IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 February 2026, accepted 21 March 2026, date of publication 25 March 2026, date of current version 30 March 2026.

*Digital Object Identifier 10.1109/ACCESS.2026.3677629*

RESEARCH ARTICLE

# Modeling Incomplete Procedural Contracts With Blockchain-Based Enforceable Business Processes

**SARA MIGLIORINI, MAURO GAMBINI, VERONICA PATERNOLLI, AND MILA DALLA PREDA**

Department of Computer Science, University of Verona, 37134 Verona, Italy

Corresponding author: Sara Migliorini (sara.migliorini@univr.it)

This work was supported in part by the Interconnected Nord-Est Innovation Ecosystem (iNEST) and the "Innovative Ph.D." initiative, funded by European Union Next-GenerationEU through PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR)–Missione 4 Componente 2, Investimento 1.5–D.D. 1058 23/06/2022 ECS00000043, and MISSIONE 4, COMPONENTE 1, under Grant CUP B31I23000830004.

**ABSTRACT** In the socio-economic landscape, we can recognize an emerging form of organization: the network coalition. A network coalition is a form of concerted cooperation in which a group of agents decides to collaborate to achieve a common goal. The decentralized and cooperative nature of a network coalition presents new challenges in automating its processes, which cannot be treated as traditional business processes managed by a centralized information system. In the literature, the notion of Exogenous Business Process (XBP) has been introduced to capture such processes. An XBP specification is intended as a potentially incomplete, renegotiable, procedural contract to which different parties adhere in order to achieve a predefined business goal. The concept of Enforceable Business Process (EBP) has been proposed as a possible abstraction for modeling and automating the XBPs of a network coalition. An EBP is essentially an evolution of the original concept of smart contract, which is able to properly manage contractual incompleteness while providing automatic enforceability. The aim of this paper is to formalize the notion of EBP and discuss how contractual incompleteness can be handled by a Decentralized Autonomous Information System (DAIS). The complete formalization of both procedural and incompleteness aspects of EBPs, as provided by this paper, lays the basis for the construction of the next generation of DAIS, as well as the diffusion of network coalition into many application domains, such as supply chains, business alliances, joint ventures, and others.

**INDEX TERMS** Blockchain, business process modeling, context modeling, contractual incompleteness, enforceable business process, network coalition, procedural contracts, smart contracts.

## I. INTRODUCTION

The socio-economic landscape is shaped by people who autonomously decide to participate in various emergent or established forms of organizations. We can identify at least three ideal forms of organizations, which we refer to here as *competitive market*, *hierarchical firm*, and *network coalition* [1], [2], [3]. The competitive market is an emergent form of coordination, the hierarchical firm is a designed form of cooperation, and the network coalition is a concerted form of cooperation, where cooperation is intended as the process of coordinating a group of actors with the explicit purpose of achieving a common goal. In this paper, we concentrate on this last form of organization because its decentralized and collaborative nature poses new challenges in the automation of its business processes, which cannot be treated as traditional ones. Supply chains, cooperatives, strategic business alliances, joint ventures, and decentralized cryptocurrencies can be good examples of coalitions. Conversely, consortia can rarely be considered coalitions, because their inter-organizational processes are related to the optimal allocation of shared resources rather than to the achievement of shared objectives through the execution of a set of concerted activities.

The term Exogenous Business Process (XBP) has been introduced in [4] to denote a cooperative, stable process,

The associate editor coordinating the review of this manuscript and approving it for publication was Francisco J. Garcia-Penalvo.

managed by a network coalition of two or more independent organizations that aim to pursue a common goal. The stability of an XBP implies that the process is repetitive in nature, while the mutual independence of the involved parties denotes the lack of a central authority. The authors recognize that such processes can be specified through potentially incomplete, renegotiable procedural contracts that a network coalition establishes to reduce uncertainty and lower transaction costs. A contract is said to be *procedural* if all its completely specified parts can be directly interpreted by a Decentralized Autonomous Information System (DAIS) without additional details. At the same time, contractual *incompleteness* is a pivotal concept of several economic theories, such as Transaction Cost Economics and Contract Theory [5], [6]. It can have manifold origins, for instance, the impossibility of predicting all relevant eventualities, the costs of describing all identified eventualities in advance, the impossibility of observing specific actions of the contracting parties, and the difficulty of making the observable actions verifiable by a trusted third party. During the contracting process, the parties evaluate the costs and benefits of including additional contractual terms. In light of this evaluation, the parties can rationally choose to omit several details, leaving out many unlikely eventualities. Therefore, it is reasonable to assume that any contract, even the most formal one, could be intentionally incomplete. In long-term interactions, such as those established within a network coalition, purposely incomplete contracts can mitigate renegotiation costs, but at the expense of increased dependence on trust.

Mutual trust between the involved parties is an essential requirement for a successful implementation of XBPs. Indeed, the lack of trust between the involved parties traditionally leads to the introduction of a central third-party authority [7]. However, in a network coalition, identifying and maintaining an appropriate central authority can be challenging given the dynamic nature of this type of organization. Its introduction could eventually prevent the organization's success and distort its nature. Blockchain technology could be an alternative approach to addressing this problem and eliminating the need for mutual trust or for introducing a third party. We can say that blockchain technology substitutes the need for mutual trust among the parties with a trust in the technology and the protocol itself. A preliminary investigation into the potential of blockchain technology for business process management (BPM) approaches has been outlined in [7] and [8]. The authors discuss how the challenges of collaborative design and the lack of trust have limited the application of BPM approaches to managing inter-organizational business processes. They also recognize that blockchain technology may help address such problems, particularly those related to the lack of trust. Indeed, the blockchain will likely enable new governance models and foster a more flexible corporate culture.

The applicability of existing blockchain-based smart contract technology to properly implement XBPs has been investigated in [4]. However, it has emerged that the inherent incompleteness of XBPs clashes with the immutability-by-default of the currently available smart contracts. To scale up, network coalitions demand not only enforceable and verifiable contracts, but also easily renegotiable ones that can foster continuous improvement of their contractual terms. For this reason, the work introduced the notion of Enforceable Business Process (EBP) as an evolution of the original concept of smart contracts, capable of automating potentially incomplete, renegotiable, and procedural contracts.

The aim of this paper is to take a step forward in this direction by providing a complete formalization of the concept of EBP and of incompleteness in XBP models. We extend the Business Process Model and Notation (BPMN) by introducing new constructs that explicitly address both enforceability and contractual incompleteness. By allowing the representation of EBP using a widely used standard such as BPMN, we aim to facilitate its use across areas such as supply chains, digital marketplaces, and inter-organizational workflows, where such notation is already in use. Moreover, we aim to bridge the gap between business process modeling and smart contract execution, making automated agreements more adaptable to real-world scenarios and enabling more flexible, adaptive execution of cooperative business processes. Furthermore, we ground our formalization in contract theory, demonstrating how EBPs can systematically capture and respond to any incompleteness scenarios that may arise in XBP specifications.

The paper is structured to promote an incremental understanding of the notion of EBP, starting with its procedural nature and progressing to the concept of incompleteness and its management. This journey involves an exploration of a proper extension of the standard BPMN notation, leading to a discussion about renegotiability and process instance migrations. As a result, the paper provides a foundation for many research directions, including, for instance, the definition of model checking and logical proofs about the completeness and correctness of the process instance migrations towards renegotiated EBPs, the management of renegotiations and migrations in the context of smart contracts, and the implementation of a fully functioning blockchain-based DAIS.

The remainder of the paper is organized as follows: Sect. II provides an overview of related work concerning both the management of inter-organizational business processes and business process flexibility. Sect. III presents a motivating example from the domain of on-demand food ordering and delivery. Sect. IV formalizes the concept of procedural contract with a particular emphasis on the enforceability aspect, while Sect. V presents an extension of BPMN able to capture the provided formalization, and Sect. VI discusses how an EBP can be implemented by using smart contracts. Sect. VII formalizes the various forms of incompleteness that can arise in a procedural contract, while Sect. VIII discusses how such incompleteness can be modeled in an EBP and managed by a DAIS, and Sect. IX discusses the possible

upgradability of smart contracts. Finally, Sect. X summarizes the work and presents some future research directions.

## II. RELATED WORK

**Inter-organizational processes** – Business Process Management has been originally conceived for supporting intra-organizational business processes. However, in recent years, different organizations have established close collaborations to achieve business goals that cannot be (easily) addressed by a single partner. The need to support inter-organizational business processes has led to the definition of new concepts, such as collaborative business processes, web service choreography, and virtual organizations.

A *Collaborative Business Process (CBP)* [9] is a business process involving multiple actors belonging to different firms. It is based on a commonly accepted operational protocol and executed by multiple workflow engines without the presence of a central coordination authority. The parties agree to follow a common interaction protocol represented by the CBP, but at the same time, the related business activities clearly belong to each party [10]. The original CBP approach is vague in some respects, in particular regarding how the individual BP instances interact, how exceptions are managed, who is responsible for updates, and what happens if a firm changes the protocol unilaterally.

In contrast to CBPs that could be considered a top-down approach, a Service-Oriented Architecture (SOA) can provide a market-like, bottom-up-oriented solution through service choreographies. A *Web-Service Choreography (WSC)* [11] is a way to compose in a unique system two or more existing web-services, potentially offered by different service providers; a choreography is usually the result of a service wiring activity that ends in a loosely-coupled message interchange protocol. The term choreography suggests that the involved parties share a common protocol but during the execution, each one can run autonomously in compliance with its declared interface [12].

An XBP can be in principle automated inside a *Virtual Organization (VO)* [13], [14] that can be defined as a group of actors, potentially located over a wide geographical area, which decides to collaborate on a common goal by putting together their competencies and resources. A suitable technological platform has to be chosen for implementing a VO and a SOA solution seems the most widely accepted one; see, for example [15]. A VO comes into existence from an *initiator*, namely an actor that proposes the initial idea, searches for external collaborations, selects suitable partners to fulfill operational needs, and delegates responsibilities. Overall, the initiator is responsible not only for the creation of the VO but also for its maintenance and termination. The VO concept reveals the classical traits of a hierarchical firm, in which the initiator plays the role of a principal. The creation of a VO is somewhat equivalent to firm integration; hence, when XBP parties cannot merge, the applicability of this concept is questionable.

**Blockchain for business process management** – The authors of [16] provide a systematic literature review of the use of blockchain technology to support collaborative business processes; they clearly identify opportunities and gaps that warrant further research. Moreover, the authors classify existing proposals by the business process lifecycle stages supported by blockchain technology. In [17], the authors propose a taxonomy of blockchain-based approaches during the enactment phase. In particular, they investigate which challenges posed by inter-organizational processes can be addressed by the use of blockchain technology. Existing approaches are classified based on three capabilities: model support, resource allocation, and process flexibility. Similarly, relative to the monitoring stage, the work in [18] identifies the benefits and pitfalls of blockchain-enabled business process monitoring. The authors propose adopting programmable blockchain platforms to manage the generation, distribution, and analysis of business-process monitoring data.

In [19], the authors propose leveraging the irreversibility of blockchain-based smart contracts to address the lack of trust in inter-organizational processes. In particular, they propose a system where the blockchain technology is applied in two forms: (1) as a choreography monitor, by storing inside the blockchain the sequence of message exchanges in an immutable and not repudiable way, and (2) as an active mediator, by using contracts to coordinate the process execution. A choreography expressed in BPMN is translated into a set of smart contracts, and the blockchain is used to store the exchanged messages. This initial work evolved in the Caterpillar project [20], in which smart contracts are used to represent an entire process, not only a choreography. More specifically, Caterpillar can translate a process model into a Petri Net and, in turn, to a smart contract that can be executed by the Ethereum Virtual Machine (EVM). In this case, the blockchain stores the state of the process instance and the overall execution steps in an immutable way.

In [21], the authors introduce a set of techniques that take business process models as input and transform them into statecharts for Blockchain and process participants. Conversely, in [22], the authors propose new language constructs for enforcing choreographies using blockchain technology. The operational semantics of such constructs are also provided, and a proof-of-concept implementation is proposed.

Ignoring efficiency concerns, Caterpillar and similar experiments provide the first evidence that the blockchain technology could offer a viable approach for the XBP automation.

**Contractual incompleteness and business process flexibility** – The inherent incompleteness of XBPs can be compared with the concept of *flexibility by underspecification*, i.e., the ability to execute an incomplete process model at run-time by managing anticipated changes for which handling strategies could not be defined at design-time [23]. Flexibility through underspecification relies on late binding

and late modeling strategies to enact underspecified nodes and is deemed useful for processes with a fixed structure but including distinct parts designed and controlled by different, autonomous work groups [23].

The unpredictability of future process eventualities is a possible source of incompleteness in XBPs and is comparable to the need to handle exceptions and unforeseen events occurring during the enactment of pre-specified processes [24], [25]. Whereas the management of anticipated exceptions may be planned in advance, e.g., by embedding the exception logic into the process logic, unanticipated exceptions require ad hoc process instance changes [26]. Unpredictability, in general, is also a key feature of loosely structured and unstructured (knowledge-intensive) processes [27], which must be executed flexibly through collaboration and negotiation among participants.

To support the modeling and execution of loosely defined processes, decisions regarding the exact specification of selected parts of the process are deferred to the process run-time [26]. To deal with the default immutability of blockchain-based smart contracts, the authors in [28] propose CoBuP, a decentralized Collaborative Business Process execution architecture using blockchain, which presents an interpreter of BPMN process models, supporting the instantiating, execution, and monitoring of process instances. In this case, a generic smart contract is deployed once and is responsible for invoking predefined functions that can also be dynamically updated at runtime.

In [29], the authors introduce a model for consensus-based control-flow flexibility, in which participants in a process can collectively agree on how to steer the business process within the boundaries defined by agreed policies. This idea is partially captured by the voting activity introduced in this paper as an extension of a generic BPMN subprocess.

