, *International Journal of Human Capital and Information Technology Professionals (IJHCITP)* 8 (1) (2017) 51–65. doi:10.4018/IJHCITP.2017010104.
179. W. M. van der Aalst, Business process management: a personal view, *Business Process Management Journal* 10 (2). doi:10.1108/bpmj.2004.15710baa.001.
180. R. Elmasri, S. B. Navathe, *Fundamentals of database systems*, Pearson, 2015.
181. E. F. Codd, A relational model of data for large shared data banks, *Commun. ACM* 13 (6) (1970) 377–387. doi:10.1145/362384.362685.
182. A. Meyer, M. Weske, Extracting data objects and their states from process models, in: 17th International Enterprise Distributed Object Computing Conference (EDOC), IEEE, 2013, pp. 27–36. doi:10.1109/EDOC.2013.13.
183. V. Künzle, M. Reichert, PHILharmonicFlows: towards a framework for object-aware process management, *Journal of Software Maintenance and Evolution: Research and Practice* 23 (4) (2011) 205–244. doi:10.1002/smr.524.
184. N. Vettoretto, et al., Consensus conference on laparoscopic appendectomy: development of guidelines, *Colorectal Disease* 13 (7) (2011) 748–754. doi:10.1111/j.1463-1318.2011.02557.x.
185. N. Trčka, W. M. Van der Aalst, N. Sidorova, Data-flow anti-patterns: Discovering data-flow errors in workflows, in: *International Conference on Advanced Information Systems Engineering (CAiSE)*, Springer, 2009, pp. 425–439. doi:10.1007/978-3-642-02144-2_34.
186. Y. Borgianni, G. Cascini, F. Rotini, Business process reengineering driven by customer value: a support for undertaking decisions under uncertainty conditions, *Computers in Industry* 68 (2015) 132 – 147. doi:https://doi.org/10.1016/j.compind.2015.01.001.
187. S. Abiteboul, R. Hull, V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.
188. S. Abiteboul, C. Beeri, The power of languages for the manipulation of complex values, *VLDB J.* 4 (4) (1995) 727–794. doi:10.1007/BF01354881.
189. G. Özsoyoğlu, Z. Özsoyoğlu, V. Matos, Extending relational algebra and relational calculus with set-valued attributes and aggregate functions, *ACM Transactions on Database Systems (TODS)* 12 (4) (1987) 566–592. doi:10.1145/32204.32219.
190. B. Momjian, *PostgreSQL: introduction and concepts*, Vol. 192, Addison-Wesley New York, 2001.
191. A. Polyvyanyy, S. Smirnov, M. Weske, *Business Process Model Abstraction*, Springer, Berlin, Heidelberg, 2010, Ch. 7, pp. 149–166. doi:10.1007/978-3-642-00416-2_7.
192. V. Diekert, G. Rozenberg, *The book of traces*, World scientific, 1995.
193. R. van Glabbeek, U. Goltz, Refinement of actions and equivalence notions for concurrent systems, *Acta Informatica* 37 (4) (2001) 229–327. doi:10.1007/s002360000041.

194. C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell, A. Wesslén, *Experimentation in Software Engineering*, Springer-Verlag Berlin Heidelberg, 2012.
195. A. Maes, G. Poels, Evaluating quality of conceptual models based on user perceptions, in: *International Conference on Conceptual Modeling (ER)*, Springer Berlin Heidelberg, 2006, pp. 54–67. doi:10.1007/11901181_6.
196. W. Wang, M. Indulska, S. W. Sadiq, B. Weber, Effect of linked rules on business process model understanding, in: *International Conference on Business Process Management (BPM)*, Vol. 10445 of LNCS, Springer, 2017, pp. 200–215. doi:10.1007/978-3-319-65000-5_12.
197. K. Mehmood, S. S.-S. Cherfi, Evaluating the functionality of conceptual models, in: *International Conference on Conceptual Modeling (ER)*, Vol. 10445 of LNCS, Springer, 2009, pp. 222–231. doi:10.1007/978-3-642-04947-7_27.
