Efficient Exploration of Cyber-Physical System Architectures Using Contracts and Subgraph Isomorphism

Yifeng Xiao*, Chanwook Oh*, Michele Lora†*, Pierluigi Nuzzo*
*Dept. of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA, US
†Dept. of Engineering for Innovation Medicine, University of Verona, Italy
{yifengx|chanwooo|nuzzo}@usc.edu, michele.lora@univr.it

***Abstract*—We present ContrArc, a methodology for the exploration of cyber-physical system architectures aiming to minimize a cost function while adhering to a set of heterogeneous constraints. We assume a system topology, defined as a graph, where components (nodes) are selected from an implementation library, and connections between components (edges) are drawn from a finite set of possible connection choices. ContrArc uses assume-guarantee contracts to formalize different viewpoints in the system requirements, such as timing and power consumption, as well as the interface of different components, and translate the exploration problem into a mixed integer linear programming problem. It then searches for efficient solutions by relying on contract decompositions and a method based on subgraph isomorphism to iteratively prune infeasible architectures out of the search space. Experiments on a reconfigurable production line and an aircraft power distribution network show up to two orders of magnitude acceleration in architectural exploration with respect to comparable approaches.**

***Index Terms*—Assume-guarantee contracts, design space exploration, cyber-physical systems, subgraph isomorphism.**

I. Introduction

Distinguished by their ability to process data and execute closed-loop control actions in real time, cyber-physical systems (CPSs) have been deployed in various domains. However, the design of CPSs remains challenging for several reasons, a major one being the heterogeneous design space, too large to be explored efficiently. We need comprehensive frameworks enabling the exploration of multiple dimensions of the design space, factoring in critical system requirements in terms of timing, workload, and energy consumption, among others, while enhancing interoperability, often hindered by the proliferation of domain-specific languages and tools [1], [2].

Several approaches have been proposed to aid the design of CPSs [3]. Among these, the ones based on assume-guarantee (A/G) contracts have been shown to offer effective and rigorous mechanisms for design abstraction and composition across heterogeneous domains, scalable requirement analysis, and system verification via compositional reasoning [4]–[6]. In the context of architecture exploration, effective representations of system architectures have been proposed in terms of parametrized graphs, leading to formulations of the design problem as a mixed integer linear programming (MILP) problem, for which efficient encodings and solution strategies have also been devised [7]–[10]. However, as the computational cost of solving MILP problems grows exponentially with the size of the architecture, devising novel methods that can significantly reduce the exploration cost and enhance scalability is highly desirable.

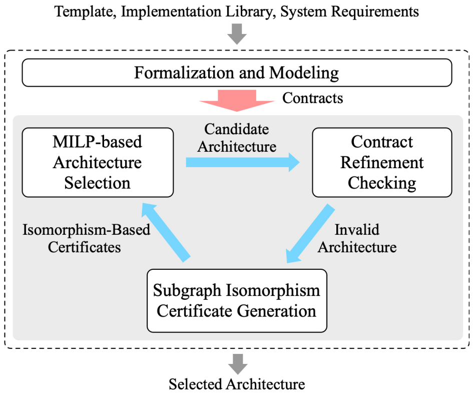

Template, Implementation Library, System Requirements
Formalization and Modeling
Contracts
MILP-based Architecture Selection
Candidate Architecture
Contract Refinement Checking
Isomorphism-Based Certificates
Invalid Architecture
Subgraph Isomorphism Certificate Generation
Selected Architecture

Fig. 1 Contract-based architecture exploration.

In this paper, we introduce , a contract-based methodology for architecture exploration. Inspired by previous approaches to architecture exploration, we also adopt a "lazy" coordination scheme between MILP solving and a procedure that can generate infeasibility certificates to prune out invalid candidates and accelerate the search [8]. The centerpiece of our solution is, however, a novel *certificate generation method that combines a contract refinement checking procedure with a subgraph isomorphism algorithm.*

As shown in Fig. 1, given an initial template for the system architecture, including a set of components and their possible interconnections, a library of implementations, and the system requirements, formally captures the requirements as well as their allocation to the components in terms of A/G contracts, and explores the design space by formulating a MILP problem. It first identifies a candidate architecture that satisfies "local" interconnection and component-level requirements. It then checks whether this candidate also satisfies "global" system requirements via contract refinement checking. If the refinement holds, returns the candidate architecture as the optimal one. Otherwise, it leverages subgraph isomorphism to identify similar invalid candidates and iteratively exclude them from the search space by generating additional constraints for the MILP problem.

Remarkably, can leverage contract-based decompositions of architectures and requirements to enhance scalability in two ways, by either breaking the original MILP problem or the contract refinement checking problem into smaller subproblems. We evaluate the performance of on two kinds of CPSs, a reconfigurable production line and an aircraft power

This research was supported in part by the US NSF under awards 1846524 and 2139982, the ONR under award N00014-20-1-2258, the Okawa Research Grant, and Siemens. It also received funding from the EU's Horizon 2020 program under the Marie Skłodowska-Curie grant agreement no. 894237.

distribution network, showing up to two orders of magnitude speedup with respect to comparable approaches.

II. Preliminaries

We introduce below the main concepts used by .

