# Optimization-Based Methodology for the Exploration of Cyber-Physical System Architectures

Doctoral Dissertation

**Candidate** Dmitrii Kirov, University of Trento, Italy

**Advisor** Prof. Roberto Passerone, University of Trento, Italy

**Committee** Prof. Luciano Lavagno, Politecnico di Torino, Italy
Prof. Pierluigi Nuzzo, University of Southern California, USA
Prof. Luigi Palopoli, University of Trento, Italy

Trento, April 2018

*The shortest distance between two points*
*is often unbearable.*
Charles Bukowski

# Acknowledgements

According to my personal statistics, not many students are given much freedom in their research without being completely abandoned at the same time. Likewise, few students have this rare possibility of randomly dropping by their advisor's office at almost any time to ask their questions (often minor or stupid or both). During this PhD adventure, I was lucky to be among these "few" and "not many". For this, I feel very thankful to **Roberto Passerone**. For this, as well as for guiding me through these 4.5 academic years, some of them more and some of them less lucky. For his support, especially during these last months of the PhD, and for offering me future opportunities. Finally, for providing me a great internship possibility, which turned my world upside down, in a sense, and ended up with the work that you, the reader, probably plan to comprehend (or scroll) after looking at this first page. Don't you?

I deeply acknowledge **Pierluigi Nuzzo**, an extraordinary person and a great mentor, for his inherent contribution to this work, and for our fruitful collaboration - current, past and, hopefully, future. I learned a lot from Pierluigi during these two years after we first met in Berkeley in 2016, and our weekly Skype calls, which almost never stopped since then, are still one of the most valuable things for me.

I would like to thank **Luciano Lavagno** and Pierluigi for reviewing this thesis and providing me their feedback. I am glad that it was an interesting read.

Listing friends in this section is perhaps the thing I do not like. I am lucky to have quite many of them. But what if I forget someone? So, instead, I would like to acknowledge one very special place in Trento, which is called **Zorba's**. A place, where most, if not all, of the friends that I wish to thank have been at least once. Sharing drinks and talks, playing table football, talking to the owner lady with all her crazy stories, or listening to me playing guitar and singing (yes, once I did even that). A place, which shared with me some happiest and some saddest moments, some best memories and few very important decisions.

Maybe you thought that I will avoid the canonical ending part of this text, but here I go. I thank my **parents** for their constant support in every possible sense. For getting worried with and, mostly, without a reason. And **Lena**, I dedicate this thing to you and thank you for your love and support, and for the path that we walk together. I am sure you would prefer to have another kind of dedication, like a song or a novel. In fact, I am working on that, even though at a slower pace with respect to research. But who said that the acknowledgements cannot have a future work part?

*Trento, 26 April 2018* D. K.

# Abstract

Embedded electronics becomes pervasive, and its tight integration with modern mechanical, thermal, chemical, biological, and other systems shows promise of making the resulting cyber-physical systems (CPS) more capable and efficient, outperforming their predecessors. Existing engineering practices, however, often fall short of coping with the complexity of CPS design. One of the dominant reasons is the inability of foreseeing the impact of the decisions made in the early stages of the design process, i.e., in the concept design phase, on the final implementation. A major bottleneck is the lack of abstractions and formalisms that are able to capture heterogeneous system requirements and enable efficient co-design. Therefore, methodologies and tools that provide such abstractions and generate high-level cyber-physical system architectures with correctness guarantees become highly desirable.

This dissertation seeks to advance the state of the art in cyber-physical system design by proposing a novel exploration methodology for system architectures at a high level of abstraction. We leverage optimization techniques to select correct-by-construction configuration and interconnection of components taken from predefined libraries, while meeting a set of system requirements and minimizing a cost function. Using a graph-based representation of the system architecture, we identify a generic basis of the exploration problem as a common semantic domain that includes a set of decision variables and mixed integer linear constraints over graph vertices, edges and paths. On top of this basis, we are able to instantiate a variety of CPS design requirements, such as interconnection, flow balance, timing, reliability, workload, routing and energy. Our resulting mathematical formulation includes the topology selection problem, i.e., whether a "virtual" component should be used in the final configuration and how it is connected to other components, and the mapping problem, i.e., which "real" library component best implements the "virtual" one.

To foster the scalability of our approach, we propose a set of algorithms for efficiently formulating and solving exploration problems. In particular, we automatically generate compact, yet approximate, encodings of paths in the architecture graph, which restrict the search to a limited subset of the most promising solutions. This leads to savings of orders of magnitude in terms of problem complexity and optimization time, at a small cost in terms of optimality, making it possible to find solutions to otherwise impractical problems. Besides the monolithic optimization with respect to a set of constraints capturing all specified system requirements, we also investigate iterative optimization schemes. In the latter, the solver is called iteratively on smaller problem instances including only a subset of constraints to generate candidate configurations. Their validity is then checked against the other constraints via exact analysis methods, and in the case of violation a conflict-driven learning function is called to incrementally add new constraints to the original formulation, prune the search space and rapidly progress towards a feasible solution.