198. D. S. Cruzes, A. Vennesland, M. K. Natvig, Empirical evaluation of the quality of conceptual models based on user perceptions: A case study in the transport domain, in: *International Conference on Conceptual Modeling (ER)*, Springer Berlin Heidelberg, 2013, pp. 414–428. doi:10.1007/978-3-642-41924-9_34.
199. H. Motulsky, *Intuitive biostatistics: a nonmathematical guide to statistical thinking*, Oxford University Press, USA, 2014.
200. T. K. Kim, T test as a parametric statistic, *Korean journal of anesthesiology* 68 (6) (2015) 540.
201. S. S. Shapiro, M. B. Wilk, An analysis of variance test for normality (complete samples), *Biometrika* 52 (3/4) (1965) 591–611. doi:10.2307/2333709.
202. F. Smith, M. Proietti, Reasoning on data-aware business processes with constraint logic, in: *4th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA)*, 2014, pp. 60–75.
203. Object Management Group, *Object Constraint Language (OCL)*, v2.4, available at: <http://www.omg.org/spec/OCL/> (2014).
204. C. Combi, B. Oliboni, A. Gabrieli, Conceptual modeling of clinical pathways: Making data and processes connected, in: *Artificial Intelligence in Medicine - 15th Conference on Artificial Intelligence in Medicine (AIME 2015)*, Springer International Publishing, Pavia, Italy, 2015, pp. 57–62. doi:10.1007/978-3-319-19551-3_7.
205. S. X. Sun, J. L. Zhao, J. F. Nunamaker, O. R. L. Sheng, Formulating the data-flow perspective for business process management, *Information Systems Research* 17 (4) (2006) 374–391. doi:10.1287/isre.1060.0105.
206. O. Savkovic, E. Marengo, W. Nutt, Query stability in monotonic data-aware business processes, in: *19th International Conference on Database Theory (ICDT)*, Vol. 48 of LIPIcs, 2016, pp. 16:1–16:18. doi:10.4230/LIPIcs.ICDT.2016.16.
207. H. A. Reijers, I. Vanderfeesten, M. G. A. Plomp, P. Van Gorp, D. Fahland, W. L. M. van der Crommert, H. D. D. Garcia, Evaluating data-centric process approaches: Does the human factor factor in?, *Software & Systems Modeling* 16 (3) (2017) 649–662. doi:10.1007/s10270-015-0491-z.
208. C. M. Chiao, V. Künzle, M. Reichert, Integrated modeling of process-and data-centric software systems with philharmonicflows, in: *2013 IEEE 1st International*

- Workshop on Communicating Business Process and Software Models Quality, Understandability, and Maintainability (CPSM), IEEE, 2013, pp. 1–10. doi:[10.1109/CPSM.2013.6703085](https://doi.org/10.1109/CPSM.2013.6703085).
209. A. Marrella, M. Mecella, A. Russo, S. Steinau, K. Andrews, M. Reichert, A survey on handling data in business process models (discussion paper), in: 23rd Italian Symposium on Advanced Database Systems, SEBD, Curran Associates, Inc., 2015, pp. 304–311.
210. P. Wohed, M. Dumas, A. H. M. T. Hofstede, N. Russell, Pattern-based analysis of uml activity diagrams, Tech. rep., Beta, Research School for Operations Management and Logistics (2004).
211. J. M. Küster, K. Ryndina, H. Gall, Generation of business process models for object life cycle compliance, in: G. Alonso, P. Dadam, M. Rosemann (Eds.), Business Process Management: Proceedings of the 5th International Conference, (BPM 2007), Springer Berlin Heidelberg, 2007, pp. 165–181. doi:[10.1007/978-3-540-75183-0_13](https://doi.org/10.1007/978-3-540-75183-0_13).
212. S. Sadiq, M. Orlowska, W. Sadiq, C. Foulger, Data flow and validation in workflow modelling, in: Proceedings of the 15th Australasian Database Conference, Australian Computer Society, Inc., 2004, pp. 207–214.
213. S. X. Sun, J. L. Zhao, J. F. Nunamaker, O. R. L. Sheng, Formulating the data-flow perspective for business process management, Information Systems Research 17 (4) (2006) 374–391. doi:[10.1287/isre.1060.0105](https://doi.org/10.1287/isre.1060.0105).