A. Assume-Guarantee (A/G) Contracts

Let M denote a component, i.e., an element of a system, characterized by a set of variables V_c and a set of behaviors $[[M]]$ over V_c. A contract C formally captures a set of specifications for M using a triple $C = (V_c, A, G)$, where A and G are sets of behaviors over V_c. A is the assumptions on the environment of M while G is the guarantees provided by M, given that the assumptions are satisfied. We say that M is a valid implementation of C, i.e., $M \models C$, if all the behaviors of M are included in the guarantees given the assumptions of C, i.e., $[[M]] \subseteq G \cup \overline{A}$. We say that component E_c is a valid environment of C if all the behaviors of E_c are contained in the assumptions of C. A contract is *consistent* if and only if there exists a valid implementation, i.e., $G \cup \overline{A} \neq \emptyset$, and it is *compatible* if there exists a valid environment, i.e., $A \neq \emptyset$.

The *refinement* relation between contracts allows reasoning about the replaceability of a contract by another contract. A contract C' can be replaced by a contract C when C refines C', written $C \preceq C'$, i.e., C has weaker assumptions and stronger guarantees. Contracts can be combined according to different rules. The *composition* ($\otimes$) of contracts can be used to build a system-level contract out of multiple component-level contracts. The *conjunction* ($\wedge$) of contracts can be used to combine contracts capturing multiple *viewpoints*, e.g., timing, power, and workload. We use contract refinement to evaluate whether a composition of component-level contracts (selected via architectural exploration) meets the requirements at the system level. We refer the readers to the literature for further details [4].

B. Architecture Exploration

We build on a formal notion of system architecture in the literature [8] as follows.

Definition 1 (System Architecture [8]). *A system architecture is a directed graph $\mathcal{G} = (V, E)$, where V denotes a set of components (nodes) and E denotes a set of connections (edges) between the components. Each node $v_i \in V$ represents either a functional component or a connector in the system. An edge $e_{i,j} \in E$ represents a connection from v_i to v_j, with $i, j \in \{1, ..., |V|\}$, where $|V|$ is the cardinality of V.*

A *template* $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$ is a system architecture assembled from a set of labeled nodes $V_\mathcal{T}$, representing different types of components, and potential interconnections $E_\mathcal{T}$ between them. A node v_i labeled with a type k_i can be mapped to implementations in the subset $\mathcal{L}_{k_i}$ of the *implementation library* $\mathcal{L}$. Each component node and implementation has a set of *attributes*, such as cost, latency, and throughput.

Definition 2 (Graph Partition and Component Type [8]). *A partition $\Pi = \{\Pi_1, \Pi_2, ...\}$ of an architecture $\mathcal{G}$ is a set of nonempty subsets of V such that V is the disjoint union of these subsets. The index k_i for each Π_{k_i} represents the type, e.g., source or sink. A node $v \in \Pi_{k_i}$ with type k_i can only be mapped to an implementation in $\mathcal{L}_{k_i}$, where $\mathcal{L} = \bigcup_{k_i} \mathcal{L}_{k_i}$.*

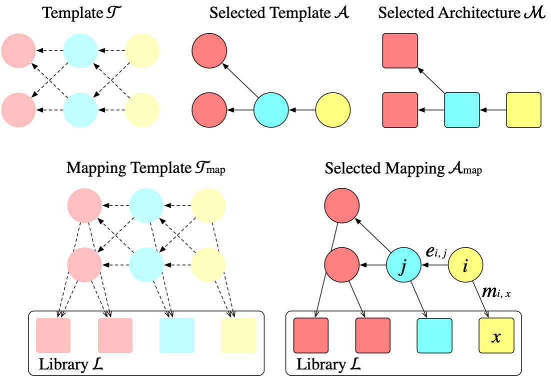

Template $\mathcal{T}$
Selected Template $\mathcal{A}$
Selected Architecture $\mathcal{M}$
Mapping Template $\mathcal{T}_{map}$
Selected Mapping $\mathcal{A}_{map}$
$e_{i,j}$
j
i
$m_{i,x}$
x
Library $\mathcal{L}$
Library $\mathcal{L}$

Fig. 2 Graph abstractions for the system architecture. Each color represents a node type; circles and squares represent components and implementations, respectively.

We frame the exploration problem by introducing binary variables $e_{i,j} \in E_\mathcal{T}$ to denote the presence or absence of a connection between components v_i and v_j in $\mathcal{T}$. Implementations in $\mathcal{L}$ can also be represented as nodes that are connected to nodes in $\mathcal{T}$ by mapping edges. We denote a *mapping edge* as a binary variable $m_{i,x}$ which indicates the presence or absence of a *mapping* from a node $v_i \in \mathcal{T}$ with type k_i to its implementation $v_x \in \mathcal{L}_{k_i}$, $x \in \{1, ..., |\mathcal{L}_{k_i}|\}$ [7], [10]. The *mapping template* $\mathcal{T}_{map} = (V_{\mathcal{T}_{map}}, E_{\mathcal{T}_{map}})$ augments $V_\mathcal{T}$ with all the implementations in $\mathcal{L}$ and $E_{\mathcal{T}_{map}}$ with all the possible mappings from the nodes in $\mathcal{T}$ to the implementations in $\mathcal{L}$. As illustrated in Fig. 2, an assignment over all the edges $e_{i,j}$ of $\mathcal{T}$ defines the selected template $\mathcal{A}$; an assignment over all the edges $e_{i,j}$ and $m_{i,x}$ in $\mathcal{T}_{map}$ defines the selected mapping $\mathcal{A}_{map}$. Finally, we formalize the system requirements in terms of: (i) a set of system-level contracts $\mathbf{C}_s := \{C_s^d | d \in \mathbf{d}\}$ from a viewpoint set $\mathbf{d}$; and (ii) sets of component-level contracts $\mathbf{C}^d := \{C_i^d | v_i \in V_\mathcal{T}\}$ for each d in $\mathbf{d}$. We further express the sets A and G of the contracts with the corresponding constraints ϕ_A and ϕ_G.