## III. MOTIVATING EXAMPLE

This section introduces an example of a network coalition, along with a characterizing XBP that will be particularly useful for discussing the various paper contributions.

Let us consider the recently proliferating context of on-demand food orders and delivery. On-demand food delivery networks connect customers with restaurants and self-employed couriers: customers can easily order food from a restaurant through an online platform and have it picked up and delivered to their doorstep by couriers. Food delivery companies such as Just Eat Italy[1] and Deliveroo[2], among others, provide centralized online platforms that act as a ''mediator'' between customers, restaurants, and independent couriers, supporting ordering, delivery, and payment. Yet, such companies typically impose high commission rates.

A more cost-effective solution, avoiding the need for centralized companies, can be realized by a network of couriers and restaurants working as independent contractors to guarantee customers a trusted, decentralized food delivery service. Such a network coalition may bring together the interests of couriers and restaurants. In particular, couriers will benefit from greater flexibility and fairer remuneration, while restaurants will have the opportunity to rely on an external delivery service and take advantage of advertising and customer loyalty opportunities. Additionally, couriers and restaurants are free to delegate delivery and food preparation, respectively, to other members of the network coalition.

Contracts are used to coordinate interactions among all parties in the network, defining all ordering and delivery aspects, including payment modalities and time constraints, and enforcing refunds in the event of delivery failure to meet customer requirements. The happy path for food delivery consists of the following steps: the customer selects a restaurant and the food to order on an online platform. Then, the order is placed, and a contract is issued containing information about the ordered food, the amount to be paid, the payment modality, the desired delivery time, and the customer's address. This information is then shared with the selected restaurant, which can accept or refuse the order. If the order is accepted, the restaurant reaches out to nearby couriers willing to deliver the order and then estimates the preparation time. At this time, the customer is provided with order confirmation details, and the courier is notified of the expected pick-up time. Then the courier collects the food and drives to the provided address to ultimate the delivery. Once the customer confirms the delivery, the payment is processed based on the chosen modality: if the customer chooses to pay in cash, the money is handed over to the courier, who must transfer the due amount to the restaurant. Finally, customers are encouraged to rate the delivery service to participate in loyalty programs.

The contract underlying such a network coalition is characterized by several clauses regulating the interaction among ''the customer'' (CU), ''the restaurant'' (R), and ''the courier'' (CO). Among them, we can mention the following ones:

1) *Order acceptance*. Both R and CO are penalized if they accept more orders than those they can fulfill on time.
2) *Order confirmation*. CU must be notified of order confirmation or rejection within 10 minutes of order placement.
3) *Delivery delay*. Delay responsibility is determined by the following clauses based on when the delay accumulates during the delivery process.
   (a) *Delayed pick-up*. R is responsible for delays caused by preparations that are not completed within the estimated pick-up time; CO is responsible for delayed food collection.
   (b) *Delayed delivery*. CO is not responsible for delays caused by events outside his or her reasonable control, such as strikes, civil commotion, natural disasters, and the impossibility of using transport; CUs are responsible for delayed delivery if the provided

[1] https://www.justeat.it
[2] https://deliveroo.it/en/

address is incorrect or inaccurate and they fail to communicate with CO.

Compensation is provided to the aggrieved parties by R or CO in proportion to their responsibility. The compensated amount is calculated based on the price of the food and delivery service.

4) *Wrong delivery*. CU is fully refunded if the delivered food is different from the ordered one. R is responsible for compensation if the delivered package contains unordered items, CO is responsible for compensation if the wrong food package is delivered.
5) *Failed delivery*. If the delivery delay exceeds 60 minutes or is canceled by R at any time after confirmation, CU will be fully refunded by the parties responsible for the delay and will receive a 20% discount on a new order. If CU is not present at the provided address within 10 minutes of expected delivery, 100% of the order will be charged.

This example illustrates a real-world scenario where the need for both enforceability and flexibility in contractual agreements becomes evident. Moreover, the complexity of coordinating independent couriers and restaurants, each with its own constraints and responsibilities, underscores the challenges of managing contract incompleteness in decentralized business collaborations. In the remainder of this article, we will use this example to illustrate our formalization of EBPs and the handling of incompleteness within XBP models.

## IV. FORMALIZATION OF PROCEDURAL CONTRACTS

This section formalizes the notion of EBP in mathematical notation by extending the traditional notion of business process. Given that, Sect. V introduces the graphical notation for representing an EBP by using an extension of BPMN, while Sect. VI discusses how the concept introduced here could be implemented in a blockchain-based environment.

As mentioned in the introduction, this paper addresses the specification of inter-organizational business processes carried out by a network coalition. Since a network coalition is a concerted form of cooperation established by a group of mutually independent agents to achieve a common goal, we need first of all to define the notion of *agent* formally.

*Definition 1 (Agent):* An agent is a subject (person or organization) that can decide to collaborate with other agents to achieve a common goal. An agent is characterized by a set of roles that he or she can play in a collaboration. An agent can be represented as a tuple $a = \langle \sigma, \varrho, \omega \rangle$ where $\sigma$ is a unique identifying address, $\varrho = \{r_1, \ldots, r_n\}$ is the set of roles that $a$ can play, and $\omega : \mathbb{N} \rightarrow \mathbb{N}$ is a function returning for each timestamp $d \in \mathbb{N}$ the amount of tokens held in the $a$'s wallet.

Let us note that the notion of wallet $\omega$ is introduced in the definition primarily to represent the voting power assigned to each agent, as well as their ability to withstand incentives and penalties during the execution of the process. Following the logic of blockchain wallets, each agent can hold only a positive number of tokens. At the same time, each agent is uniquely identified by his or her address $\sigma$ in a blockchain-based environment.

A network coalition is dynamic in nature: agents are free to join and leave the network at any time.

*Definition 2 (Network Coalition):* A network coalition can be defined as a tuple $NC = \langle A, R \rangle$, where $A$ is the set of agents belonging to the network coalition and $R$ is the set of roles involved in the network coalition. Let $\Delta$ be the lifespan of the network coalition, function $\phi : A \times \Delta \rightarrow \{0, 1\}$ is a *participation function* denoting whether a given agent $a \in A$ is available at a given time $d \in \Delta$.

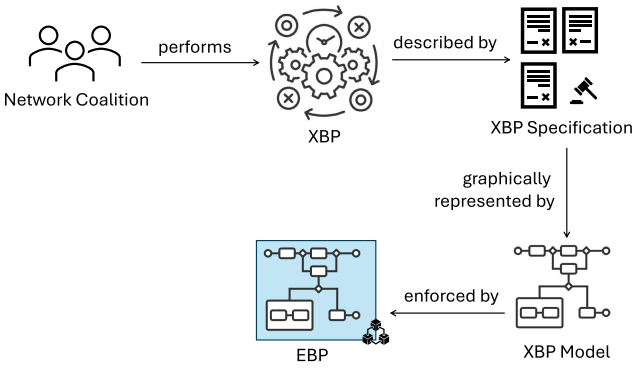


**FIGURE 1. Relationships between the concepts of network coalition, XBP and EBP.**

To guarantee decentralization within a network coalition, a constraint can be imposed on the minimum number of agents required to cover each role. In particular, if having at least one agent for each role is necessary to ensure the executability of the process instances, this number could be increased based on the specific application domain. Formally, we can say that

$$\forall d \in \Delta \; \forall r \in R \,. \\ |\{a \in A \mid \phi(a, d) = 1 \; \wedge \; r \in a.\varrho\}| \geq n \qquad (1)$$

where the notation $a.\varrho$ is used to denote the set of roles $\varrho$ of $a$, and $n$ characterizes the desired decentralization degree of the network coalition.

As reported in Fig. 1, the activities carried on by a network coalition can be organized into a set of inter-organizational business processes called Exogenous Business Processes.

*Definition 3 (XBP):* An *Exogenous Business Process (XBP)* is a cooperative, stable process managed by a coalition of two or more mutually independent agents in order to pursue a common goal.

The XBPs carried on by a network coalition can be formally described by means of XBP specifications. An *XBP specification* is a potentially incomplete, renegotiable, procedural contract [4]. A contract is said to be procedural if a DAIS can directly interpret all its completely specified parts without additional details. The properties of incompleteness and renegotiability will be discussed in Sect. VII, while this section concentrates on its procedural and enforceable nature.

According to the classical notion of business process modeling, an *XBP model* can be defined as an XBP specification built using graphical constructs with a clearly stated semantics. As discussed in the introduction, to address the lack of trust characterizing a network coalition, this semantics is provided with reference to the notion of blockchain-based smart contracts to avoid introducing a central authority while maintaining enforceability.

An *Enforceable Business Process (EBP)* is an XBP model archetype representing a renegotiable smart contract. The following definition provides a formalization of the notion of EBP by extending the one of business process.

*Definition 4 (EBP):* An *Enforceable Business Process (EBP)* carried out by a network coalition $NC = \langle A, R \rangle$ is represented as a tuple:

$$\mathcal{P} = \langle \mathcal{N}, \mathcal{C}, \mathcal{R}, \alpha, \tau, \gamma_t, \gamma_m, \epsilon, \rho \rangle \tag{2}$$

where $\mathcal{N}$ is a non-empty set of flow nodes, $\mathcal{C}$ is a non-empty set of control-flow edges, and $\mathcal{R} = A$ is a set of resources or actors involved in the network coalition $NC$.

- The set $\mathcal{N} = \mathcal{A} \cup \mathcal{G} \cup \mathcal{E}$ of control flow nodes consists of the disjoint sets $\mathcal{A}$ of activities (i.e., tasks or sub-process), $\mathcal{G}$ of gateways, and $\mathcal{E} = \{s\} \cup \{e\} \cup \mathcal{E}_{int}$ of events, where $s$ is the start event, $e$ is the end event, and $\mathcal{E}_{int}$ is a set of intermediate events.
- The set $\mathcal{C} \subseteq \mathcal{N} \times \mathcal{N}$ of control flow edges defines precedence between the elements in $\mathcal{N}$.
- The function $\alpha : \mathcal{A} \to \{task, subprocess\}$ distinguishes activities into tasks and subprocesses.
- The function $\tau : \mathcal{A} \to \{user, send, receive, script, enforceable, voting\}$ assigns to each activity its type.
- The function $\gamma_m : \mathcal{G} \to \{split, join\}$ assigns a gating mechanism to each gateway.
- The function $\gamma_t : \mathcal{G} \to \{parallel, exclusive, event\text{-}based\}$ assigns a type to the gateways of $G$.
- The function $\epsilon : \mathcal{E} \to \{throwing, catching\}$ distinguishes between events that throw a trigger and those that catch a result. $\epsilon(s) \mapsto catching$ whereas $\epsilon(e) \mapsto throwing$.
- Finally, the function $\rho : \mathcal{A} \to \mathcal{R}^n$ assigns to each activity the resource or resources responsible for its execution.

The main difference distinguishing an EBP from a traditional business process is the presence of the following two additional specific kinds of activities: enforceable activity and voting activity. These new kinds of activities mainly reflect the nature of a blockchain-based smart contract: the presence of activities that can automatically enforce rewards or penalties, with the consequent transfer of a specific amount of tokens, and the possibility to express a vote through the amount of tokens owned in the wallet.

*Definition 5 (Enforceable Activity):* An *enforceable activity* is an automatically enforced sub-process eventually enriched with the specification of some incentives and penalties, together with a deadline for its execution. It can be formalized as a tuple:

$$ET = \langle T, i, p, d \rangle \tag{3}$$

where $T \in \mathcal{A}$ is the activity (i.e., task or sub-process) to be performed, $i \in \mathbb{N}$ the incentive associated with the activity execution, $p \in \mathbb{N}$ is the penalty due if the activity is not executed (both expressed in number of tokens) and $d \in \Delta$ is an execution deadline.

Notice that all parts, except $T$, are optional. For instance, an enforceable activity could be defined without specifying an incentive, a penalty, a deadline, or both, depending on the specific contextual needs and the information at disposal during the EBP design. What remains mandatory is the automatic enforceability of $T$, namely its execution on a blockchain. This aspect can lead to an incomplete specification, as it will be described in the second part of the paper.

Whereas enforceable activities are executed by a single resource, namely $|\rho(ET)| = 1$, voting activities involve the participation of multiple agents; specifically, at least two agents must take part in the execution of this particular kind of activity. In this sense, it is intended as a collaboration activity which could be initiated by one of the actors in the network coalition and then requires the execution of actions also by other participants in order to complete.

*Definition 6 (Voting Activity):* A *voting activity* is a decision activity prompted by a conflict in which two or more agents make individual decisions and interact with each other until an agreement is reached. It can be formalized as a tuple:

$$VT = \langle c, s, d, \ell \rangle \tag{4}$$

where $c$ is the conflict prompting the negotiation, $s$ is the solution agreed upon by the executing agents, $d$ is an eventual deadline imposed for collecting the votes, and $\ell$ is the minimum number of collected votes.

While the conflict $c$ and the set of solutions $s$ are mandatory to instantiate a voting activity, the threshold $\ell$ and the deadline $d$ could be omitted. If $\ell$ is omitted, it is intended that a vote needs to be expressed and collected by all agents currently belonging to the network coalition, while a missed deadline $d$ means that no temporal limit is posed on the voting achievement. The agents participating in a voting activity must have a positive token balance in their wallets to be eligible to vote. Let $\Delta_{va}$ be the lifespan of a voting activity $VT$ and let $\omega(d)$ be the amount of tokens that the agent holds in his or her wallet at a certain time $d$. The set of agents eligible to vote is defined as $A_{va} = \{a_1, \ldots, a_n | \forall d \in \Delta_{va} : a.\omega(d) > 0\}$. It is required that for each voting activity $va$, there exists at least $l$ agents in $A_{va}$, namely $|A_{va}| \geq \ell$.