214. A. Awad, G. Decker, N. Lohmann, Diagnosing and repairing data anomalies in process models, in: International Conference on Business Process Management (BPM), Springer, 2009, pp. 5–16. doi:[10.1007/978-3-642-12186-9_2](https://doi.org/10.1007/978-3-642-12186-9_2).
215. A. Pnueli, The temporal logic of programs, in: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), IEEE, 1977, pp. 46–57. doi:[10.1109/SFCS.1977.32](https://doi.org/10.1109/SFCS.1977.32).
216. S. Von Stackelberg, S. Putze, J. Mülle, K. Böhm, Detecting data-flow errors in BPMN 2.0, Open Journal of Information Systems (OJIS) 1 (2) (2014) 1–19.
217. F. Hasić, J. De Smedt, J. Vanthienen, Developing a modelling and mining framework for integrated processes and decisions, in: On the Move to Meaningful Internet Systems. OTM 2017 Workshops, Springer International Publishing, Cham, 2018, pp. 259–269. doi:[10.1007/978-3-319-73805-5_28](https://doi.org/10.1007/978-3-319-73805-5_28).
218. J. Taylor, J. Purchase, Real-World Decision Modeling with DMN, Meghan-Kiffer Press, 2016.
219. L. Janssens, J. De Smedt, J. Vanthienen, Modeling and enacting enterprise decisions, in: International Conference on Advanced Information Systems Engineering (CAiSE), Springer, 2016, pp. 169–180. doi:[10.1007/978-3-319-39564-7_17](https://doi.org/10.1007/978-3-319-39564-7_17).
220. H. van der Aa, H. Leopold, K. Batoulis, M. Weske, H. A. Reijers, Integrated process and decision modeling for data-driven processes, in: Business Process Management Workshops, Vol. 256 of LNBIP, Springer International Publishing, 2016, pp. 405–417. doi:[10.1007/978-3-319-42887-1_33](https://doi.org/10.1007/978-3-319-42887-1_33).
221. D. L. Parnas, On the criteria to be used in decomposing systems into modules, Communications of the ACM 15 (12) (1972) 1053–1058. doi:[10.1145/361598.361623](https://doi.org/10.1145/361598.361623).

222. E. Bazhenova, S. Buelow, M. Weske, Discovering decision models from event logs, in: International Conference on Business Information Systems (BIS), Vol. 255 of LNBIP, Springer, 2016, pp. 237–251. doi:10.1007/978-3-319-39426-8_19.
223. M. De Leoni, M. Dumas, L. García-Bañuelos, Discovering branching conditions from business process execution logs, in: Fundamental Approaches to Software Engineering (FASE), Vol. 7793 of LNCS, Springer Berlin Heidelberg, 2013, pp. 114–129. doi:10.1007/978-3-642-37057-1_9.
224. B. Von Halle, L. Goldberg, The Decision Model, 1st Edition, Auerbach Publications, 2010.
225. A. Rozinat, W. van der Aalst, Decision mining in ProM, in: Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings, Vol. 4102 of LNCS, 2006, pp. 420–425. doi:10.1007/11841760_33.
226. L. Janssens, E. Bazhenova, J. De Smedt, J. Vanthienen, M. Denecker, Consistent integration of decision (DMN) and process (BPMN) models, in: Proceedings of the CAiSE '16 Forum, Vol. 1603 of CEUR Workshop Proceedings, CEUR-WS.org, 2016, pp. 121–128.
227. A. Zarghami, B. Sapkota, M. Z. Eslami, M. van Sinderen, Decision as a service: Separating decision-making from application process logic, in: 2012 IEEE 16th International Enterprise Distributed Object Computing Conference, 2012, pp. 103–112. doi:10.1109/EDOC.2012.21.
228. F. Boumahdi, R. Chalal, A. Guendouz, K. Gasmia, Soa⁺: a new way to design the decision in soa—based on the new standard decision model and notation (dmn), Service Oriented Computing and Applications 10 (1) (2016) 35–53. doi:10.1007/s11761-014-0162-x.