Problem 1 (Optimized Architecture Selection). *Given a template $\mathcal{T}$, the library $\mathcal{L}$, the system-level contracts $\mathbf{C}_s$, the component-level contracts $\mathbf{C}^d$ for each d from a viewpoint set $\mathbf{d}$, and a cost function $c : \mathbb{R}^{|E_{\mathcal{T}_{map}}|} \rightarrow \mathbb{R}$, select an architecture that solves the following problem:*

$$\min_{e_{i,j}, m_{i,x}} \quad c(E_{\mathcal{T}_{map}}) \text{ s.t.} \quad \bigwedge_{v_i \in V_\mathcal{T}, d \in \mathbf{d}} (\phi_{A_i^d} \wedge \phi_{G_i^d}),$$
$$\bigotimes_{v_i \in V_\mathcal{T}} C_i^d \preceq C_s^d, \ \forall d \in \mathbf{d}.$$

The solution to Problem 1 will be derived by solving three sub-problems (Problems 2, 3, and 4), described in the following sections.

III. Contract-Based Modeling and Formalization

We use A/G contracts to model and formalize both component and system requirements and constraints.

A. Architecture Modeling and Interconnection Constraints

We can capture interconnection and mapping constraints [8], [10] for architecture components in terms of contracts as follows. Consider, for example, a node v_i selected from a partition Π_{k_i} of an architecture. Let its predecessors be in Π_{k_i-1}, its successors in Π_{k_i+1}, and its potential implementations in $\mathcal{L}_{k_i}$. The assumptions for v_i state that v_i can only be mapped to one implementation x if it has at least one connection with other components. Under these assumptions, the guarantees for v_i ensure that the attributes are also mapped and the following constraints hold: (i) there are at most M and N connections to nodes in Π_{k_i-1} and Π_{k_i+1}, respectively; (ii) if there is at least one connection with a node in Π_{k_i-1}, then there must be at least one connection with a node in Π_{k_i+1} and *vice versa*. We then obtain the following *interconnection contract* C_i^C for v_i:

$$\begin{aligned}\phi_{A_i^C} :=& ((\sum_{v_a\in\Pi_{k_i-1}} e_{a,i} + \sum_{v_b\in\Pi_{k_i+1}} e_{i,b} > 0) \rightarrow \sum_{x\in\mathcal{L}_{k_i}} m_{i,x} = 1)\\ &\wedge ((\sum_{v_a\in\Pi_{k_i-1}} e_{a,i} + \sum_{v_b\in\Pi_{k_i+1}} e_{i,b} = 0) \rightarrow \sum_{x\in\mathcal{L}_{k_i}} m_{i,x} = 0),\\ \phi_{G_i^C} :=& \bigwedge_{j\in\mathbf{u}_i} (u_{j,i} = \sum_{x\in\mathcal{L}_{k_i}} m_{i,x} U_{j,x}) \wedge \sum_{v_a\in\Pi_{k_i-1}} e_{a,i} \leq M\\ &\wedge ((\sum_{v_a\in\Pi_{k_i-1}} e_{a,i} > 0) \rightarrow \sum_{v_b\in\Pi_{k_i+1}} e_{i,b} > 0) \wedge \sum_{v_b\in\Pi_{k_i+1}} e_{i,b} \leq N\\ &\wedge ((\sum_{v_a\in\Pi_{k_i-1}} e_{a,i} = 0) \rightarrow \sum_{v_b\in\Pi_{k_i+1}} e_{i,b} = 0),\end{aligned}$$

where $\mathbf{u}_i$ is the set of attributes for v_i, and $u_{j,i}$ and $U_{j,x}$ are the values of attribute j for v_i and its implementation x, respectively. The variable $u_{j,i}$ takes the value of $U_{j,x}$ when $m_{i,x}$ evaluates to one and it is set to zero if v_i is not instantiated, i.e., mapped to an implementation.

B. Flow Requirements

Various system requirements deals with the flow across the network of certain elements, e.g., power, products, or messages, that are delivered from the source to the sink nodes [8], [10]. Each component can either generate or consume flow, and the throughput sets a limit on the maximum flow that can enter the component. We can express flow requirements for v_i with a *flow contract* C_i^F of the form

$$\begin{aligned}\phi_{A_i^F} :=& f_i^P \geq \sum_{v_a\in\Pi_{k_i-1}} e_{a,i} f_{a,i} \geq f_i^C,\\ \phi_{G_i^F} :=& \sum_{v_a\in\Pi_{k_i-1}} e_{a,i} f_{a,i} + f_i^S \geq \sum_{v_b\in\Pi_{k_i+1}} e_{i,b} f_{i,b} + f_i^C,\end{aligned}$$

where $f_{a,i}$ denotes the input flow from a component v_a, $f_{i,b}$ denotes the output flow to a component v_b, and f_i^P, f_i^S, f_i^C are the throughput, the flow generated by v_i, and the flow consumed by v_i, respectively. In this contract, if the input flow remains below the throughput, the sum of the input and generated flows must at least equal the sum of the output and consumed flows.