Clearly, the possibility of omitting the deadline $d$ in both enforceable activity and voting activity can lead to an indefinite wait in some situations. However, if during the definition of an EBP the correct value for $d$ is difficult to establish and the possibility of indefinite wait is considered rare to occur, the network coalition could decide to omit it in

the EBP specification. This will be discussed further in the following sections related to incompleteness.

As highlighted in Def. 4, the set $\mathcal{C}$ of control flow edges defines an execution precedence among the nodes in the set $\mathcal{N}$, making an EBP essentially a labeled graph where there is at least one starting node (i.e., start event $s$) and one end node (i.e., the end event $e$). Inside such EBP graph, one or more alternative execution paths can be identified, based on the presence of exclusive and event-based gateways that can produce several alternative sequences of activities to be executed. Notice that such gateways can also represent cycles, which can produce different execution paths depending on the number of iterations in each specific instance.

*Definition 7 (EBP path):* Given an EBP $\mathcal{P} = \langle \mathcal{N}, \mathcal{C}, \mathcal{R}, \alpha, \tau, \gamma_t, \gamma_m, \epsilon, \rho \rangle$, an execution path $\pi$ of $\mathcal{P}$ denotes a connected maximal sub-graph included between a start event $s \in \mathcal{E}$ and an end event $e \in \mathcal{E}$, and containing all the nodes $n \in \mathcal{N}$ that are reachable starting from $s$ following the control flow edges, such that all split gateways of the kind exclusive and event-based have exactly one branch.

Each EBP path can also be represented as a tuple containing the identifiers of the control-flow nodes, sorted with respect to their execution order and separated by a comma if the order is sequential or by a double vertical bar if the order is parallel. For exclusive and event-based gateways, the considered outcome is reported between brackets:

$$\pi ::= \langle s, X, e \rangle$$
$$X ::= T_{id} \mid G^h_{id}(o) \mid T_{id_1} || \ldots || T_{id_n}$$

where $s$ and $e$ are the identifiers of the start and end event, respectively, $T_{id}$ is the identifier associated with an activity (i.e., task or subprocess), $G^h_{id}$ is the identifier associated with a gateway of type $h \in \{x = exclusive, e = event\text{-}based\}$, and $o$ is the outcome of the evaluation of the gateway condition, or the event happened.

In the following, the set of all possible execution paths for an EBP $\mathcal{P}$ is denoted as $\Pi$. Moreover, given an execution path $\pi \in \Pi$ and a node $n \in \pi$, the notation $\bullet n$ denotes the node immediately before $n$, and $n\bullet$ denotes the node immediately after $n$, following the control flow edges in $\pi$.

An example of EBP path represented with this notation is the following one: $\langle s, T_1, G^x_1(o), T_2 || T_3, e \rangle$, where $s$ and $e$ are the identifiers of the start and end event, respectively, $T_1$, $T_2$, and $T_3$ are identifiers of activities, $G^x_1(o)$ denotes an exclusive gateway with identifier $G^x_1$ whose outcome is represented by the value $o$, while $T_2$ and $T_3$ are intended to be executed in parallel. Notice that only the identifier of the split exclusive and event-based gateways are reported in the representation of the execution path, while the split parallel gateways are denoted through the double vertical bar, and the join gateways are omitted since they are not useful during the analysis of the possible path executions.

The execution of an EBP $\mathcal{P}$ by a DAIS originates a so-called EBP instance $\iota$, whose upper completion can be considered the realization of a specific path $\pi$. At the beginning of the execution (i.e., during the $s$ event), $\iota$ is compatible with all the possible execution paths, while as some events occur or some choices are made, the set of paths compatible with $\iota$ is progressively reduced, until the final end node $e$ is reached and a unique path $\pi$ is compatible with the execution of $\iota$. In the following, we will use the notation $\iota \subseteq \pi$ to denote the fact that the execution of $\iota$ is compatible with the path $\pi$, while the symbol $\Pi_\iota \subseteq \Pi$ is the set of all paths compatible with $\iota$. Finally, the notation $\pi[\![n]\!]$ is used to denote a portion (i.e., subsequence) of $\pi$ starting from the start event $s$ and stopping at the $n$-th element inside $\pi$, while $|\pi|$ is the number of sequential elements in $\pi$, namely the length of $\pi$.

*Definition 8 (EBP instance):* Given an EBP $\mathcal{P} = \langle \mathcal{N}, \mathcal{C}, \mathcal{R}, \alpha, \tau, \gamma_t, \gamma_m, \epsilon, \rho \rangle$, an EBP instance $\iota$ represents an execution of a path $\pi$ of $\mathcal{P}$ which is arrived at a specific element $x_n \in \mathcal{N}$ and is compatible with a subset $\Pi_\iota$ of all possible paths for $\mathcal{P}$, i.e., $\Pi_\iota \subseteq \Pi$. The following conditions are satisfied:

- If $n = |\pi|$, then $x_n = e$, namely the instance $\iota$ has been completed and $|\Pi_\iota| = 1$.
- If $n < |\pi|$, then $(\forall \pi_h \in \Pi_\iota \,.\, |\pi_h| \geq n) \wedge (\forall \pi_h, \pi_j \in \Pi_\iota \,.\, \pi_h[\![n]\!] = \pi_j[\![n]\!])$.

In other words, if the execution of $\iota$ is complete, then there could be only one compatible path $\pi$. Otherwise, if $\iota$ represents a partial execution, then all the compatible paths share the subsequence of activities already executed and the evaluation of the gateways already traversed.

An EBP instance $\iota$ is formalized by a tuple including the elements of $\Pi_\iota$ which have already been started or completed together with a timestamp denoting when such activity has started or the split gateway condition has been evaluated.

Considering again the execution path $\pi = \langle s, T_1, G^x_1(o), T_2 || T_3, e \rangle$, a possible instance $\iota$ of $\pi$ could be:

$$\iota = \langle s[t_1], T_1[t_2], G^x_1(o)[t_3] \rangle \tag{5}$$

denoting the fact that the start event happens at time $t_1$, while the first task $T_1$ has been started at time $t_2$, the condition of the exclusive gateway $G^x_1$ has been evaluated as $o$ at time instant $t_3$. Conversely, none of the subsequent parts of the path have been executed yet.

Given this mathematical notation, the following section introduces the graphical constructs used to represent an EBP.

## V. A GRAPHICAL NOTATION FOR PROCEDURAL CONTRACTS

In this section, we provide a graphical syntax for the novel concepts formalized in Sect. IV starting from the descriptive conformance level of the well-known BPMN [30] standard. Our intent is to adhere to BPMN as much as possible, and thus, we extend the notation only when really necessary to address specific constructs of an EBP.

Fig. 2 illustrates the standard graphical elements taken from the BPMN graphical notation to represent an EBP. In particular, traditional flow nodes representing gateways and events are used, and with respect to the descriptive conformance level of BPMN, we include event-based gateways,

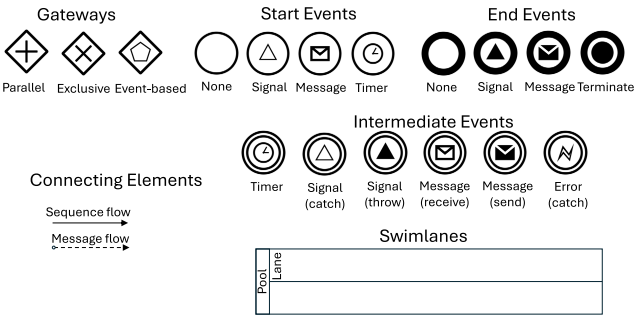


**FIGURE 2.** BPMN elements used to represent an EBP.

signal events, and error events in our graphical alphabet. A *parallel gateway* is used to visualize the concurrent execution of activities, while an *exclusive gateway* identifies alternative paths in the model, and an *event-based gateway* is used to make a decision based on events. Relatively to the events, we can recognize *message events*, which waits until a proper message is received (i.e., receive) or is published (i.e., send), *signal events*, which are used to broadcast or capture a triggered signal, *error signals* used to catch an exception in process execution, and finally *timer events* that are triggered by a deadline or timer.

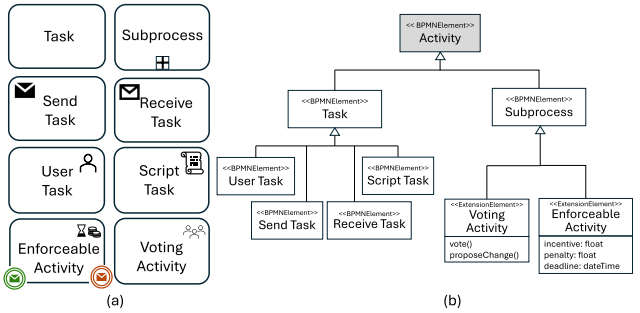


**FIGURE 3.** Taxonomy of BPMN activites.

Regarding the activities composing an EBP, we use the taxonomy provided by the BPMN language and extend it with the two new types of activities discussed in Sect. IV. Fig. 3 illustrates such hierarchy and the provided extensions. In particular, as in BPMN, activities are classified into tasks (i.e., atomic activities) and sub-processes (i.e., compound activities). Several kinds of tasks can be specified which are directly derived from the BPMN notation: *user tasks* representing general activities done by a component of the network coalition, *send tasks* which send a message to another participant inside the network coalition, *receive tasks* which wait for a message to arrive. Moreover, we have *script tasks*, defined in the BPMN notation as tasks executed by a business process engine, which, in our case, is represented by a blockchain infrastructure; namely, they will be implemented as smart contract functions. At the same time, relatively to the sub-process side, we have the two novel compound activity types introduced in the previous section.

*Enforceable activities* are characterized by a deadline (represented by an hourglass) and the set of incentives and penalties (i.e., token icon) obtained in case the prescribed contract clauses are respected or violated. These two possible outcomes are represented in Fig. 3 by the green and red message icons, respectively. Conversely, *voting activities* allow the network coalition to reach an agreement about something. They are represented by a group icon to denote the involvement of multiple agents, while the two possible outcomes (i.e., the approval or the rejection of the voting subject) are given in Fig. 3 by the green and red message icons, respectively.

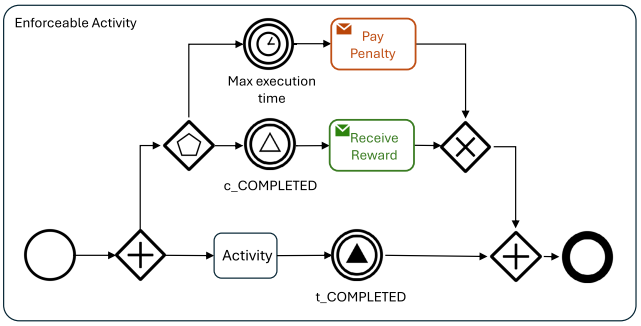


**FIGURE 4.** Implementation of an enforceable activity in BPMN.

Fig. 4 illustrates a possible representation of an enforceable activity in BPMN. The general idea is that if the activity is not completed in time (represented by the timer/deadline (check)), an event is generated by sending an appropriate message that triggers the automatic payment of penalties by the executor and the premature termination of the activity. Conversely, if the activity is completed on time, a reward is automatically generated in favor of the executor before the task is completed. In both cases, an appropriate message is generated to notify the other agents in the network coalition through the corresponding send task. Clearly, more complex situations can be modeled by specifying different kinds of penalties or rewards, depending on the contractual clauses agreed upon by the parties. Moreover, the usage of an event-based gateway for the two branches, leading to a rejection or an approval, allows us to make the deadline optional. Indeed, even if the above branch is never activated, the sub-process can still complete soon or later. As introduced in the previous section, and as will be further discussed in the following one, this could constitute a form of incompleteness in the specification of an enforceable activity. Similarly, depending on the situation, both or only one of the two top branches (i.e., the one containing "pay penalty" and the one containing "receive reward") could be present. Using this construction, we can define a general enforceable activity, where the core activity in Fig. 4 can be replaced with an appropriate instantiation that represents more or less complex smart contract functionality.

A possible implementation of voting activity is depicted in Fig. 5. In this case, a receive task is iteratively executed until

the required number of votes $X$ has been received. A timer event can also be used to require that such votes be received before a given deadline. Similarly to the previous case, the maximum waiting time can be omitted if an admissible deadline can be established in advance. Another parameter to be established in a voting activity is represented by the value of $X$, which represents the minimum number of positive votes required to complete the execution. This can be defined, for instance, as a dynamic parameter passed at run-time that depends on the number of actors currently present, or as a fixed percentage of the number of currently available actors.

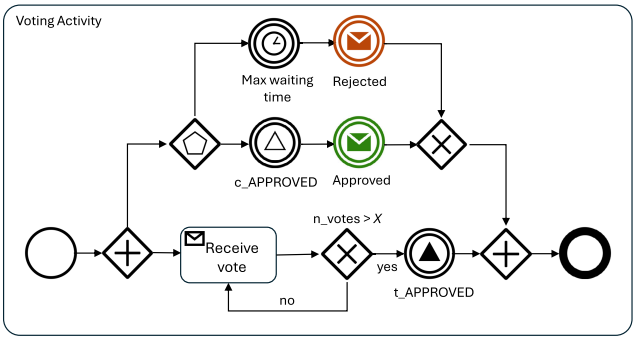


**FIGURE 5.** Implementation of a voting activity in BPMN.

Let us notice a final characteristic about a voting activity. As mentioned in the previous section, it is a collaborative activity that requires actions (i.e., votes) performed by different agents. If each agent is represented as a distinct lane inside a BPMN representation of an EBP, this means that a voting activity will be initiated by an agent inside the corresponding pool, which activate a procedure in a specific collaboration space, represented by a dedicated pool called *coalition*, where participants will send their votes and the final result will be published. In particular, the agreement reached will be notified through a proper approval or rejection message sent to all the involved parties.