229. K. Kirchner, N. Herzberg, A. Rogge-Solti, M. Weske, Embedding conformance checking in a process intelligence system in hospital environments, in: Process Support and Knowledge Representation in Health Care, Vol. 7738 of LNCS, Springer Berlin Heidelberg, 2013, pp. 126–139. doi:10.1007/978-3-642-36438-9_9.
230. M. Chinosi, A. Trombetta, BPMN: An introduction to the standard, Computer Standards and Interfaces 34 (1) (2012) 124–134.
231. W. Wang, M. Indulska, S. Sadiq, B. Weber, Effect of linked rules on business process model understanding, in: Proceedings of the Business Process Management: 15th International Conference (BPM 2017), Vol. 10445 of LNCS, Springer International Publishing, 2017, pp. 200–215. doi:10.1007/978-3-319-65000-5_12.
232. D. Calvanese, M. Dumas, F. M. Maggi, M. Montali, Semantic DMN: formalizing decision models with domain knowledge, in: Proceedings of the International Joint Conference on Rules and Reasoning (RuleML+RR), 2017, pp. 70–86. doi:10.1007/978-3-319-61252-2_6.
233. E. Bazhenova, M. Weske, Deriving decision models from process models by enhanced decision mining, in: Business Process Management Workshops, Vol. 256 of LNBIP, Springer, 2015, pp. 444–457. doi:10.1007/978-3-319-42887-1_36.
234. W. Wang, M. Indulska, S. Sadiq, To integrate or not to integrate – the business rules question, in: Proceedings of the 28th International Conference on Advanced Information Systems Engineering (CAiSE), Vol. 9694 of LNCS, Springer

- International Publishing, Cham, Switzerland, 2016, pp. 51–66. doi:10.1007/978-3-319-39696-5_4.
235. L. Pufahl, S. Mandal, K. Batoulis, M. Weske, Re-evaluation of decisions based on events, in: Enterprise, Business-Process and Information Systems Modeling, Springer International Publishing, Cham, 2017, pp. 68–84. doi:10.1007/978-3-319-59466-8_5.
236. H. Leopold, J. Mendling, A. Polyvyanyy, Generating natural language texts from business process models, in: International Conference on Advanced Information Systems Engineering (CAiSE), Springer, 2012, pp. 64–79. doi:10.1007/978-3-642-31095-9_5.
237. W. M. van der Aalst, A. ter Hofstede, B. Kiepuszewski, A. Barros, Workflow patterns, Distributed and Parallel Databases 14 (1) (2003) 5–51. doi:10.1023/A:1022883727209.
238. J. F. Fiore, A. Bialocerkowski, L. Browning, I. G. Faragher, L. Denehy, Criteria to determine readiness for hospital discharge following colorectal surgery: an international consensus using the delphi technique, Diseases of the Colon & Rectum 55 (4) (2012) 416–423. doi:10.1097/DCR.0b013e318244a8f2.
239. A. H. Morris, Developing and implementing computerized protocols for standardization of clinical decisions, Annals of internal medicine 132 (5) (2000) 373–383. doi:10.7326/0003-4819-132-5-200003070-00007.
240. M. Dumas, M. La Rosa, J. Mendling, H. A. Reijers, et al., Fundamentals of business process management, Vol. 1, Springer, 2013. doi:10.1007/978-3-662-56509-4.
241. F. Corradini, A. Ferrari, F. Fornari, S. Gnesi, A. Polini, B. Re, G. O. Spagnolo, A guidelines framework for understandable BPMN models, Data & Knowledge Engineering 113 (2018) 129 – 154. doi:10.1016/j.datak.2017.11.003.
242. H. Scheuerlein, F. Rauchfuss, Y. Dittmar, R. Molle, T. Lehmann, N. Pienkos, U. Settmacher, New methods for clinical pathways—business process modeling notation (BPMN) and tangible business process modeling (t.BPM), Langenbeck’s Archives of Surgery 397 (5) (2012) 755–761. doi:10.1007/s00423-012-0914-z.