At the system level, we expect that the overall flow through the system and the total consumption are bounded by F_s^S and F_s^C, respectively. Assuming that the source and sink nodes of the system are in partition Π_1 and Π_n, respectively, we obtain the following contract C_s^F:

$$\begin{aligned}\phi_{A_s^F} :=& \sum_{v_a\in\Pi_1} e_{a,s} f_{a,s} \leq F_s^S,\\ \phi_{G_s^F} :=& \sum_{v_a\in\Pi_1} e_{a,s} f_{a,s} - \sum_{v_b\in\Pi_n} e_{s,b} f_{s,b} \leq F_s^C,\end{aligned}$$

where $e_{a,s}$ and $e_{s,b}$ represent connections to a source node v_a and a sink node v_b, and $f_{a,s}$ and $f_{s,b}$ are the corresponding input (from v_a) and output (to v_b) flows.

C. Timing Requirements

Timing requirements define the time by which a flow must reach a certain component. We specify the maximum allowable *latency* for a component to execute a function and the *jitter*, modeling variability in the latency. A *timing contract* C_i^T for component v_i can then be defined as follows:

$$\begin{aligned}\phi_{A_i^T} :=& \bigwedge_{v_a\in\Pi_{k_i-1}} (e_{a,i} \rightarrow |t_{a,i} - \tau_{a,i}| \leq j_i^I),\\ \phi_{G_i^T} :=& \bigwedge_{v_b\in\Pi_{k_i+1}} (e_{i,b} \rightarrow |t_{i,b} - \tau_{i,b}| \leq j_i^O)\\ &\wedge \bigwedge_{v_a,v_b} (e_{a,i} \wedge e_{i,b} \rightarrow \tau_{i,b} - t_{a,i} \leq l_i).\end{aligned}$$

For edge $e_{i,j}$, $\tau_{i,j}$ is the nominal time of occurrence of an event while $t_{i,j}$ is the actual time due to the jitter. Assuming that the input jitter does not exceed a maximum limit j_i^I, if a component is connected, its operation time and output jitter must be bounded by l_i and j_i^O, respectively.

Analogously, we can express system-level timing requirements from sources to sinks with a contract C_s^T of the form

$$\begin{aligned}\phi_{A_s^T} :=& \bigwedge_{v_a\in\Pi_1} (e_{a,s} \rightarrow |t_{a,s} - \tau_{a,s}| \leq J_s^I),\\ \phi_{G_s^T} :=& \bigwedge_{v_b\in\Pi_n} (e_{s,b} \rightarrow |t_{s,b} - \tau_{s,b}| \leq J_s^O)\\ &\wedge \bigwedge_{v_a,v_b} (\tau_{s,b} - t_{a,s} \leq L_s^{a,b}),\end{aligned}$$

where $L_s^{a,b}$ is the maximum latency between source v_a and sink v_b, $\tau_{a,s}$ and $t_{a,s}$ are the nominal and actual flow generation times, $\tau_{s,b}$ and $t_{s,b}$ are the nominal and actual consumption times, and J_s^I and J_s^O are bounds on the source and sink jitters.

IV. Contract-Based Architecture Exploration

This section details the proposed optimization scheme.

A. MILP-Based Architecture Selection

We start by defining the architecture exploration problem under component-level contracts.

Problem 2 (Candidate Architecture Selection)**.** *Given the component-level contracts* $\mathbf{C}^d$ *for each* d *in* $\mathbf{d}$, *select the lowest cost candidate architecture* $\mathcal{A}_{map}$ *such that all the contracts in* $\bigcup_{d\in\mathbf{d}} \mathbf{C}^d$ *are satisfied.*

Problem 2 is encoded to the following MILP problem providing an optimal system architecture that guarantees that all the viewpoint contracts are also consistent and compatible:

$$\min_{e_{i,j},m_{i,x}} \sum_{i=1}^{|V_{\mathcal{T}}|} \alpha_i \beta_i c_i \ \text{s.t.} \bigwedge_{v_i\in V_{\mathcal{T}},d\in\mathbf{d}} (\phi_{A_i^d} \wedge \phi_{G_i^d}) \wedge \phi_{\mathfrak{c}}, \tag{1}$$

where $\phi_{A_i^d}$ and $\phi_{G_i^d}$ are the contract assumptions and guarantees for all viewpoints $d \in \mathbf{d}$ of component v_i and $\phi_{\mathfrak{c}}$ is the conjunction of constraints in set $\mathfrak{c}$ generated from the infeasibility certificates in Problem 4, which initially evaluates to

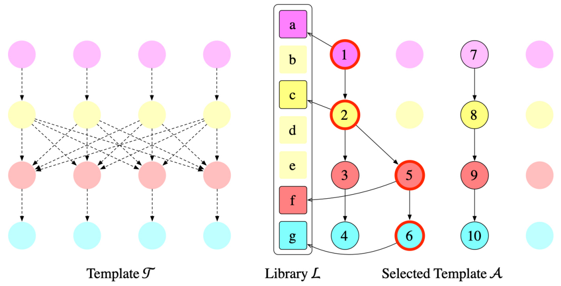


Fig. 3 Example of candidate architecture from a template.

true (i.e., no constraints). We assume an additive cost function $c = \sum_{i=1}^{|V_{\mathcal{T}}|} \alpha_i \beta_i c_i$, where c_i is the cost associated with v_i after it is mapped to an implementation, α_i is a user-defined weight, β_i is a binary variable that evaluates to one if and only if v_i is instantiated. The solution to Problem 2 is an assignment to all the variables $e_{i,j}$ and $m_{i,x}$, which provides the selected mapping $\mathcal{A}_{map}$ as a candidate architecture. We include logical operations in the above constraints for convenience. They can all be translated into linear arithmetic expressions using standard MILP encoding techniques [11].

B. Contract Refinement Checking

We cast a refinement problem to verify whether the candidate architecture satisfies the system-level requirements.