*Example 1:* With reference to the example introduced in Sect. III, its graphical representation is given through the business process which has been split in Fig. 6 and Fig. 7 for increasing readability. The dots at the end of the process in each pool denote that the specification continues in the next figure; labels are used to clearly identify the conjunction parts. As you can see, three kinds of agents (i.e., roles) are represented by distinct pools: the courier (CO), the restaurant (R), and the customer (CU), to which an additional lane called ''coalition'' is added to accommodate voting activities, together with a lane for the activities of an ''oracles'' used to retrieve reliable and immutable information about the external world. Notice that the ''courier'' pool is characterized by two lanes. In BPMN, each lane of a pool represents different roles inside the same organization. In this case, they are used to represent the different scenarios played by couriers: the courier who actually performs the delivery, and the other couriers who participate in the voting tasks.

The EBP starts when a customer selects a restaurant and the food to order (T1), then an order is placed through a send message task (T2), and the restaurant initiates its activities. When an order is received, the restaurant checks it and decides to accept or reject it (T3). If the order is declined, an event message is sent to the customer, and the EBP terminates. Conversely, if the order is accepted, the restaurant notifies the available couriers (T4) who can then self-candidate themselves for the delivery (E5-T9). The choice of courier is made via a voting activity (T8) in which both the restaurant and the customer and any other possible couriers participate.

Once an agreement on the courier has been reached, the restaurant estimates the pick-up time (T5). Given that, in Fig. 7, a parallel gateway starts two concurrent parts of the process, one related to delivery and one to food preparation. If, during food preparation (T14), the courier notifies the restaurant that he/she will not be able to reach it in time, the order will be canceled and the process terminated. This is managed via an enforceable activity, so the ''Cancel order'' activity (T13) will include steps to manage the courier penalty. In the same way, the ''Prepare food'' activity (T14) is also an enforceable activity, and if it is not completed correctly, the order will be canceled, and a penalty will be applied to the Restaurant (T24).

When the food is ready, the Courier will pick it up, and the ''Ride to customer'' (T16) enforceable activity will start. In this case, if the enforceable activity (T16) is not completed in time, the order will be canceled, and a penalty will be applied to the Courier through the task ''Cancel Order'' (T15). However, it could be possible that an exceptional event has occurred, preventing the courier from delivering the food on time. In this case, a voting activity ''Handle courier accident'' (T22) will be initiated, through which the coalition can decide to give the courier more time or cancel the order. Conversely, if the food is delivered on time, the delivery could be fulfilled, and the process could terminate successfully.

With reference to the notion of EBP path introduced in Def. 7 and by using the identifiers put in light blue near each task and gateway in the EBP $\mathcal{P}$ in Fig. 6-7, some execution paths in $\Pi$ among the others could be:

$$\pi_1 = \langle s_1, T_1, T_2, E_1, T_3, G_3^x(\text{“no”}), E_2, G_1^e(E_3), e_1$$

$$\pi_2 = \langle s_1, T_1, T_2, E_1, T_3, G_3^x(\text{“yes”}), T_4, \langle\langle E_5, T_9, T_6\rangle || E_{16}\rangle, T_7, \langle\langle E_{17}, E_{18}\rangle || \langle E_4, E_{53}\rangle || \langle E_{11}, E_{12}\rangle\rangle, T_8, E_6, E_7, \langle\langle E_{13}, e_2\rangle || e_3\rangle\rangle$$

Many other possible paths can be identified, based on the evaluation of the event-based gateways $G_1^e$, $G_2^e$, $G_4^e$, $G_5^e$, $G_6^e$, $G_7^e$, $G_8^e$, $G_9^e$,$G_{10}^e$, $G_{11}^e$, $G_{13}^e$ and of the exclusive gateways $G_3^x$, $G_{12}^x$, $G_{14}^x$. In general, all the paths in $\Pi$ share the same initial sub-part $\langle s_1, T_1, T_2, E_1, T_3\rangle$. Therefore, an instance $\iota$, which is currently performing one of the initial three tasks, is compatible with all the execution paths in $\Pi$

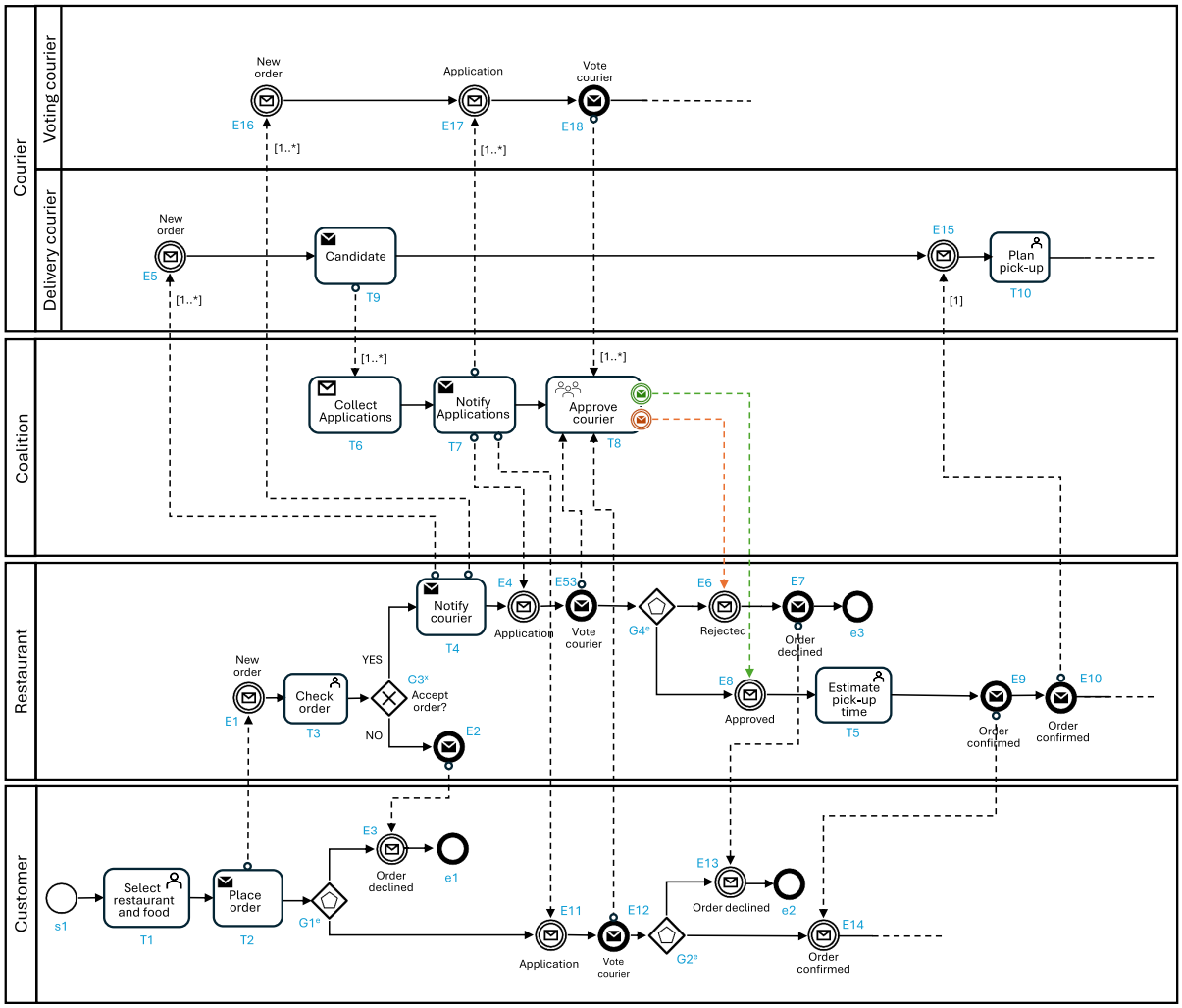


**FIGURE 6.** A graphical representation of the contract described in Sect. III. The dots at the end of each pool mean that the EBP specification continues in Fig. 7, where the same identifiers in light blue are used to ease the identification of the common parts.

(i.e., $\Pi_i = \Pi$). However, as some branches are discarded (e.g., $G_3^x$ is evaluated), some paths become incompatible with $\iota$ and will not be part of $\Pi_\iota$ anymore (i.e., $\Pi_\iota \subset \Pi$). Another aspect to note is that in identifying possible execution paths, the sending of a message typically activates another branch in parallel, similar to the execution of a parallel gateway, but without a corresponding join gateway.

## VI. A SMART CONTRACT IMPLEMENTATION OF PROCEDURAL CONTRACTS

Smart contracts represent a major extension of the blockchain system introduced and popularized by the Ethereum platform [31]. They transformed blockchain from a distributed ledger for payments into a general-purpose computational platform. A smart contract is a self-executing program deployed on the blockchain that automatically enforces predefined rules and agreements once specific conditions are met. Unlike traditional contracts, which rely on intermediaries for validation and enforcement, smart contracts operate in a decentralized and trustless environment where the logic is transparent, verifiable, and tamper-resistant [32]. A smart contract is deployed by submitting its compiled bytecode to the blockchain through a transaction, after which it is assigned a unique contract address and becomes part of the immutable ledger. Once deployed, interaction with the contract occurs by submitting transactions to its address, which include a data payload specifying the invoked function and its input arguments; in addition, read-only calls can be used to query contract data without modifying the ledger. Each state-changing invocation triggers execution by the virtual machine and produces a deterministic state transition that is validated and replicated across the network, making it immutable.

This section discusses a possible implementation of the building blocks of a procedural contract introduced in the previous sections, utilizing blockchain-based smart contracts. Clearly, it does not provide a fully functioning implementation, since its details strictly depend on the specific

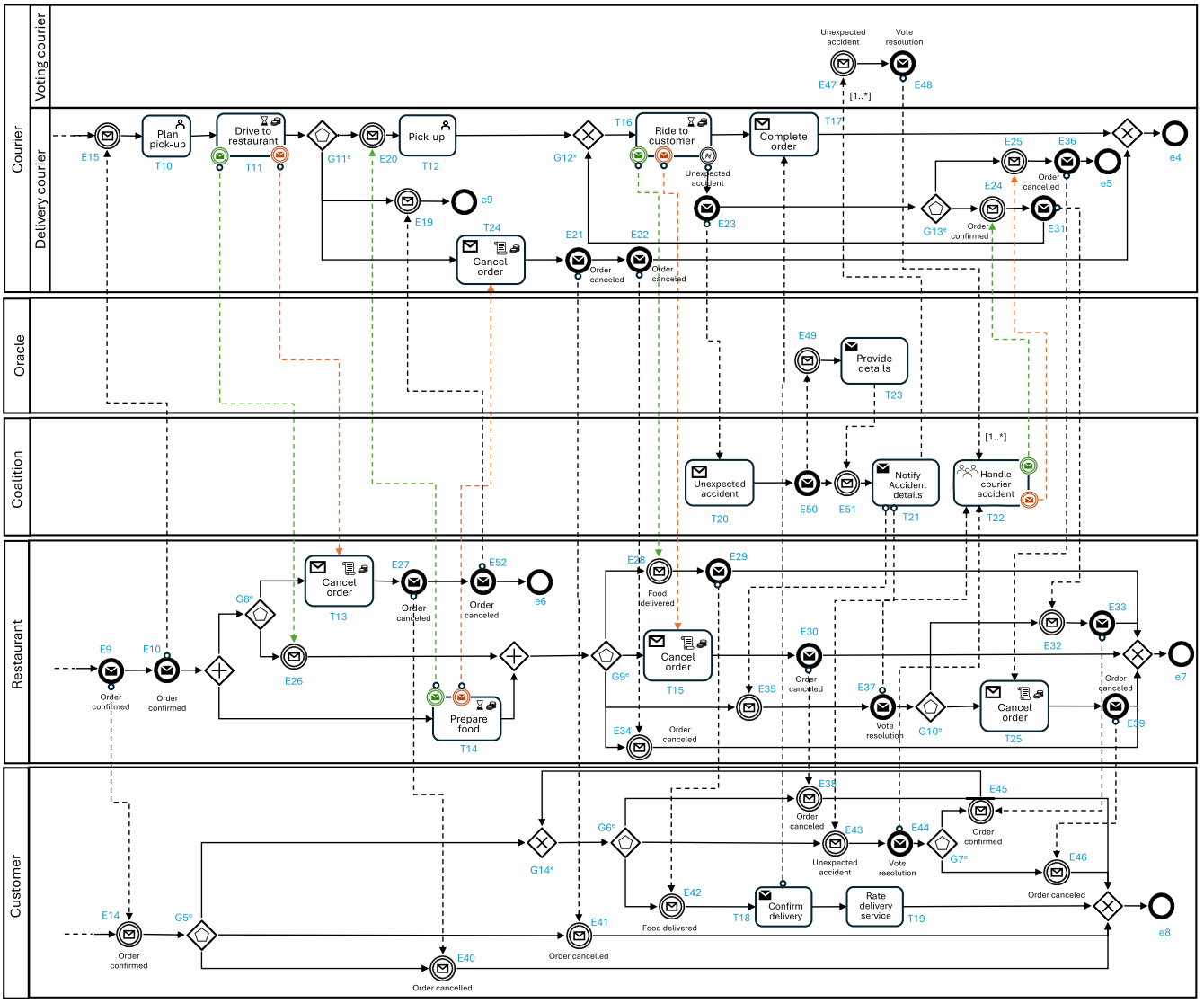


**FIGURE 7.** A graphical representation of the contract described in Sect. III. The dots at the beginning of each pool mean that the specification continues the one in Fig. 6, where the same identifiers in light blue are used to ease the identification of the common parts.

application needs. However, the proposed pseudo-code can be intended as patterns that a real DAIS should follow during a real implementation.