243. M. G. Rojo, E. Rolón, L. Calahorra, F. Ó. García, R. P. Sánchez, F. Ruiz, N. Ballester, M. Armenteros, T. Rodríguez, R. M. Espartero, Implementation of the business process modelling notation (BPMN) in the modelling of anatomic pathology processes, Diagnostic pathology 3 (1) (2008) S22. doi:10.1186/1746-1596-3-S1-S22.
244. M. Bertolini, M. Bevilacqua, F. E. Ciarapica, G. Giacchetta, Business process re-engineering in healthcare management: a case study, Business Process Management Journal 17 (1) (2011) 42–66. doi:10.1108/14637151111105571.
245. T. Dingsøyr, S. P. Nerur, V. Balijepally, N. B. Moe, A decade of agile methodologies: Towards explaining agile software development, Journal of Systems and Software 85 (6) (2012) 1213–1221. doi:10.1016/j.jss.2012.02.033.
246. W. J. Kettinger, J. T. Teng, S. Guha, Business process change: a study of methodologies, techniques, and tools, MIS quarterly 21 (1) (1997) 55–80. doi:10.2307/249742.

247. S. Khodambashi, Business process re-engineering application in healthcare in a relation to health information systems, *Procedia Technology* 9 (2013) 949 – 957. doi:<http://dx.doi.org/10.1016/j.protcy.2013.12.106>.
248. A. M. Rotar, M. J. van den Berg, D. S. Kringos, N. S. Klazinga, Reporting and use of the oecd health care quality indicators at national and regional level in 15 countries, *International Journal for Quality in Health Care* 28 (3) (2016) 398–404. doi:[10.1093/intqhc/mzw027](https://doi.org/10.1093/intqhc/mzw027).
249. N. Russell, A. H. Ter Hofstede, D. Edmond, W. M. Van der Aalst, Workflow data patterns: Identification, representation and tool support, in: *International Conference on Conceptual Modeling*, Springer, 2005, pp. 353–368. doi:[10.1007/11568322_23](https://doi.org/10.1007/11568322_23).
250. R. N. Shiffman, R. A. Greenes, Improving clinical guidelines with logic and decision-table techniques: application to hepatitis immunization recommendations, *Medical Decision Making* 14 (3) (1994) 245–254. doi:[10.1177/0272989X9401400306](https://doi.org/10.1177/0272989X9401400306).
251. D. Calvanese, M. Dumas, Ü. Laurson, F. M. Maggi, M. Montali, I. Teinemaa, Semantics and analysis of dmn decision tables, in: *International Conference on Business Process Management (BPM)*, Springer International Publishing, Cham, 2016, pp. 217–233. doi:[10.1007/978-3-319-61252-2_6](https://doi.org/10.1007/978-3-319-61252-2_6).
252. M. R. D. Turco, A. Ponti, et al., Quality indicators in breast cancer care, *European Journal of Cancer* 46 (13) (2010) 2344 – 2356. doi:<http://dx.doi.org/10.1016/j.ejca.2010.06.119>.
253. M. Peleg, S. Tu, A. Manindroo, R. B. Altman, Modeling and analyzing biomedical processes using workflow/Petri Net models and tools, in: *MEDINFO 2004 - Proceedings of the 11th World Congress on Medical Informatics*, Vol. 107 of *Studies in Health Technology and Informatics*, IOS Press, San Francisco, California, USA, 2004, pp. 74–78. doi:[10.3233/978-1-60750-949-3-74](https://doi.org/10.3233/978-1-60750-949-3-74).
254. N. Russell, W. van der Aalst, A. ter Hofstede, Workflow exception patterns, in: E. Dubois, K. Pohl (Eds.), *Advanced Information Systems Engineering*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 288–302. doi:[10.1007/11767138_20](https://doi.org/10.1007/11767138_20).
255. B. S. Lerner, S. Christov, A. Wise, L. J. Osterweil, Exception handling patterns for processes, in: *Proceedings of the 4th international workshop on Exception Handling*, ACM, 2008, pp. 55–61. doi:[10.1109/TSE.2010.1](https://doi.org/10.1109/TSE.2010.1).