Problem 3 (Contract Refinement Verification)**.** *Given the selected mapping $\mathcal{A}_{map}$, the system-level contracts $\mathbf{C}_s$, the component-level contracts $\mathbf{C}^d$ for each viewpoint d in $\mathbf{d}$, check whether the composition of the contracts in $\mathbf{C}^d$ refines C_s^d for each d. If the refinement fails for a viewpoint, return an invalid architecture $\mathcal{G}_{map}$.*

Given the composition $C_c^d = \bigotimes_{v_i \in V_{\mathcal{A}}} C_i^d$ and the system contract C_s^d for viewpoint d, we check that $C_c^d \preceq C_s^d$ by solving a satisfiability problem. Refinement holds if and only if $\phi_{A_c^d} \wedge \neg\phi_{A_s^d}$ and $\phi_{G_s^d} \wedge \neg\phi_{G_c^d}$ are both unsatisfiable [6]. Based on the nature of the requirements, refinement checking can also be performed compositionally, which breaks the verification problem into smaller sub-problems. This is the case when system requirements are specified along paths, e.g., by defining bounds on path delays or power consumption constraints on certain routes. For an in invalid path, the infeasibility certificate generated in Problem 4 will also be smaller, hence more effective at reducing the search space.

Definition 3 (Path)**.** *A path $\mu(v_a, v_b)$ of a graph $\mathcal{G}$ is a sequence of nodes $\{n_{(0)}, ..., n_{(k)}\}$ such that $n_{(0)} = v_a$, $n_{(k)} = v_b$, and $e_{i,i+1} \in E$ for each $i \in \{0, ..., k-1\}$.*

Let Π_1 and Π_n be the source type and sink type partitions, respectively. We can then search for all paths $\mu(v_a, v_b)$ such that $v_a \in \Pi_1$ and $v_b \in \Pi_n$. For example, in the candidate architecture of Fig. 3, three paths can be identified from $\Pi_1 = \{v_1, v_7\}$ to $\Pi_4 = \{v_4, v_6, v_{10}\}$. For each such path $\mathcal{G}_{map}$ generated from $\mathcal{A}_{map}$, we can compute $C_p^d = \bigotimes_{v_i \in \mu(v_a,v_b)} C_i^d$ and check that $C_p^d \preceq C_s^d$.

Algorithm 1 Compositional Refinement Verification

```
Require: Selected mapping $\mathcal{A}_{map}$, Requirement viewpoints $\mathbf{d}$, $\forall d \in \mathbf{d}$: Component
    contracts $\mathbf{C}^d$, System contract $C_s^d$.
Provide: Invalid architecture $\mathcal{G}_{map}$, Violated viewpoint $d_v$.
 1: $\mathbf{d}_p, \mathbf{d}_o \leftarrow$ Identify path-specific viewpoints versus other viewpoints from $\mathbf{d}$.
 2: $\Pi_1, \Pi_n \leftarrow$ Identify sources and sinks in $\mathcal{A}_{map}$;
 3: $\mathfrak{p} \leftarrow$ Search paths from $\Pi_1$ to $\Pi_n$;
 4: for $d$ in $\mathbf{d}_p$ do
 5:     for $\mu$ in $\mathfrak{p}$ do
 6:         $C_p^d \leftarrow \bigotimes_{v_i \in \mu} C_i^d$;
 7:         if not Refine($C_p^d, C_s^d$) then
 8:             $\mathcal{G}_{map} \leftarrow$ Generate architecture from $\mu$ in $\mathcal{A}_{map}$;
 9:             return $\mathcal{G}_{map}, d$
10: for $d$ in $\mathbf{d}_o$ do
11:     $C_c^d \leftarrow \bigotimes_{v_i \in V_{\mathcal{A}}} C_i^d$;
12:     if not Refine($C_c^d, C_s^d$) then
13:         return $\mathcal{A}_{map}, d$
14: return $\emptyset$
```

Algorithm 1 details the procedure. We first isolate the path-specific viewpoint set $\mathbf{d}_p$ from the other viewpoint set $\mathbf{d}_o$ (line 1). We then search for all paths from the source partition Π_1 to the sink partition Π_n (line 2–3). For each viewpoint $d \in \mathbf{d}_p$, we verify whether $C_p^d \preceq C_s^d$ along each path. If the refinement is infeasible, we return the invalid path architecture $\mathcal{G}_{map}$ and d (line 4–9). Otherwise, we verify refinement for each of the other viewpoints d on the entire architecture candidate and return $\mathcal{A}_{map}$ and d if it is infeasible (line 10–13).

C. Subgraph Isomorphism-Based Certificate Generation

We use subgraph isomorphism to prune infeasible architectures. A graph $G' = (V', E')$ is defined as a subgraph of another graph $G = (V, E)$ if and only if $V' \subseteq V$ and $E' \subseteq E$. The problem of subgraph isomorphism is to extract all *subgraph isomorphic embeddings* of G' in G [12].