First of all, the notion of network coalition given in Def. 2 could be implemented through a smart contract like the one in Alg. 1, where the multi-role environment is implemented as suggested in [33]. More specifically, the smart contracts take care of a set of roles, whose kind and quantity depend on the specific application needs, and associate them with the agents currently participating in the network coalition. For instance, with reference to the motivating example in Sect. III, we can identify three roles **CU**, **CO**, and **R** (see line 2). The modifier *hasRole*($a$, $r$) takes care of checking if an actor $a$ has the role $r$ and can be used to guard each smart contract functionality requiring a particular role for being executed.

To perform such a check and take a trace of the connection between each agent and its roles, two mappings are maintained: one to connect each agent with the corresponding roles (i.e., *agents*) and one to connect each role with the set of agents covering it (i.e., *roles*). Each agent is represented by a blockchain address and is also connected to a certain number of available tokens via the mapping *wallets*.

These mappings are also useful to check the property stated in Eq. 1 through a modifier that can be applied to each function as a prerequisite for the execution of each enforceable or voting task. It essentially checks, through the function *checkValidity*($n$), that for each available role in the enumeration *Roles* there are at least $n$ different associated addresses in the mapping *roles*.

As explained in the previous sections, the decentralized nature of a network coalition allows agents to freely join

**Algorithm 1** Smart Contract for Implementing a network coalition: Role Management and Validity Property Check

```
1:  contract NetworkCoalition
2:      enum Roles ← {CU, CO, R}
3:      mapping(address → Roles[]) agents
4:      uint numAgents
5:      mapping(address → int) wallets
6:      mapping(Role → address[]) roles
7:      modifier hasRole(a, r)
8:          require(r ∈ agents.get(a))
9:          _
10:     end modifier
11:     modifier isValid(n)
12:         require(checkValidity(n), "Invalid coalition")
13:         _
14:     end modifier
15:     function checkValidity(n)
16:         for r ∈ Roles do
17:             if roles[r].length < n then
18:                 return false
19:             end if
20:         end for
21:         return true
22:     end function
23:     ...
24: end contract
```

**Algorithm 2** Smart Contract for Implementing a network coalition: Functions for Agent's Joining and Leaving

```
1:  contract NetworkCoalition
2:      ...
3:      function join(rs) payable
4:          a ← msg.sender
5:          t ← msg.value
6:          agents.put(a, rs)
7:          numAgents ← numAgents + 1
8:          for r ∈ rs do
9:              roles.get(r).add(a)
10:         end for
11:         wallets.put(a, t)
12:     end function
13:     function leave( )
14:         a ← msg.sender
15:         rs ← agents(a)
16:         agents.remove(a)
17:         numAgents ← numAgents - 1
18:         for r ∈ rs do
19:             roles.get(r).remove(a)
20:         end for
21:         t ← wallets.get(a)
22:         wallets.remove(a)
23:         payable(a).transfer(t)
24:     end function
25:     ...
26: end contract
```

and leave the coalition itself. This means that the smart contracts in Alg. 2 will provide at least two functions: one for adding a new agent and another for the agent to leave the coalition. When an agent joins the network coalition through the function *join*(), he or she needs to deposit some amount of tokens in the wallet, which will be used to determine the voting power, or implement the mechanism of incentives and penalties. Conversely, when an agent $a$ leaves the network coalition, he/she will receive back the deposited tokens. Notice that the transfer of the token follows the established withdrawal pattern [34] for securing the transfer and preventing attacks, such as the reentrancy attack [35].

The two types of activities characterizing an EBP, as illustrated in Sect. IV, can be implemented naturally using a blockchain solution. In particular, relatively to an enforceable activity, since the core activity contained inside it could be different, we can model it through a modifier wrapping the body of a general smart contract function, as in Alg. 3. The modifier initially checks whether the agent has sufficient tokens in their wallet to cover the penalty if the deadline is not met. Then, the specific activity is executed, and depending on whether the deadline $d$ is satisfied, the incentive $i$ is added to, or the penalty is subtracted from, the agent's wallet. Notice that the deadline is expressed in terms of block height, not through a timestamp, in order to avoid possible attacks performed by a miner [34]. An alternative implementation of the deadline mechanism could use an external transaction, for instance, one originating from an Oracle service, that calls a smart contract function that emits a specific event, which is then checked within the enforceable activity. Moreover,

**Algorithm 3** Implementation of an enforceable activity

```
1:  contract NetworkCoaltion
2:      ...
3:      modifier enforceable(i, p, d)
4:          a ← msg.sender
5:          require(p = ⊥ ∨ wallets.get(a) > p)
6:          _
7:          if d ≠ ⊥ ∧ block.number ≤ d then
8:              if i ≠ ⊥ then
9:                  wallets.get(a).add(i)
10:             end if
11:         else
12:             if p ≠ ⊥ then
13:                 wallets.get(a).subtract(p)
14:             end if
15:         end if
16:     end modifier
17:     ...
18: end contract
```

depending on the complexity of the enforceable activity, the deadline check could also be performed multiple times

during the execution for a more reactive termination. The implementation in Alg. 3 also takes care of the situations in which some parameters among $i$, $p$, or $d$ have not been specified.

**Algorithm 4** Implementation of a voting activity

```
1:  contract NetworkCoalition
2:      ...
3:      enum State ← {created, voting, ended}
4:       State state
5:      mapping voters ← ∅
6:      uint numVotes
7:      uint[] options
8:      mapping preferences ← ∅
9:      uint startTime
10:     uint minVotes
11:     uint deadline
12:     modifier checkState(State s)
13:         require(state == s)
14:         _
15:     end modifier
16:     function createVote(opts, d, ℓ)
17:         state ← State.created
18:         options ← opts
19:         deadline ← d
20:         minVotes ← ℓ ∨ numAgents
21:         voters ← ∅
22:         numVotes ← 0
23:         preferences ← ∅
24:     end function
25:     function startVote( ) checkState(created)
26:         state ← State.voting
27:         startTime ← block.number
28:     end function
29:     function vote(o) checkState(voting)
30:         a ← msg.sender
31:         t ← block.number - startTime
32:         require(t = ⊥ ∨ t < deadline, "Expired.")
33:         require(wallets.get(a) > 0, "No right to vote.")
34:         require(voters.get(a) == 0, "Already vote.")
35:         x ← options.get(o)
36:         options.put(o, x + wallets.get(a))
37:         voters.get(msg.sender) ← 1
38:         numVotes ← numVotes + 1
39:         if numVotes ≥ minVotes then
40:             state ← State.ended
41:         end if
42:     end function
43:     function endVote( ) checkState(voting)
44:         state ← State.ended
45:     end function
46:     ...
47: end contract
```

A voting activity could be implemented using a set of smart contract functionalities, as in Alg. 4. In particular, each voting activity is characterized by an initiation, an actual voting phase, and an ending step. Each of these steps is encoded in the enumeration *State* (see line 3) and checked through a specific modifier (see lines 12-15). A new voting activity is started by the function *createVote*$(opts, d, \ell)$, which essentially initializes the current state (see variable *state*), the available options (see variable *options*), two mappings, one registering who already votes (see variable *voters*) and the other collecting the provided preferences (see variable *preferences*), and a set of other parameters like the deadline $d$ and the minimum number of votes $\ell$. In particular, if the minimum number of votes to receive $\ell$ has not been specified, it is set equal to the number of agents in the network coalition (see line 20). Once a voting activity has been created, it can be started; namely, the state can be changed so that the agents can start expressing their preferences through the function *vote*$(o)$. This function checks whether the agent calling it has the right to cast a vote: namely, that his/her balance is greater than zero and that he/she has not already voted. If both conditions are met, the expressed option is registered with a weight proportional to the agent's shares.

Functions used to represent both an enforceable activity and a voting activity could also be guarded by a modifier that checks the ownership of the appropriate role before calling the function. Namely, with reference to the motivating example in Fig. 6-7, there could be an enforceable activity or a voting activity that can be called by the courier, the restaurant, or the customer, depending on which pool it is located.

The pseudo-code presented in this section provides a first hint on how an EBP could be implemented using a blockchain-based smart contract directly executable by a DAIS. The next section begins investigating the incompleteness characteristics of a procedural contract.

## VII. FORMALIZATION OF INCOMPLETENESS IN PROCEDURAL CONTRACTS

Previous sections present the procedural aspects of an XBP. In this section, we will treat the other characterizing aspect of an XBP specification, namely its incompleteness and renegotiability, while in Sect. VIII we discuss how these aspects can be represented by an EBP and in Sect. IX incompleteness is treated in terms of smart contract renegotiation.

Contractual incompleteness is legally tied to the subject matter of the contract, as it arises from the inherent limitations in fully specifying all possible contingencies, obligations, and performance conditions within the agreement. This dogmatic category is merely related to the question of determinability, a requirement established, for example, by Art. 1346 of the Italian Civil Code, which also regulates elements such as possibility, lawfulness, and determinateness. In contrast, in Common Law, contracts are not the subject of a specific and unambiguous but contract theory is based on the concept of a bargain, that is, a negotiation between the parties aimed at generating an agreement, the object of which consists of an exchange of promises (consideration), susceptible to economic evaluation [36]. It follows that, although there is

no specific normative reference, in both Italian Civil Law and Common Law, the subject matter of the contract must comply with certain essential characteristics to validly form and exist.

We can summarize that, from a legal point of view, the concept of contractual incompleteness is characterized by asymmetric information between the parties involved in the execution of contractual clauses [37], [38]. Considering the actors CU, R, and CO, mentioned in the motivating example of Sect. III, the contractual incompleteness is due to not specifying all the elements that characterize the exchange relationship. This approach also reflects the commercial theory of Transactional Construct Theory [39].

Contractual incompleteness can be categorized with respect to different aspects; in particular, we consider its observability and verifiability first, and then its intentionality. With respect to the former, contractual incompleteness can be categorized as overt or latent.

*Definition 9 (Contractual overt incompleteness):* An *overt incompleteness* occurs when a contract is found to be lacking in the specification and obligations under it or when vague expressions have been used in its clauses, i.e., which do not specify in detail the behavior of the actors (i.e. ''reasonable effort'', ''good faith'', ''efficient behavior'').

Since an EBP model is a graphical specification of a procedural contract, this form of incompleteness may be considered rare to occur. Indeed, the aim is to formally define a process directly interpretable by a DAIS without additional details. However, with respect to the definition provided in Sect. V, this form of incompleteness can still arise, for instance, in the presence of manual user tasks assigned to agents inside the network coalition. Conversely, the most common form of EBP incompleteness can be classified as latent.

*Definition 10 (Contractual latent incompleteness):* A *latent incompleteness* refers to the fact that an apparently complete contract may not turn out to be so when the contractual relationship changes significantly or some knowledge not previously available becomes evident. In this case, the contracting partner assumes negative and non-cooperative commercial behavior, depriving the other party of the benefits expected from the contractual transaction, thereby creating the so-called *hold up* [40]. These scenarios highlight that the vulnerable party is unlikely to make optimal specific investments or to improve the quality of personal performance.

Since this form of contractual incompleteness arises in the presence of significant changes to the contractual relationship or the execution environment, it underscores the need for the EBP to define a re-negotiation process that addresses the traditional immutability of a blockchain-based smart contract, as we will discuss further below.

On the other hand, given the intent of incompleteness in the negotiation process, contractual incompleteness can be either unintended or intentional.

*Definition 11 (Contractual unintended incompleteness):* An *unintended incompleteness* happens when the parties do not know in advance all the details or eventualities that need to be included in the contract, so they leave some parts unintentionally underspecified or missed.

This form of incompleteness may not be evident inside an EBP model since it represents the final result of a consultation between the parties, whose intent is not expressed in the specification. Conversely, we are more interested in the last form of incompleteness, the intended one.

*Definition 12 (Contractual intended incompleteness):* An *intentional incompleteness* is merely a strategic choice by the parties to maintain greater flexibility, reduce transaction costs, or exploit future uncertainties. It may be original or subsequent to contracting.

Given these two categorizations of contractual incompleteness, we can identify the following taxonomy:

- **OU** *Overt-unintended*: the contractual relationship is based on a lack of specification or a vague specification, a condition that burdens the weaker party to the contract. The latter, although aware of the information asymmetry that characterized the relationship, was unable to negotiate more definite terms or to fill the contractual gaps.
- **OI** *Overt-intentional*: the information asymmetry in the relationship was consciously determined by both parties involved in the relationship, either at the original stage or at a later time during the execution of the contractual clauses.
- **LU** *Latent-unintended*: modification during the performance of the contractual relationship places the weaker party at a disadvantage, depriving him or her of the benefits expected from entering into the agreement due to the adoption of unagreed behavior by one or more of the parties involved.
- **LI** *Latent-intentional*: the sensitive change in the contractual relationship could be based on a mere strategic choice. This may occur at the original stipulation stage or at a later time during execution.

From a computer science perspective, temporal logic [41] is widely used to describe and verify system behavior over time. In relation to contract theory, it can be applied to describe how obligations, permissions, and clauses evolve over time. In particular, it allows us to reason explicitly about time-dependent conditions, such as the one stated in Fig. 6: if the courier does not reach the restaurant in time, a penalty will be applied to them. The use of temporal logic eliminates the ambiguity inherent in natural language when defining such conditions. In particular, it helps to formally specify time-dependent clauses, making assumptions explicit and supporting verification [42]. Additionally, it can be useful to identify certain forms of incompleteness when the logic lacks a rule for a specific temporal event. For instance, in the context of smart contracts, works such as [43] and [44] provide tools to reason about incompleteness in specifications

by identifying when a smart contract does not specify what happens after a given temporal condition. However, temporal logic cannot eliminate the inherent incompleteness in contract theory, as explained above, particularly when it arises from changes in contractual relationships or the execution environment. With reference to the specification of an enforceable activity, temporal logic could identify the missed specification of consequences of a specified deadline, but if the parties decide not to specify such a deadline eventuality (i.e., eliminating the corresponding gateway and branch), the smart contract specification will not be considered incomplete, and the liveness of the process is preserved.