256. M. Kunze, M. Weske, *Behavioural Models: From Modelling Finite Automata to Analysing Business Processes*, Springer International Publishing, 2016. doi:[10.1007/978-3-319-44960-9](https://doi.org/10.1007/978-3-319-44960-9).
257. A. Lanz, B. Weber, M. Reichert, Time patterns for process-aware information systems, *Requirements Engineering* 19 (2) (2014) 113–141. doi:[10.1007/s00766-012-0162-3](https://doi.org/10.1007/s00766-012-0162-3).
258. Global Initiative for Chronic Obstructive Lung Disease (GOLD), *Global strategy for diagnosis, management and prevention of COPD* (updated 2016).
259. R. A. Pauwels, A. S. Buist, P. M. Calverley, C. R. Jenkins, S. S. Hurd, Global strategy for the diagnosis, management, and prevention of chronic obstructive pulmonary disease, *American journal of respiratory and critical care medicine* 163 (2001) 1256–1276. doi:[10.1164/ajrccm.163.5.2101039](https://doi.org/10.1164/ajrccm.163.5.2101039).

260. M. L. M. Ortiz, J. Morera, COPD: Differential diagnosis, in: *Pulmonary Disease - Current Concepts and Practice*, INTECH Open Access Publisher, 2012, pp. 105–116.
261. A. Cavallès, G. Brinchault-Rabin, A. Dixmier, F. Goupil, C. Gut-Gobert, S. Marchand-Adam, J.-C. Meurice, H. Morel, C. Person-Tacnet, C. Leroyer, et al., Comorbidities of COPD, *European Respiratory Review* 22 (130) (2013) 454–475. doi:10.1183/09059180.00008612.
262. D. B. Price, B. P. Yawn, R. C. Jones, Improving the differential diagnosis of chronic obstructive pulmonary disease in primary care, *Mayo Clinic Proceedings* 85 (12) (2010) 1122–1129. doi:10.4065/mcp.2010.0389.
263. J. A. Hardie, A. S. Buist, W. M. Vollmer, I. Ellingsen, P. S. Bakke, O. Mørkve, Risk of over-diagnosis of COPD in asymptomatic elderly never-smokers, *European Respiratory Journal* 20 (5) (2002) 1117–1122. doi:10.1183/09031936.02.00023202.
264. B. Kiepuszewski, A. H. M. ter Hofstede, C. J. Bussler, On structured workflow modelling, in: *International Conference on Advanced Information Systems Engineering (CAiSE)*, Springer, 2000, pp. 431–445. doi:10.1007/3-540-45140-4_29.
265. M. L. Brantly, L. D. Paul, B. H. Miller, R. T. Falk, M. Wu, R. G. Crystal, Clinical features and history of the destructive lung disease associated with alpha-1-antitrypsin deficiency of adults with pulmonary symptoms, *American review of respiratory disease* 138 (2) (1988) 327–336. doi:10.1164/ajrccm/138.2.327.
266. M. Miravittles, I. Andreu, Y. Romero, S. Sitjar, A. Altés, E. Anton, Difficulties in differential diagnosis of COPD and asthma in primary care, *British Journal of General Practice* 62 (595) (2012) e68–e75. doi:10.3399/bjgp12X625111.
267. Centers for Disease Control and Prevention and others, *International classification of diseases, ninth revision, clinical modification (ICD-9-CM)* (2013).
268. K.-O. Fagerstrom, N. G. Schneider, Measuring nicotine dependence: a review of the fagerstrom tolerance questionnaire, *Journal of behavioral medicine* 12 (2) (1989) 159–182. doi:10.1007/BF00846549.
269. R. Rodriguez-Roisin, Toward a consensus definition for COPD exacerbations, *Chest* 117 (5) (2000) 398S–401S. doi:10.1378/chest.117.5_suppl_2.398S.
270. A. Casas, T. Troosters, J. Garcia-Aymerich, J. Roca, C. Hernández, A. Alonso, F. del Pozo, P. de Toledo, J. M. Antó, R. Rodríguez-Roisín, et al., Integrated care prevents hospitalisations for exacerbations in COPD patients, *European Respiratory Journal* 28 (1) (2006) 123–130. doi:10.1183/09031936.06.00063205.