Definition 4 (Graph Isomorphism)**.** *Given graphs $G = (V_1, E_1)$ and $H = (V_2, E_2)$, we say G is isomorphic to H if there exists a bijection $f : V_1 \to V_2$ such that $\{u, v\} \subseteq E_1$ if and only if $\{f(u), f(v)\} \subseteq E_2$.*

Problem 4 (Subgraph Isomorphism-Based Constraint Generation)**.** *Given the mapping template $\mathcal{T}_{map}$, the selected mapping $\mathcal{A}_{map}$, the invalid architecture $\mathcal{G}_{map}$ and the violated viewpoint d_v from refinement verification, generate a constraint set $\mathfrak{c}$ from subgraph isomorphism to exclude invalid architecture candidates.*

Intuitively, we match isomorphic embeddings of $\mathcal{G}_{map}$ in $\mathcal{T}_{map}$ and exclude them in the MILP problem to prune the search space. As detailed by Algorithm 2, if contract refinement holds (Algorithm 1 returns $\emptyset$), we return the architecture $\mathcal{M}$, generated from $\mathcal{A}_{map}$, as the optimal solution of Problem 2. Otherwise, we first disconnect the implementation nodes from $\mathcal{T}_{map}$ and $\mathcal{G}_{map}$ (line 4) and match all the subgraph isomorphic embeddings of $\mathcal{G}$ in $\mathcal{T}$ (line 5). We then search the implementation nodes that violate the system requirement from $\mathcal{L}$ (line 7–8) by comparing the implementation attributes related to the violated viewpoint d_v with those in the original invalid candidate. For example, if the latency requirement fails and less latency is expected in the system, implementations with longer latency will also result in an invalid architecture. We collect in $\mathcal{L}_g^+$ the implementation nodes violating the system requirements. For each isomorphic embedding $\mathcal{G}'$ in $\mathbf{G}$, if $\mathcal{G}'$ is a path architecture, we ensure that $\mathcal{G}'$ is not selected together with the implementations in $\mathcal{L}_g^+$ (line 12). Otherwise, we allow $\mathcal{G}'$ to still be selected with implementations from $\mathcal{L}_g^+$ to satisfy the system-level

Algorithm 2 Generation of Subgraph Isomorphism-Based Constraints.

```
Require: Mapping template $\mathcal{T}_{map}$, Selected mapping $\mathcal{A}_{map}$, Invalid architecture
    $\mathcal{G}_{map}$, Violated viewpoint $d_v$.
Provide: Set of constraints $\mathfrak{c}$ or the selected architecture $\mathcal{M}$.
 1: if $\mathcal{G}_{map} = \emptyset$ then
 2:     $\mathcal{M} \leftarrow$ Generate architecture from $\mathcal{A}_{map}$;
 3:     return $\mathcal{M}$;
 4: $\mathcal{T}, \mathcal{G} \leftarrow$ Detach implementation nodes from $\mathcal{T}_{map}$ and $\mathcal{G}_{map}$;
 5: $\mathbf{G} \leftarrow \mathsf{SubgraphIsomorphism}(\mathcal{T}, \mathcal{G})$;
 6: $\mathfrak{c} \leftarrow \emptyset$;
 7: $\mathcal{L}_g \leftarrow$ Get selected implementation nodes from $\mathcal{G}_{map}$;
 8: $\mathcal{L}_g^+ \leftarrow \mathsf{ImplementationSearch}(\mathcal{L}_g, \mathcal{L}, d_v)$;
 9: for $\mathcal{G}'$ in $\mathbf{G}$ do
10:     $\mathbf{m} \leftarrow$ Get mapping variables from nodes of $\mathcal{G}'$ to $\mathcal{L}_g^+$;
11:     if $\mathcal{G}_{map} \neq \mathcal{A}_{map}$ then
12:         $\mathfrak{c} \leftarrow \mathfrak{c} \cup \{\sum_{v \in E_{\mathcal{G}'} \cup \mathbf{m}} v < |E_{\mathcal{G}'}| + |V_{\mathcal{G}'}|\}$;
13:     else
14:         $\mathbf{e} \leftarrow$ Get unselected edges in $\mathcal{T}$ that enter or exit from the subgraph $\mathcal{G}'$;
15:         $\mathfrak{c} \leftarrow \mathfrak{c} \cup \{\sum_{v \in E_{\mathcal{G}'} \cup \mathbf{e}} v > |E_{\mathcal{G}'}| \vee \sum_{v \in E_{\mathcal{G}'} \cup \mathbf{m}} v < |E_{\mathcal{G}'}| + |V_{\mathcal{G}'}|\}$;
16: return $\mathfrak{c}$.
```

requirements of d_v whenever other edges connecting to $\mathcal{G}'$ exist in $\mathcal{T}$ (line 14–15).

As an example, consider the architecture in Fig. 3 and assume that $\mathcal{G}_{map}$ generated from path $\mu(v_1, v_6)$ with its mappings to $\mathcal{L}_g = \{v_a, v_c, v_f, v_g\}$ is invalid since it violates a latency requirement. Assuming that the latency for each implementation of v_2 (in yellow) satisfies $L_b \geq L_c \geq L_d \geq L_e$, we obtain $\mathcal{L}_g^+ = \{v_a, v_b, v_c, v_f, v_g\}$. Paths $\mu(v_1, v_4)$ and $\mu(v_7, v_{10})$ are isomorphic to $\mu(v_1, v_6)$, which generates three constraints to prune the three architectures. For $\mu(v_7, v_{10})$ and corresponding $\mathcal{G}'$, the constraint can be expressed as $(e_{7,8} + e_{8,9} + e_{9,10}) + (m_{a,7} + m_{b,8} + m_{c,8} + m_{f,9} + m_{g,10}) \leq |E_{\mathcal{G}'}| + |V_{\mathcal{G}'}| = 7$, meaning that path $\mu(v_7, v_{10})$ and implementations from $\mathcal{L}_g^+$ cannot be selected together at the next iteration.