The next section examines how the various form of incompleteness relates to the structure of an EBP model, while Sect. IX discusses how incompleteness can be managed in the context of smart contracts.

## VIII. REPRESENTATION OF INCOMPLETENESS IN PROCEDURAL CONTRACTS

This section provides a mapping between the forms of incompleteness formalized from a legal perspective in the previous section and the notion of EBP. In particular, starting from the formal definition of EBP in Sect. IV and the elements of the graphical notation in Sect. V, a complete classification of the various forms of incompleteness needs to start from the structural elements of an EBP, namely the sets $\mathcal{N}$ and $\mathcal{C}$. However, we can notice that eventual changes in $\mathcal{C}$ can happen only in the presence of changes in $\mathcal{N}$, so we can start from the identification of incompleteness in the set of nodes. Indeed, for each activity or event, there can be exactly one incoming and one outgoing control flow edge, so in this regard, edges in $\mathcal{C}$ can be added or removed only in the correspondence of changes in $\mathcal{N}$. Similarly, for edges outgoing or ongoing from a gateway, we can have changes in the set $\mathcal{C}$ only for an incomplete specification of the gateway itself.

Starting from these considerations, we can identify the following forms of incompleteness: (1) a missing activity (task or sub-process), (2) an underspecified activity, (3) a missing deadline inside an enforceable activity, (4) a missing cost (incentive or penalty) inside an enforceable activity, (5) a missing timeout in a voting activity, (6) a missing event in an event-based gateway, (7) a missing eventuality in an exclusive gateway, (8) a missing branch in a parallel gateway. In the following, each of them is further defined and classified from a legal perspective. The general idea is that if the incomplete parts are clearly identifiable inside an EBP, then the incompleteness can be considered overt; otherwise, it is considered latent. Conversely, the intentionality will be identified based on the procedural nature of EBP and the underlying aim to make it directly interpretable by a DAIS.

*Definition 13 (Missing activity):* A missing activity happens in an EBP when a task or sub-process is not specified or is omitted inside a specified control-flow path. This form of incompleteness is typically *latent-unintended* since the set of tasks and sub-processes contained in the EBP make up the object of the contract and therefore, specifically, they must be determined and determinable to properly reflect the will that led to the manifestation of consent in the conclusion of the negotiated agreement.

This kind of incompleteness may occur, particularly in the case of unexpected changes to contractual relationships, leading to a renegotiation of the EBP and the deployment of a new version, with the consequent need to migrate existing running instances from the previous version to the new one. A check will be performed to ensure the running instances are compatible with the new version of EBP. Depending on the nature of the change, the DAIS will determine which instances can be migrated, which will continue with the previous EBP model, and which will eventually be stopped and restarted from the beginning. In the last case, compensation activities could be performed to reverse the effects of the previous partial execution.

With reference to the motivating example in Fig. 6, we can imagine a possible missing activity in the CU lane between the selection of the restaurant (i.e., $T_1$) and the placing of the order (i.e., $T_2$), which is related to the choice of payment method and its verification. Updating the EBP to include this task will trigger the deployment of a new EBP model and the migration of the running instances that will adhere to this change. Since the change occurs in a very early stage of the EBP, we can imagine that running instances can continue with the existing specification, while instances that have just started can be stopped and restarted with the new one.

Another possible missing activity, in this case relative to the CO lane, could regard an activity $a$ between the pick-up of the delivery (i.e., $T_{12}$) and the ride to the customer (i.e., $T_{16}$), which introduces a check performed by the CU about the state of the parcel and its integrity. In this case, an instance $\iota = \langle s_1, T_1, T_2, E_1, T_3, G_3^x(\text{“no”})\rangle$ does not contain in its possible execution paths $\Pi_\iota$ nor $T_{12}$ or $T_{16}$, so it can complete with the original specification $\mathcal{P}$ without any migration to $\mathcal{P}'$. Conversely, an instance $\iota'$ such that $\iota'[\![n]\!] = T_{10}$ should contain this new activity $a$ in its execution paths and can be migrated without problems, since it does not reach $T_{12}$ yet. If $\iota'$ evolves into $\iota''$ such that $\iota''[\![n]\!] = T_{12}$, then the network coalition could decide to complete the current instance with the original specification $\mathcal{P}$ or move to the new one $\mathcal{P}'$. In the last case, compensation could include halting the current delivery and conducting an immediate inspection of the parcel.

The second form of incompleteness is the presence of an *underspecified activity*. In this case, the EBP contains an activity represented as an atomic task that is revealed to be a more complex sub-process during execution.

*Definition 14 (Underspecified activity):* An underspecified activity happens in an EBP when an activity is modeled as an atomic task that instead needs to be further exploded into a subprocess to be properly executed in a procedural way, or as a sub-process where one or more parts need to be further subdivided. This form of incompleteness can be

*overt-intended* or *latent-unintended* depending on the reason justifying such a kind of incompleteness: the inconvenience or impossibility of further specifying the activity or the missing knowledge about its specific nature.

Referring back to the example in Fig. 7, we can imagine that the ''Prepare food'' task (i.e., $T_{14}$) may need to be further decomposed through a sub-process definition, for instance, if such activity includes many steps performed by different agents in the same ''Restaurant'' role. The reason for this underspecification may be intentional if, during the initial modeling, even if the parts recognize the complexity of this activity, they do not consider the need to specify all the details of such activity further. Conversely, it can be unintentional if they do not recognize the need to consider such a task as a compound, instead of an atomic one.

The presence of an underspecified activity could be managed by a DAIS only through a renegotiation of the EBP, which leads to an updated specification in which this activity is completely specified. Since the underspecification can only regard some specific execution paths, some running instances that do not involve it can continue with the original EBP specification, while all instances that recognize such missing parts will be migrated to the new one. Based on these considerations, a further specification of $T_{14}$ will not affect an instance $\iota = \langle s_1, T_1, T_2, E_1, T_3, G_3^x(\text{``no''})\rangle$, while for an instance $\iota'$ such that $\iota'[\![n]\!] = T_{14}$ which is currently executing this activity, the change could induce a cold migration of $\iota'$ or its continuation with the specification $\mathcal{P}'$. Finally, any other instance $\iota''$ which already terminates $T_{14}$ and includes the sub-path starting from the exclusive gateway $G_9^e$ will terminate with the original specification $\mathcal{P}'$.

Despite these two forms of incompleteness regarding general BPMN activities, some other forms of incompleteness can arise in the presence of the specific kinds of tasks composing an EBP.

*Definition 15 (A missing deadline inside an enforceable activity):*

A missing deadline within an enforceable activity may occur when the parties decide not to establish a temporal constraint for achieving task completion. This choice may be based on the need to provide flexibility in the execution of the contractual relationship. Thus, this type of incompleteness is, in most cases, *overt-intended*.

The agents composing a network coalition may decide to omit the temporal constraints, in particular for those enforceable activity belonging to paths that rarely occur. If this temporal constraint becomes necessary, various renegotiation stages may occur within a stable business relationship between the parties. Let us consider again the example in Fig. 7 and assume that also the ''Rate delivery service'' task (i.e., $T_{19}$) is modeled as an enforceable activity. In this case, the CU will be rewarded if he/she reviews the service, but the network coalition can decide not to set a deadline for this task, so the reward can be awarded independently from the delay in the task completion, assuming that the rate will arrive in a reasonable amount of time or that any delay is not relevant.

When a missing deadline is imposed, the network coalition can decide which running instances to migrate towards the updated specification. In particular, for those instances already running the involved enforceable activity, the network coalition needs to decide if the deadline applies to them or not. In the former case, if the deadline has already expired, the activity needs to be properly reverted. For instance, in case $T_{19}$ is intended as an enforceable activity and the deadline for its completion is defined ex-post, then this change could not influence an instance like $\iota = \langle s_1, T_1, T_2, E_1, T_3, G_3^x(\text{``no''})\rangle$, since $T_{19}$ does not belong to $\Pi_\iota$, while all the other instances that include $T_{19}$ in its possible paths but have not reached it could be migrated to the new specification without any problem. Finally, an instance $\iota'$, which is currently running $T_{19}$, could be migrated based on the decision of the network coalition.

Another form of incompleteness can arise inside an enforceable activity, which is the missing specification of the incentive or penalty associated with its execution.

*Definition 16 (A missing cost (incentive or penalty) inside an enforceable activity):*

A missing specification of incentive or penalty for an enforceable activity can happen when it is difficult to clearly state during the design phase the amount to correspond or pretend in case particular situations occur. In this case, the enforceable activity does not provide any information about the specific amount of tokens to be transferred through the reward or penalty message. This form of incompleteness needs to be avoided as much as possible during the specification of an EBP, making it essentially *overt-intended*, since this missing part is clear from the EBP and the parties need to agree about that underspecification.

Leaving the parties free to set penalties or incentives on a case-by-case basis in an ex-post way may raise disputes and uncertainties in the operational phase, completely missing the benefits of using a procedural contract. Indeed, in this case, neither party has knowledge of any applicable incentives or penalties for delays, errors, or substandard performance. However, some real-world cases exist that require leaving reward and penalty explicitly underspecified. They are represented, for instance, by a gamification system in which the issuance of loyalty rewards is not rigidly regulated but left to the discretion of the company.

In Fig. 7, the enforceable activity ''Ride to customer'' (i.e., $T_{16}$) assumes that the CO will pay a penalty in case of delays in the arrival at the customer. Even if the amount of penalty should be clear at the beginning of the EBP execution, there could be cases in which the precise amount cannot be clearly stated or specified during the modeling phase, for instance, for specific exceptional situations. In case an exceptional situation happens which is represented by the error event, the agents can decide through the voting activity to give the courier the chance to complete the order, but some penalties

could be applied, even if not the same associated with a missing delivery.

When a renegotiation occurs for dealing with such kind of incompleteness, we can assume that running instances will be moved to the new specification only if this enforceable activity has not been reached during the execution path or the task is stuck at this point waiting for further details, while all the instances that already execute it without any problem could proceed with the previous specification. With reference to the example mentioned before, any instance that, due to the evaluation of gateway $G_3^x$ as ''no'', will not have $T_{16}$ in its possible paths will not be interested in the change. Conversely, any instance that does not reach $T_{16}$ yet can be migrated without any problem. Instead, any instance that already reaches the task $T_{17}$ following activity $T_{16}$ could continue with the original specification. Finally, if an instance is currently running $T_{16}$, it will be migrated or not at the discretion of the network coalition.

We can also observe that this kind of renegotiation could clash with the requirement that each agent can only maintain a positive amount of tokens and that such an amount should be enough to pay eventual penalties specified for an enforceable activity he/she is involved in. Clearly, this requirement needs to be kept in mind when the network coalition reaches an agreement about the migration of existing EBP instances towards the updated enforceable activity, eventually identifying an incompatibility preventing the migration or the need for compensation activities consisting of the deposit of additional tokens by the involved parties.

Another form of flexibility that could be introduced inside an EBP may regard the possibility of defining a deadline for completing a voting activity.

*Definition 17 (A missing timeout in a voting activity):* A voting activity could require the presence of a deadline before which the required number of votes has to be provided, namely, the quorum will be reached. This form of incompleteness is essentially *overt-intended* since it is directly visible from the EBP, and it is typically due to an agreement between the parties that decides not to establish such a timeout value.

In Fig. 7, the ''Handle courier accident'' task (i.e., $T_{22}$) requires the network coalition to vote for a resolution, giving the CO the opportunity to complete the delivery anyway. The network coalition can decide in a first analysis not to put a specific deadline, assuming that the quorum could be reached in a reasonable but not predefined amount of time.

In case the agents inside a network coalition decide ex-post to impose a deadline for the vote, a re-negotiation phase is necessary, and the definition of a new version of the EBP needs to be concerted and deployed by a DAIS. This change will affect only the new instances or the running ones that are currently executing this voting activity. In particular, for these latter cases, if the network coalition decides to migrate such an instance and the deadline has already expired, the voting activity execution will clearly fail.

The voting activity $T_{22}$ is not part of all possible instances of $\mathcal{P}$, in particular, it is included only in the execution paths of instances where task $T_{16}$ raises an ''unexpected accident'' exception. Any instance that does not include task $T_{16}$ in its possible execution paths will not be affected by the change, while each instance that already includes $T_{17}$ does not need the further provided details, and any instance $\iota$ that is currently executing $T_{16}$, namely has $T_{16}$ as last element, could be migrated or not.

The last three forms of incompleteness regard the branches of the various gateways, and in particular, they are due to a missing event, a missing choice, or a missing parallel activity.

*Definition 18 (A missing event in an event-based gateway):* Event specifications are essential for the execution of activities and subprocesses. The absence of such a specification in a contract generates a *latent* form of incompleteness since it cannot be directly deduced from the EBP. Such a form of incompleteness could be *intended* if it can be interpreted as an initial strategic choice aimed at ensuring a high degree of flexibility in how the business relationship is executed. Conversely, it can be considered *unintended* if identifying all the possible events is not economically feasible or is practically impossible.

With reference to the EBP in Fig. 6, we can imagine that the initial event gateway $G_1^e$ in the CU lane can miss a message event ''Order suspended'' which activates another path requiring the CU to perform some additional tasks, such as providing a valid and alternative payment method, before the order can be confirmed.

A renegotiation step to include a missing event will produce a new version of the EBP model, which can involve running instances in different ways: the network coalition could decide to complete their execution with the previous specification assuming that this event has not happened, or it could decide to move the instances towards the new one if the specific gateway has not been reached. Depending on the nature of the event, the network coalition could also decide to completely revert a running instance, since this event has not been evaluated but is considered relevant for the correct execution of the current running instance. Following these considerations, an instance of the process $\mathcal{P}$ in Fig. 6 that already evaluates $G_1^e$ can continue with the original specification, unless the network coalition decides to revert the process and evaluate the gateway again. Conversely, an instance $\langle s_1, T_1, T_2, E_1, T_3 \rangle$, which has not evaluated the gateway yet, could be migrated to the new version without any problem.