271. J. Garcia-Aymerich, C. Hernandez, A. Alonso, A. Casas, R. Rodriguez-Roisin, J. M. Anto, J. Roca, Effects of an integrated care intervention on risk factors of COPD readmission, *Respiratory Medicine* 101 (7) (2007) 1462 – 1469. doi:10.1016/j.rmed.2007.01.012.
272. M. Guerrero, E. Crisafulli, A. Liapikou, A. Huerta, A. Gabarrús, A. Chetta, N. Soler, A. Torres, Readmission for acute exacerbation within 30 days of discharge is associated with a subsequent progressive increase in mortality risk in COPD patients: a long-term observational study, *PLoS one* 11 (3) (2016) e0150737. doi:10.1371/journal.pone.0150737.
273. S. L. Norris, R. E. Glasgow, M. M. Engelgau, P. J. Os'Connor, D. McCulloch, Chronic disease management, *Disease Management & Health Outcomes* 11 (8) (2003) 477–488. doi:10.2165/00115677-200311080-00001.

274. R. Blackburn, D. Stokes, Breaking down the barriers: using focus groups to research small and medium-sized enterprises, *International Small Business Journal* 19 (1) (2000) 44–67. doi:10.1177/0266242600191003.
275. Camunda services GmbH, Camunda DMN Simulator, available at: <https://camunda.org/dmn/simulator/>.
276. S. F. Wamba, D. Mishra, Big data integration with business processes: a literature review, *Business Process Management Journal* 23 (3) (2017) 477–492. doi:10.1108/BPMJ-02-2017-0047.
277. J. De Smedt, F. Hasić, S. K. L. M. vanden Broucke, J. Vanthienen, Towards a holistic discovery of decisions in process-aware information systems, in: *Proceedings of the 15th International Conference on Business Process Management (BPM 2017)*, Vol. 10445 of LNCS, Springer International Publishing, 2017, pp. 183–199. doi:10.1007/978-3-319-65000-5_11.
278. F. Mannhardt, M. De Leoni, H. A. Reijers, W. M. Van der Aalst, Decision mining revisited - discovering overlapping rules, in: *Proceedings of the 28th International Conference on Advanced Information Systems Engineering (CAiSE)*, Vol. 9694 of LNCS, Springer, 2016, pp. 377–392. doi:10.1007/978-3-319-39696-5_23.
279. P. Dadam, M. Reichert, The ADEPT project: a decade of research and development for robust and flexible process support, *Computer Science-Research and Development* 23 (2) (2009) 81–97. doi:10.1007/s00450-009-0068-6.
280. N. Herzberg, K. Kirchner, M. Weske, Modeling and monitoring variability in hospital treatments: A scenario using CMMN, in: *Business Process Management Workshops: International Workshops (BPM 2014)*, Springer International Publishing, Eindhoven, The Netherlands, 2014, pp. 3–15. doi:10.1007/978-3-319-15895-2_1.
281. R. Braun, H. Schlieter, M. Burwitz, W. Esswein, BPMN4CP: Design and implementation of a BPMN extension for clinical pathways, in: *Bioinformatics and Biomedicine (BIBM)*, 2014 IEEE International Conference on, IEEE, 2014, pp. 9–16. doi:10.1109/BIBM.2014.6999261.
282. S. Ferrante, S. Bonacina, G. Pozzi, F. Pinciroli, S. Marceglia, A design methodology for medical processes, *Applied clinical informatics* 7 (1) (2016) 191–210. doi:10.4338/ACI-2015-08-RA-0111.
283. D. Calvanese, M. Dumas, F. M. Maggi, M. Montali, Semantic dmn: Formalizing decision models with domain knowledge, in: *Rules and Reasoning*, Springer International Publishing, 2017, pp. 70–86. doi:10.1007/978-3-319-61252-2_6.
284. M. Bliemel, K. Hassanein, E-health: applying business process reengineering principles to healthcare in Canada, *International Journal of Electronic Business* 2 (6) (2004) 625–643. doi:10.1504/IJEB.2004.006129.