To summarize, solving Problems 2 (architecture selection for component-level contracts), 3 (refinement checking for system-level contracts), and 4 (search space pruning) iteratively is equivalent to solving Problem 1. In fact, the constraints generated in Problem 4 at each iteration will only exclude architectures that violate the system-level contracts and the actual feasible space for the exploration problem remains unchanged. Finally, can also utilize contract-based decompositions to enhance efficiency by breaking the system into subsystems and casting separate optimization problems. We illustrate this approach in Section V-A.

V. Evaluation Results

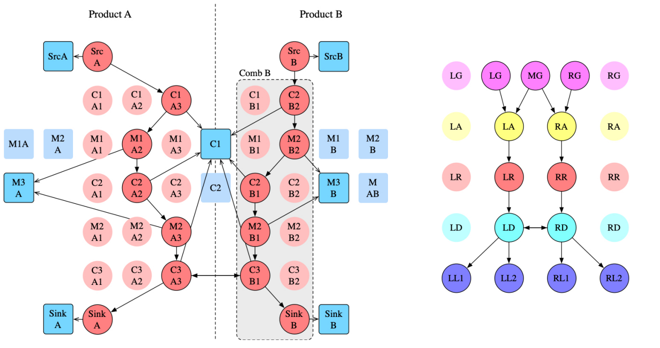
We implemented using the Python interface provided by CHASE [6] for formalization and modeling. We interfaced our libraries with Gurobi [13], to solve the MILP problems and check refinements, and with DotMotif [14], to identify isomorphic subgraphs.

A. Reconfigurable Production Line (RPL)

We applied to the design of a reconfigurable production line (RPL) system. An RPL includes a source (*Src*) that provides elements of a product, machines (M) that process these elements, and a sink ($Sink$). Machines are connected by conveyors (C). Each component in the system is labeled with a cost c and a subtype s. Nodes Src and M are characterized by a flow rate f^S and throughput f^P, respectively. We assemble products A and B on the left and right production lines, respectively. Each line has two machines and three conveyors along the path. The provided components and implementations are shown in Table I, where n_A and n_B are the problem parameters. Figure 4(a) depicts the selected mapping for a scenario with $n_A = 3$ and $n_B = 2$ where the red nodes are available components in $\mathcal{T}$ and the blue nodes are implementations in $\mathcal{L}$.

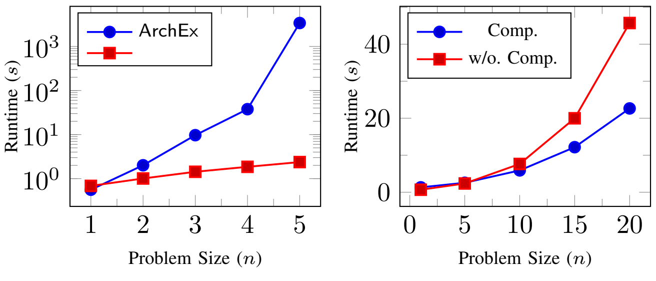
TABLE I Template and library for the RPL example.

Type	Max # in $\mathcal{T}$ (A,B)	Cost $\times 10^3$	f^S, f^P (elements/min)		
			A	B	AB
Source	1,1	0	12	10	-
Machine	n_A,n_B	{2, 3, ..., 15}	3, 6, 20	3, 5, 13	10
Conveyor	n_A,n_B	0.5, 1	-	-	-
Sink	1,1	0	0	0	-



(a) RPL selected mapping (b) EPN selected template

Fig. 4 Architectures for the RPL and EPN case studies.



(a) Comparison with ArchEx [7]. (b) Impact of compositionality.

Fig. 5 Runtime as a function of the problem size.

Figure 5(a) shows the results obtained by with respect to ArchEx [7], a comparable system architecture exploration tool based on effective MILP formulations. The horizontal axis represents the maximum number of machines and conveyors ($n_A = n_B = n$) in $\mathcal{T}$ while the vertical axis shows the runtime of the exploration process. outperformed ArchEx in every experiment, while finding optimal architectures with the same cost. The runtime gap increases with the problem size.

We further evaluated the impact of compositional exploration provided by . We partitioned the system and synthesized each subsystem independently. We aggregated the production line components of the line for product B to create *Comb B*, connected to the conveyors in line A as shown in Figure 4(a). Assuming a maximum throughput f^P for *Comb B*, we first synthesized the architecture of line A. Then, we synthesized the architecture for line B, assuming the selected architecture for line A. To check the compatibility between the selected production lines A and B, it was sufficient to verify that the composition of component-level contracts from line B for each viewpoint refines the system-level contract defined

TABLE II Evaluation of the effectiveness of different solution methods on the EPN example with various templates.

Max # in $\mathcal{T}$	# of	# of	Only subgraph isomorphism		Only decomposition		Complete	
(L,R,APU)	variables	constraints	Time (s)	# of iterations	Time (s)	# of iterations	Time (s)	# of iterations
1,0,0	454	195	0.57	3	0.58	3	0.56	3
2,0,0	1178	592	4.78	8	10.53	28	2.50	4
3,0,0	2280	1281	50.21	12	84.77	104	8.52	6
4,0,0	3868	2352	6.31×10^3	18	4.45×10^3	231	20.55	4
1,1,0	1138	576	11.18	22	10.72	24	9.15	24
2,1,0	2374	1383	4.09×10^3	93	4.82×10^2	320	27.12	20
2,2,0	4004	2508	2.73×10^4	152	5.59×10^3	1581	1.55×10^2	34
1,1,1	1294	666	62.79	85	13.89	30	16.26	31
2,1,1	2604	1532	1.57×10^2	56	1.99×10^2	168	40.94	26
2,2,1	4320	2726	2.35×10^3	60	3.87×10^3	1353	1.06×10^2	23
Average			4.04×10^3	50.9	1.07×10^3	384.1	38.67	17.5
Ratio			104.36	2.91	27.68	21.95	**1.00**	**1.00**

for *Comb B*. The runtime with and without compositional exploration is shown in Fig. 5(b). Compositional exploration becomes more effective as the problem size increases.