A similar kind of incompleteness occurs when a missing eventuality is present in an exclusive gateway, which can happen when a further alternative has not been specified.

*Definition 19 (A missing eventuality in an exclusive gateway):* Similar to the previous case, an EBP can contain a missing branch in an exclusive gateway, corresponding to a missing eventuality in the EBP specification. The classification of this kind of incompleteness is similar to the previous one: *latent* since it is not visible, and *intentional*

or *unintentional* depending on the reason from which it originates.

The EBP in Fig. 6 contains an exclusive gateway $G_3^x$ in the R lane; it may become evident ex-post the need to manage through it another outcome besides the "yes" and "no" ones, which represents the need to acquire further information before directly accepting or rejecting the order. Similarly to the previous case, the missed eventuality could be considered relevant or not for the running instances. In the first case, running instances could be migrated and eventually reverted if necessary; in the second case, they can be completed by using the original specification. In the case of gateway $G_3^x$, any instance of the process will include such a gateway and, therefore, could be affected by the change in some way. However, some instances could be migrated without any problem since they have not reached it yet, as, for instance, $\iota = \langle s_1, T_1, T_2, E_1 \rangle$, or can be migrated because they are currently evaluating such a gateway, namely their sequential representation terminates with $G_3^x$. Other instances could be completed with the previous specification since they already evaluate $G_3^x$, like $\iota' = \langle s_1, T_1, T_2, E_1, T_3, G_3^x(\text{"no"}), E_2 \rangle$.

*Definition 20 (A missing branch in a parallel gateway):* A parallel gateway is designed to execute all paths simultaneously. If a branch is missing, the model may be formally wrong or may not correctly represent the desired behavior. Therefore, if the process expects to activate multiple paths but one is missing, the execution may stall or be inconsistent. This results in *latent-unintended* incompleteness, where benefits are not obtained by any of the actors involved in the execution of the contractual agreement.

An example of this form of incompleteness can be found in the example of Fig. 7 where a parallel gateway is started in the R lane which includes a sequence of activities for the restaurant (i.e., $T_{14}$) to be done in parallel with another sequence of activities depending on the courier (i.e., $G_8^e$, $T_{13}$, or $E_{26}$). In this case, we can imagine a missing branch including a path performed by the restaurant to require some additional items that are not prepared internally but are offered to the customer.

The introduction of a missing branch in a parallel gateway indicates that some relevant activities have been omitted from the original specification. In this case, running instances need to adhere to the new one and their execution can: (i) simply move to the new EBP specification if the interested gateway has not been reached yet, (ii) moved to the new EBP specification if the gateway has been started just now, so the missed parallel branch can immediately start, (iii) reverted if the gateway has been passed or has been started from a while, preventing the instances from respecting some eventual deadlines.

In the EBP in Fig. 6-7, there are some instances like $\langle s_1, T_1, T_2, E_1, T_3, G_3^x(\text{"no"}) \rangle$ that will not include the parallel gateway following $E_{10}$ and so will not be affected by the change. Other instances, like $\langle s_1, T_1, T_2, E_1, T_3, G_3^x(\text{"yes"}) \rangle$ could contain such a parallel gateway but have not reached it, so they can be easily migrated. Conversely, an instance that has just started the parallel gateway can also easily start this new branch in parallel. Finally, any instance reaching the gateway $G_9^e$ has already completed the parallel execution connected with the affected gateway, and the network coalition could decide to revert the instance immediately before the join parallel gateway to execute the missed branch, or to continue the execution without considering it.

This discussion about the various possible forms of incompleteness in an EBP suggests that an updated version of the contract requires managing the migration of current running instances from a previous version to the new one. The discussion also suggests that such a migration could involve only a subset of the running instances, depending on their current execution state, and sometimes it needs to be concerted by the network coalition. In the following, we try to summarize and formalize the possible recovery cases discussed above. We leave for future work the eventual definition of correctness regarding such a possible dynamic migration. Several works in the literature identify correctness criteria for determining when changes are allowed (or safe) on in-progress workflow instances, such as the one in [45], which needs to be properly extended and contextualized in the field of contractual incompleteness.

*Construction 1 (Recovery from incompleteness):* Given an EBP $\mathcal{P} = \langle \mathcal{N}, \mathcal{C}, \mathcal{R}, \tau, \gamma_t, \gamma_m, \epsilon, \rho \rangle$, where $\mathcal{N} = \mathcal{A} \cup \mathcal{G} \cup \mathcal{E}$, and its re-negotiated version $\mathcal{P}' = \langle \mathcal{N}', \mathcal{C}, \mathcal{R}, \tau' \gamma_t, \gamma_m, \epsilon, \rho \rangle$ where $n' \in \mathcal{N}' \setminus \mathcal{N}$ is the node updated or added in the new specification $\mathcal{P}'$. For each instance $\iota$ of $\mathcal{P}$:

- $\mathcal{N}' = \mathcal{A}' \cup \mathcal{G} \cup \mathcal{E}$, where $\mathcal{A}' = \mathcal{A} \cup \{n\}$, namely $n$ is a missing activity added in $\mathcal{P}'$. Let us denote as $x_i$ the node $\bullet n$ and as $x_j$ the node $n\bullet$ in the new specification $\mathcal{P}'$, respectively.
  - If $x_i, x_j \notin \Pi_\iota$, then the two activities are not part of the possible execution paths of $\iota$, and hence also $n$ will not be part of them. Therefore, in this case, no migration is needed, and $\iota$ could be completed using the previous specification.
  - If $x_i, x_j \in \Pi_\iota \wedge (x_i \notin \iota \vee (x_i \in \iota \wedge x_j \notin \iota))$, then instance $\iota$ has not already reached the point of the specification subject to modification. Therefore, $\iota$ can be migrated to the new specification $\mathcal{P}'$.
  - If $x_i, x_j \in \Pi_\iota \wedge x_j \in \iota$, then $\iota$ has already passed the execution point in which $n$ should be executed considering the new specification $\mathcal{P}'$. In this case, the network coalition can decide to complete $\iota$ using the specification $\mathcal{P}$ or to require the DAIS to perform some compensation activities to rollback some effects and turn back the execution of $\iota$ immediately after the completion of $x_i$.
- $\mathcal{N}' = \mathcal{A}' \cup \mathcal{G} \cup \mathcal{E} \wedge \mathcal{A}' = \mathcal{A} \setminus \{n\} \cup \{n'\}$, namely $n'$ is an updated version of an activity, or $\mathcal{N}' = \mathcal{A} \cup \mathcal{G}' \cup \mathcal{E} \wedge \mathcal{G}' = \mathcal{G} \setminus \{n\} \cup \{n'\}$, namely $n'$ is an updated version of a gateway.
  - If $n \notin \Pi_\iota$, then the updated node $n'$ is not part of the possible execution paths of $\iota$. Therefore, in this

case, no migration is needed, and $\iota$ could be completed using the previous specification.

- If $n \in \Pi_\iota \wedge n \notin \iota$, it means that $\iota$ does not yet reach the point involving $n$, so the instance could be directly migrated to the new specification $\mathcal{P}'$ by replacing $n$ with $n'$, provided that all agents continue to have an amount of tokens in their wallet sufficient to cover all eventual penalties, particularly if $n'$ is a update of $n$ where all missing penalties have been specified. Specific compensation actions should be implemented by the DAIS to satisfy such property.
- If $n \in \Pi_\iota \wedge n\bullet \in \iota$, it means that $n$ has already been performed, so the process instance was able to proceed without the further details provided by $n'$.
  - $n \in \mathcal{A}$, the instance $\iota$ can be completed with the original specification $\mathcal{P}$.
  - $n \in \mathcal{G}$ and $n'$ is an updated version of an event-based gateway $n$ that contains a missing event, or of an exclusive gateway $n$ that contains a missing eventuality, or of a parallel gateway $n$ that contains a missing branch. For the event-based or the exclusive gateway, the network coalition could decide to complete $\iota$ without considering the update, or it could decide to stop and revert the execution immediately before $n$ if the new event, or the new eventuality, is considered relevant for all running instances. For the parallel gateway, in the case a migration is chosen by the network coalition, the new branch can be initiated as soon as possible in parallel with the existing ones.
- If $n \in \Pi_\iota \wedge n \in \iota \wedge n\bullet \notin \iota$, it means that $\iota$ is running $n$. In this case, the network coalition could decide to complete $\iota$ with the previous version $n$ or with the new version $n'$ depending on the incompleteness type and the characteristics of the running instance $\iota$:
  - $\mathcal{N}' = \mathcal{A}' \cup \mathcal{G} \cup \mathcal{E}$, where $\mathcal{A}' = \mathcal{A} \setminus \{n\} \cup \{n'\}$ and $n'$ is a sub-process redefining an underspecified activity $n$. In this case, $\iota$ is blocked in $n$ requiring the additional details provided by $n'$, so $\iota$ will be migrated to the new specification, and the network coalition could identify the appropriate compensations that the DAIS should perform for the steps already done, if necessary.
  - $\mathcal{N}' = \mathcal{A}' \cup \mathcal{G} \cup \mathcal{E}$, where $\mathcal{A}' = \mathcal{A} \setminus \{n\} \cup \{n'\}$ and $n' = \langle t, i, p, d\rangle$ is the updated version of an enforceable activity $n$, or $n' = \langle c, s, d\rangle$ is the updated version of a voting activity $n$, where the missing deadline has been specified and set equal to $d$. In this case, the network coalition could decide to complete $\iota$ without adding the new details about the deadline, or it could decide to migrate towards the new specification $\mathcal{P}'$. If the latter option is chosen, a consistency check will be performed between the timestamp associated with $n$ in $\iota$ and the deadline $d$. If the deadline has already expired, then the DAIS will raise the corresponding exception; otherwise, the execution will continue as if the deadline was present from the beginning.
  - $\mathcal{N}' = \mathcal{A}' \cup \mathcal{G} \cup \mathcal{E}$, with $\mathcal{A}' = \mathcal{A} \setminus \{n\} \cup \{n'\}$ and $n' = \langle t, i, p, d\rangle$ is the updated version of enforceable activity $n$ whose incentive or penalty has been specified and set equal to $i$ or $p$, respectively. In this case, $\iota$ could be migrated to the new specification by adding the details related to the incentives and penalties. If such details cannot be added automatically, since a partial execution of $n$ has already been performed or some actors do not have enough tokens in their wallet to cover such new penalties, the DAIS could implement some appropriate compensation steps to recover the execution and incorporate the new information.
  - $\mathcal{N}' = \mathcal{A} \cup \mathcal{G}' \cup \mathcal{E}$, with $\mathcal{G} \setminus \{n\} \cup \{n'\}$ and $n'$ is the updated version of $n$ whose set of events (or eventualities) has been updated. It means that $\iota$ is evaluating the events (or eventualities) managed by the gateway $n$. In this case, $\iota$ could be stopped and migrated to the new specification to also handle the new event (or eventuality).
  - $\mathcal{N}' = \mathcal{A} \cup \mathcal{G} \cup \mathcal{E}$, with $\mathcal{G} \setminus \{n\} \cup \{n'\}$ and $n'$ is the updated version of $n$ whose set of parallel branches has been updated. The network coalition could decide to ask the DAIS to start in parallel also the newly added branch, without the need to revert the work already done, or to complete $n$ with only the existing ones, following the original specification $\mathcal{P}$.

As discussed above, the decision to migrate current instances should be subject to a formal check about the compatibility between the current instance execution and the proposed update. This last investigation is left as future work.

## IX. HANDLING INCOMPLETENESS IN PROCEDURAL CONTRACTS

This section discusses how incompleteness and renegotiability could be managed in a blockchain environment. Indeed, smart contracts are immutable by design, and once they have been deployed, they cannot be changed anymore. However, to ensure some level of flexibility while promoting immutability, certain mechanisms have been consolidated in recent years, such as the concept of proxies.

With reference to the form of incompleteness discussed in Sect. VIII, we can categorize them from the implementation point of view into incompleteness that: (I1) need the adjunction of a new function, (I2) need the change of a function implementation, and (I3) need to change the arguments of a function call.

Category (I1) is essentially represented by the first form of incompleteness, namely, the presence of a *missing*

*activity* (see Def. 13). In this case, as already explained in Sect. VIII, a renegotiation of the entire EBP is necessary. This can be translated into the implementation and deployment of a new smart contract, which includes the additional function. In particular, the inheritance mechanism could be exploited, so that the new smart contract representing $\mathcal{P}'$ will automatically inherit all the functionalities of $\mathcal{P}$ while adding the required new function. At this point, the DAIS could decide to migrate instances to the new smart contract by updating the reference address or continue the execution with the old address.

Category (I2) includes the case of an underspecified activity (see Def. 14), a missing event in an event-based gateway (see Def. 18), a missing eventuality in an exclusive gateway (see Def. 19), and a missing branch in a parallel gateway (see Def. 20). In this case, the existence and the signature of the smart contract function remain the same, but its body needs to be updated. This is a specific case in which the proxy mechanism could be exploited to accommodate and manage incompleteness. As prescribed by the proxy pattern [34], in this case, we have a first wrapper contract (i.e., a ''proxy'') that users interact with, which essentially performs forwarding transactions to and from a second contract that actually contains the logic. In this case, the logic contract can be replaced, but the proxy remains unchanged. Both contracts are still immutable, namely, their code cannot be changed, but the logic contract can simply be swapped with another contract. In particular, the proxy contract stores the address of an implementation contract and delegates all calls to it, for instance, through a function like the `delegatecall` in Solidity. In this way, the implementation executes the logic, but within the proxy storage context, namely, the state is stored in and read from the proxy storage.