285. T. McNulty, Reengineering as knowledge management a case of change in UK healthcare, *Management Learning* 33 (4) (2002) 439–458. doi:10.1177/1350507602334003.
286. P. A. De Clercq, J. A. Blom, H. H. Korsten, A. Hasman, Approaches for creating computer-interpretable guidelines that facilitate decision support, *Artificial intelligence in medicine* 31 (1) (2004) 1–27. doi:10.1016/j.artmed.2004.02.003.

287. P. Ciccarese, E. Caffi, L. Boiocchi, S. Quaglini, M. Stefanelli, A guideline management system, in: M. Fieschi, E. W. Coiera, J. Y. Li (Eds.), *MEDINFO 2004 - Proceedings of the 11th World Congress on Medical Informatics*, Vol. 107 of *Studies in Health Technology and Informatics*, IOS Press, San Francisco, California, USA, 2004, pp. 28–32. doi:10.3233/978-1-60750-949-3-28.
288. M. Peleg, A. A. Boxwala, O. Ogunyemi, Q. Zeng, S. W. Tu, R. Lacson, E. Bernstam, N. Ash, P. Mork, L. Ohno-Machado, et al., GLIF3: the evolution of a guideline representation format, in: *Proceedings of the AMIA Symposium*, American Medical Informatics Association, 2000, p. 645.
289. S. Miksch, Y. Shahar, P. Johnson, Asbru: a task-specific, intention-based, and time-oriented language for representing skeletal plans, in: *Proceedings of the 7th Workshop on Knowledge Engineering: Methods & Languages (KEML-97)*, Open University, Milton Keynes, UK, 1997, pp. 9–19.
290. D. R. Sutton, J. Fox, The syntax and semantics of the PROforma guideline modeling language, *Journal of the American Medical Informatics Association (JAMIA)* 10 (5) (2003) 433–443. doi:10.1197/jamia.M1264.
291. S. W. Tu, J. R. Campbell, J. Glasgow, M. A. Nyman, R. McClure, J. McClay, C. Parker, K. M. Hrabak, D. Berg, T. Weida, et al., The SAGE guideline model: achievements and overview, *Journal of the American Medical Informatics Association* 14 (5) (2007) 589–598. doi:10.1197/jamia.M2399.
292. E. Shalom, Y. Shahar, E. Lunenfeld, An architecture for a continuous, user-driven, and data-driven application of clinical guidelines and its evaluation, *Journal of Biomedical Informatics* 59 (2016) 130 – 148. doi:10.1016/j.jbi.2015.11.006.
293. M. Peleg, Y. Shahar, S. Quaglini, Making healthcare more accessible, better, faster, and cheaper: the mobiguide project, *European Journal of ePractice: Issue on Mobile eHealth* 20 (2014) 5–20.
294. R. H. Dolin, L. Alschuler, S. Boyer, C. Beebe, F. M. Behlen, P. V. Biron, et al., HL7 clinical document architecture, release 2, *Journal of the American Medical Informatics Association* 13 (1) (2006) 30–39. doi:10.1197/jamia.M1888.
295. W. H. Organization, P. M. I. Corporation, *ICD-9-CM: International Classification of Diseases, 9th Revision: Clinical Modification*, Vol. 1, PMIC (Practice Management Information Corporation), 1998.
296. K. Donnelly, SNOMED-CT: The advanced terminology and coding system for eHealth, *Studies in health technology and informatics* 121 (2006) 279.
297. W. M. van Der Aalst, A. H. Ter Hofstede, B. Kiepuszewski, A. P. Barros, Workflow patterns, *Distributed and parallel databases* 14 (1) (2003) 5–51. doi:10.1023/A:1022883727209.
298. J. Beyer, P. Kuhn, M. Hewelt, S. Mandal, M. Weske, Unicorn meets Chimera: Integrating External Events into Case Management, in: *Proceedings of the BPM Demo Track 2016 Co-located with the 14th International Conference on Business Process Management (BPM)*, 2016, pp. 67–72.