B. Aircraft Electrical Power Distribution Network (EPN)

We applied to the exploration of an aircraft electric power distribution network (EPN). An EPN delivers power from generators (GEN) to a set of loads via AC and DC buses. Rectifier units (RU) are used to convert AC power to DC power. A selected EPN template is shown in Figure 4(b). Components are grouped based on their locations, i.e., on the left (L) or right (R) side, with the following types: generators ($LG/RG/MG$), where MG is used for auxiliary power units (APU), AC buses (LB/RB), RUs (LR/RR), DC buses (LD/RD), and loads (LL/RL). APUs can be connected to both the left and right sides.

We aim to generate an EPN architecture that satisfies a set of interconnection, power, and timing requirements while minimizing the overall cost. We explore EPN systems with an increasing number of components in the template, as detailed in Table II. The "Max # in $\mathcal{T}$" column indicates the number of components on both the left and right side for each component type, along with the number of APUs. For each type of node, four implementations are provided in the library. The "# of variables" and "# of constraints" columns report the size of the MILP problem. We evaluated three scenarios: (i) "only subgraph isomorphism" applies only the subgraph isomorphism technique without any further decomposition strategy; (ii) "only decomposition" involves only contract decompositions for refinement checking while disabling the subgraph isomorphism certificates; and (iii) "Complete " enables both methods.

As shown in Table II, the decomposition of the system architecture reduced the complexity of the refinement checking procedure, resulting in decreased runtime, while subgraph isomorphism was capable of excluding all isomorphic embeddings in a single iteration, thus taking fewer iterations to find a feasible architecture. achieved up to two orders of magnitude acceleration, on average, compared to the "only subgraph isomorphism" case, and it required 20 times fewer iterations compared to the "only decomposition" case. When a template consists of two sides, it tends to generate a larger search space, which explains the difference in runtime between templates with the same number of components, as in template "3,0,0" versus "2,1,0". Moreover, introducing an APU in the template as an additional power source in the middle makes it easier to satisfy the power supply requirements, which accounts for the improvement observed for template "2, 1, 1" versus template "2, 1, 0".

VI. Conclusion

We presented , an efficient methodology for system architecture exploration based on A/G contracts. enables modeling and exploration of system architectures compositionally as well as methods using subgraph isomorphism to iteratively prune infeasible architectures out of the search space. We demonstrated its effectiveness on two case studies. Future work includes extensions to incorporate contracts in different formalisms and other graph-based algorithms to enhance efficiency.

References

[1] E. A. Lee, "Cyber physical systems: Design challenges," in *Proc. IEEE Int. Symp. Object Orient. Real Time Distrib. Comput.*, May 2008.

[2] J. Sztipanovits, X. Koutsoukos, G. Karsai, N. Kottenstette *et al.*, "Toward a science of cyber–physical system integration," *Proc. IEEE*, vol. 100, 2011.

[3] S. A. Seshia, S. Hu, W. Li *et al.*, "Design automation of cyber-physical systems: Challenges, advances, and opportunities," *IEEE Trans. Comput.-Aided Design of Integr. Circuits and Syst.*, vol. 36, 2016.

[4] A. Benveniste, B. Caillaud, D. Nickovic *et al.*, "Contracts for system design," *Foundations and Trends in Electronic Design Automation*, vol. 12, 2018.

[5] P. Nuzzo, A. L. Sangiovanni-Vincentelli *et al.*, "A platform-based design methodology with contracts and related tools for the design of cyber-physical systems," *Proc. IEEE*, vol. 103, 2015.

[6] P. Nuzzo, M. Lora, Y. A. Feldman *et al.*, "CHASE: Contract-based requirement engineering for cyber-physical system design," in *Proc. Design Autom. Test Europe*, 2018.

[7] D. Kirov, P. Nuzzo, R. Passerone, and A. Sangiovanni-Vincentelli, "ArchEx: An extensible framework for the exploration of cyber-physical system architectures," in *Proc. Design Autom. Conf.*, 2017.

[8] P. Nuzzo, N. Bajaj, M. Masin *et al.*, "Optimized selection of reliable and cost-effective safety-critical system architectures," *IEEE Trans. Comput.-Aided Design of Integr. Circuits and Syst.*, 2019.

[9] D. Kirov, P. Nuzzo, R. Passerone *et al.*, "Optimized selection of wireless network topologies and components via efficient pruning of feasible paths," in *Proc. Design Autom. Conf.*, 2018.

[10] D. Kirov, P. Nuzzo, A. Sangiovanni-Vincentelli *et al.*, "Efficient encodings for scalable exploration of cyber-physical system architectures," *IEEE Trans. Comput.-Aided Design of Integr. Circuits and Syst.*, 2023.

[11] W. L. Winston, *Operations research: applications and algorithms*. Cengage Learning, 2022.

[12] F. Bi, L. Chang, X. Lin *et al.*, "Efficient subgraph matching by postponing cartesian products," in *Proc. Int. Conf. Management of Data*, 2016.

[13] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com

[14] J. K. Matelsky, E. P. Reilly, E. C. Johnson *et al.*, "DotMotif: an open-source tool for connectome subgraph isomorphism search and graph queries," *Scientific Reports*, vol. 11, 2021.