Finally, category (I3) includes the case of a missing deadline inside an enforceable activity (see Def. 15), or a missing cost (incentive or penalty) inside an enforceable activity (see Def. 16), or a missing timeout in a voting activity (see Def. 17). In case an enforceable activity or a voting activity has been implemented following the structure in Sect. VI, the corresponding smart contract function includes all the required arguments in its signature, but during its call, only a subset of them has been properly specified, while others, for instance, include only a default or dummy value. In this case, the function body could implement an exception mechanism for handling the scenario where such a parameter is necessary, allowing its execution to be reverted and eventually providing a new complete call. Conversely, if the missing arguments are omitted from the function signature, a new version of the smart contract must be deployed, as in case I1.

This section provides an initial discussion on the possible implementation of renegotiable smart contracts that can handle incomplete procedural contracts. The discussion does not claim to be exhaustive, but lays the basis for an actual EBP implementation.

## X. CONCLUSION

The advent of network coalition as a concerted form of coordination between economic agents poses new challenges and opportunities in the field of business process modeling and execution. In particular, in the literature, the notion of Exogenous Business Process (XBP) has been introduced to denote a cooperative, stable process managed by a network coalition to pursue a common goal. XBPs can be specified by means of potentially incomplete, procedural contracts, which can be modeled by a set of Enforceable Business Processes (EBPs). An EBP addresses the procedural aspects of an XBP through the use of blockchain-based smart contracts, and the incompleteness requirements by properly extending the immutability characteristic of the latter. In this paper, we take a step forward in this direction by providing a complete formalization of the notion of EBP, which includes both the procedural aspects and the incompleteness requirements. Relatively, to the first aspect, a proper extension of the BPMN graphical language is provided, which includes, in particular, two specific kinds of tasks: enforceable activity and voting activity, that extend the traditional notions of activity and tasks with the facilities of blockchain-based smart contracts, like enforceable tasks, voting power, and token transfers. Conversely, for incompleteness and renegotiability needs, a mapping is provided towards the forms of incompleteness formalized in the legal domain for contracts, along with a detailed discussion about their management by a Decentralized Autonomous Information System (DAIS).

A first discussion is also made about the possible implementation of an EBP in terms of blockchain-based smart contracts and how incompleteness could be managed and combined with the immutability characteristics of blockchain. As future work, we plan to exploit this formalization for the development of a complete DAIS based on the use of blockchain-based smart contracts, which can be easily renegotiated while providing traceability and immutability.

An additional future work includes the exploration of the possibility of integrating tools based on temporal logic for detecting some forms of incompleteness in smart contracts, and eventually of extending them for reducing the presence of other forms of incompleteness identified in this paper, even if they cannot be completely removed as explained in Sect. VII. Moreover, the most promising extension point is the use of model checking tools for verifying the compatibility and correctness of the updated specification with the running instances, with the aim to determine the extent to which they can be migrated towards the new specification.

## ACKNOWLEDGMENT

The authors would like to thank Dr. Francesca Zerbato for her invaluable help during the initial definition of the BPMN extension needed for managing EBP, as well as during the formalization of the first version of the motivating example.

This manuscript reflects only their views and opinions, neither European Union nor European Commission can be considered responsible for them.

## REFERENCES

[1] D. Milošević, "Neither market nor hierarchy: Network forms of organization," *Res. Organizational Behav.*, vol. 2, pp. 295–336, May 2014.
[2] H. A. Simon, "Organizations and markets," *J. Econ. Perspect.*, vol. 5, no. 2, pp. 25–44, 1991.
[3] O. E. Williamson, "Comparative economic organization: The analysis of discrete structural alternatives," *Administ. Sci. Quart.*, vol. 36, no. 2, pp. 269–296, Jun. 1991.
[4] S. Migliorini, M. Gambini, C. Combi, and M. La Rosa, "The rise of enforceable business processes from the hashes of blockchain-based smart contracts," in *Enterprise, Business-Process and Information Systems Modeling*. Cham, Switzerland: Springer, 2019, pp. 130–138.
[5] O. Hart, "Incomplete contracts and control," *Amer. Econ. Rev.*, vol. 107, no. 7, pp. 1731–1752, Jul. 2017.
[6] O. E. Williamson, *The Economic Institutions of Capitalism: Firms, Markets, Relational Contracting*. New York, NY, USA: Free Press, 1985.
[7] J. Mendling, "Towards blockchain support for business processes," in *Proceedings of Business Modeling and Software Design (BMSD)*, vol. 319. Cham, Switzerland: Springer, 2018, pp. 243–248.
[8] J. Mendling et al., "Blockchains for business process management—Challenges and opportunities," *ACM Trans. Manage. Inf. Syst.*, vol. 9, no. 1, pp. 1–16, 2018.
[9] Q. Chen and M. Hsu, "Inter-enterprise collaborative business process management," in *Proc. 17th Int. Conf. Data Eng.*, 2001, pp. 253–260.
[10] D. Werth, W. Philipp, and P. Loos, "Distribution and composition of collaborative business processes through peer-to-peer networks," in *Proc. Bus. Process Manage. Workshops*, 2009, pp. 597–608.
[11] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, no. 10, pp. 46–52, 2003.
[12] M. Weske, *Business Process Management: Concepts, Languages, Architectures*, 2nd ed., Cham, Switzerland: Springer, 2012.
[13] T. J. Strader, F. Lin, and M. J. Shaw, "Information infrastructure for electronic virtual organization management," *Decis. Support Syst.*, vol. 23, no. 1, pp. 75–94, 1998.
[14] P. Robinson, Y. Karabulut, and J. Haller, "Dynamic virtual organization management for service oriented enterprise applications," in *Proc. Int. Conf. Collaborative Comput., Netw., Appl. Worksharing*, 2005, pp. 10–16.
[15] I. Weber, J. Haller, and J. A. Mulle, "Automated derivation of executable business processes from choreographies in virtual organisations," *Int. J. Bus. Process Integr. Manage.*, vol. 3, no. 2, pp. 85–95, 2008.
[16] J. A. Garcia-Garcia, N. Sánchez-Gómez, D. Lizcano, M. J. Escalona, and T. Wojdynski, "Using blockchain to improve collaborative business process management: Systematic literature review," *IEEE Access*, vol. 8, pp. 142312–142336, 2020.
[17] F. Stiehle and I. Weber, "Blockchain for business process enactment: A taxonomy and systematic literature review," in *Proc. Bus. Process Manage., Blockchain, Robotic Process Autom., Central Eastern Eur. Forum*. Cham, Switzerland: Springer, 2022, pp. 5–20.
[18] C. Di Ciccio, G. Meroni, and P. Plebani, "Business process monitoring on blockchains: Potentials and challenges," in *Enterprise, Business-Process and Information Systems Modeling*. Cham, Switzerland: Springer, 2020, pp. 36–51.
[19] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling, "Untrusted business process monitoring and execution using blockchain," in *Proc. Bus. Process Manage. (BPM)*. Cham, Switzerland: Springer, 2016, pp. 329–347.
[20] L. García-Bañuelos, A. Ponomarev, M. Dumas, and I. Weber, "Optimized execution of business processes on blockchain," in *Proc. Bus. Process Manage. (BPM)*. Cham, Switzerland: Springer, 2017, pp. 130–146.
[21] H. Nakamura, K. Miyamoto, and M. Kudo, "Inter-organizational business processes managed by blockchain," in *Proc. Web Inf. Syst. Eng.* Cham, Switzerland: Springer, 2018, pp. 3–17.
[22] J. Ladleif, M. Weske, and I. Weber, "Modeling and enforcing blockchain-based choreographies," in *Proc. Bus. Process Manage.* Cham, Switzerland: Springer, 2019, pp. 69–85.
[23] H. Schonenberg, R. S. Mans, N. Russell, N. Mulyar, and W. van der Aalst, "Process flexibility: A survey of contemporary approaches," in *Business Process Management*. Cham, Switzerland: Springer, 2008, pp. 16–30.
[24] M. Borkowski, W. Fdhila, M. Nardelli, S. Rinderle-Ma, and S. Schulte, "Event-based failure prediction in distributed business processes," *Inf. Syst.*, vol. 81, pp. 220–235, Mar. 2019.
[25] N. Russell, W. M. P. van der Aalst, and A. H. M. ter Hofstede, "Workflow exception patterns," in *Advanced Information Systems Engineering.*, vol. 61. Cham, Switzerland: Springer, 2006, pp. 288–302.
[26] M. Reichert and B. Weber, *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Cham, Switzerland: Springer, 2012.
[27] C. D. Ciccio, A. Marrella, and A. Russo, "Knowledge-intensive processes: Characteristics, requirements and analysis of contemporary approaches," *J. Data Semantics*, vol. 4, no. 1, pp. 29–57, 2014.
[28] F. Loukil, K. Boukadi, M. Abed, and C. Ghedira-Guegan, "Decentralized collaborative business process execution using blockchain," *World Wide Web*, vol. 24, no. 5, pp. 1645–1663, Sep. 2021.
[29] O. López-Pintado, M. Dumas, L. García-Bañuelos, and I. Weber, "Controlled flexibility in blockchain-based collaborative business processes," *Inf. Syst.*, vol. 104, Feb. 2022, Art. no. 101622.
[30] Object Management Group. (Dec. 2013). *Business Process Model and Notation (BPMN), Version 2.0.2*. [Online]. Available: https://www.omg.org/spec/BPMN/2.0.2/PDF/
[31] V. Buterin. (2014). *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. [Online]. Available: https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf
[32] S. N. Khan, F. Loukil, C. Ghedira-Guegan, E. Benkhelifa, and A. Bani-Hani, "Blockchain smart contracts: Applications, challenges, and future trends," *Peer-Peer Netw. Appl.*, vol. 14, no. 5, pp. 2901–2925, Sep. 2021.
[33] M. B. Saif, S. Migliorini, and F. Spoto, "Blockchain-based multirole authentication and authorization in smart contracts with a hierarchical factory pattern," in *Proc. 6th Int. Conf. Blockchain Comput. Appl. (BCCA)*, Nov. 2024, pp. 22–29.
[34] S. Azimi, A. Golzari, N. Ivaki, and N. Laranjeiro, "A systematic review on smart contracts security design patterns," *Empirical Softw. Eng.*, vol. 30, no. 4, p. 95, Apr. 2025.
[35] F. Ghiyami Pour, G. Costa, and L. Galletta, "Welcome back: A systematic literature review of smart contract reentrancy and countermeasures," *Blockchain, Res. Appl.*, vol. 6, Jul. 2025, Art. no. 100347.
[36] P. S. Atiyah, *Essays on Contract*. Oxford, U.K.: Clarendon Press, 1988.
[37] S. Wang, "Incomplete contracts with disparity, uncertainty, information and incentives," *SSRN Electron. J.*, Aug. 2019, doi: 10.2139/ssrn.3433086.
[38] K. E. Spier, "Incomplete contracts and signalling," *RAND J. Econ.*, vol. 23, no. 3, pp. 432–443, 1992. [Online]. Available: http://www.jstor.org/stable/2555872
[39] O. E. Williamson, "Transaction cost economics: How it works; where it is headed," *De Economist*, vol. 146, no. 1, pp. 23–58, Apr. 1998.
[40] R. T. Holden and A. Malani, "Can blockchain solve the hold-up problem in contracts?" Coase-Sandor Inst. Law Econ., Univ. Chicago Law School, Chicago, IL, USA, Working Paper 846, 2017, doi: 10.2139/ssrn.3093879.
[41] A. Pnueli, "The temporal logic of programs," in *Proc. 18th Annu. Symp. Found. Comput. Sci. (SFCS)*, Sep. 1977, pp. 46–57.
[42] A. Cimatti and S. Tonetta, "A temporal logics approach to contract-based design," in *Proc. Archit.-Centric Virtual Integr. (ACVI)*, Apr. 2016, pp. 1–3.
[43] P. Bottoni, A. Labella, and R. Pareschi, "A formal model for ledger management systems based on contracts and temporal logic," *Blockchain, Res. Appl.*, vol. 3, no. 1, Mar. 2022, Art. no. 100062.
[44] J. Stephens, K. Ferles, B. Mariano, S. Lahiri, and I. Dillig, "SmartPulse: Automated checking of temporal properties in smart contracts," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 555–571.
[45] C. Ellis, K. Keddara, and G. Rozenberg, "Dynamic change within workflow systems," in *Proc. Conf. Organizational Comput. Syst. (COCS)*, 1995, pp. 10–21.

**SARA MIGLIORINI** received the master's and Ph.D. degrees in computer science from the University of Verona, Italy, in 2007 and 2012, respectively. From 2012 to 2019, she was a Postdoctoral Research Associate with the Department of Computer Science, University of Verona, where she has been an Assistant Professor, since 2019. Her main research interests include information systems, big data systems and analytics, recommendation systems, machine learning, and blockchain technology.

**MAURO GAMBINI** received the master's and Ph.D. degrees in computer science from the University of Verona, Verona, Italy, in 2007 and 2012, respectively. He is currently a Postdoctoral Research Associate with the Department of Computer Science, University of Verona. His current research interests include the database and information system field, big data analytics, and blockchain technology.

**VERONICA PATERNOLLI** received the bachelor's degree in labor consultancy from the University of Padua, Italy, in 2021, and the master's degree in law for technologies from the University of Verona, Italy, in 2023, where she is currently pursuing the industrial Ph.D. (PNRR) degree in computer science. She holds a solid legal background. Her research investigates the intersection of legal language and computational reasoning, with particular attention to smart contracts and AI.

**MILA DALLA PREDA** received the master's and Ph.D. degrees in computer science from the University of Verona, Italy, in 2003 and 2007, respectively. She is currently a Full Professor with the Department of Computer Science, University of Verona. Her research interests include abstract interpretation and formal methods for program analysis, semantics, and transformation, as well as software protection—covering code obfuscation, watermarking, security, and intellectual-property preservation—automatic malware detection and model checking, with recent work extending toward the interface between software security and law and technology.