We have implemented ArchEx 2.0, an extensible framework that supports all steps of the proposed methodology. Its software structure is modular and amenable to extensibility and design reuse. To simplify the problem specification, we propose a pattern-based formal language. It allows for automatic translation of high-level requirements into appropriate mixed integer linear constraints, which provides large savings in terms of development time and guarantees the correctness of the generated specification. Provided requirement patterns have clear semantics and support a rich set of cyber-physical system properties, thus ensuring the expressiveness of the framework.

We perform an extensive numerical evaluation of the proposed methodology on a set of CPS applications of industrial relevance. In particular, we demonstrate the efficiency of ArchEx 2.0 on reliability-driven designs of aircraft electrical power distribution networks and reconfigurable manufacturing systems. We then investigate the capabilities of the approach in the wireless networking domain by synthesizing topologies and node placements for data collection and localization networks of realistic size. Our results confirm the expressiveness, extensibility, usability and scalability of the proposed techniques and tools. These important characteristics allow the methodology to effectively address the challenges of the concept design stage of cyber-physical systems.

**Key words:** design space exploration, architecture exploration, optimization, design methodology, cyber-physical systems, embedded systems, wireless sensor networks, internet of things.

# Contents

# List of Figures

# List of Tables

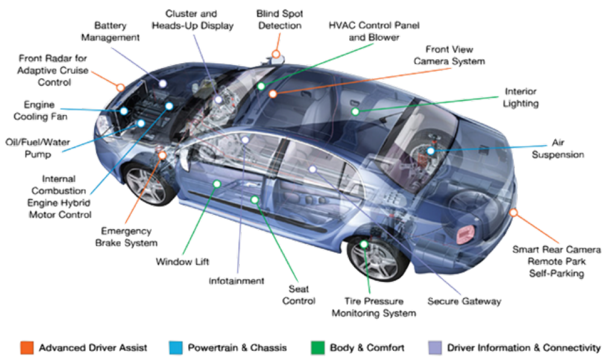
# 1 Introduction

*This chapter unveils the motivation for the work presented in this dissertation, outlines the objectives and the main results. We first identify the role of architecture exploration in a hierarchical design flow of cyber-physical systems and discuss its main challenges. We then summarize the optimization-based approach that we apply to tackle the architecture exploration problem, highlight the main contributions and put them in the context of related work. Finally, we outline the structure of the dissertation and preview the contents of subsequent chapters.*

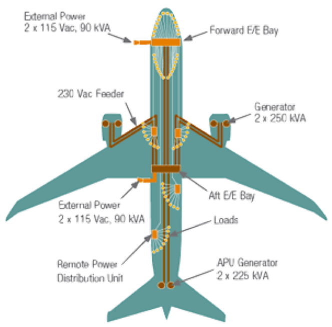
## 1.1 Cyber-Physical System Architecture

The rapid development of embedded electronics and communication networks has led to the transformation of various existing mechanical systems, such as lighting, HVAC, power distribution, building automation or car braking, to *cyber-physical systems (CPS)*. Being traditionally separated from each other, computation, communication and control are instead tightly coupled in a CPS. A large body of emerging, “smart” applications (e.g., cars, aircrafts, buildings on Figure 1.1a-c) are developed on distributed platforms with a tight integration between “cyber” (e.g., computation, networking) and “physical” (e.g., pneumatic, hydraulic) parts [37, 97]. The latter is monitored and controlled by the former using a set of sensors and feedback loops, so that physical processes affect computations and vice versa. This is an important distinction from traditional, “previous generation” electromechanical systems, which are typically characterized by unidirectional dependencies between the two domains.

The intention of consolidating previously separated cyber and physical worlds in one system gave birth to new, unforeseen challenges that brought the design and the verification of CPS to another level of complexity. In particular, existing abstractions used in embedded systems design fail to accurately capture the timing behavior of programs, which does not affect the semantic correctness of the latter, but may violate the correctness of the system [71]. Another paramount reason is the increased heterogeneity of components, because in a CPS one has to deal both with electronic (computational) and physical sides. To capture the behavior of the former, designers typically use discrete-event, dataflow, synchronous-reactive, state machine, process network and other models of computation [37]. The latter, instead, is often expressed using differential equations, i.e., continuous, dynamic models.



(a)


External Power 2 x 115 Vac, 90 kVA
Forward E/E Bay
230 Vac Feeder
Generator 2 x 250 kVA
Aft E/E Bay
External Power 2 x 115 Vac, 90 kVA
Loads
Remote Power Distribution Unit
APU Generator 2 x 225 kVA

(b) (c)

Figure 1.1 – Examples[1] of complex cyber-physical systems: (a) There are almost 100 electronic control units (ECUs) in a premium car, while millions of lines of code are required to program them, and several kilometers of wiring are needed to interconnect them; (b) Many subsystems of modern passenger aircrafts (e.g., power distribution, fuel management, environmental control) are tightly coupled with an increasing number of electronic components; (c) In a "smart" building, a large number of systems (e.g., adaptive lighting, HVAC, firefighting, security) are built on networked sense-and-control platforms.

The heterogeneous nature of cyber-physical systems has immediately revealed the lack of interoperability between existing design instrumentation (e.g., simulators, verification tools, requirement engineering tools) that captures different cyber and physical aspects of the system. Moreover, it drastically raised the need of mathematical formalisms and abstractions that could serve as a "common ground" for different system aspects - functional and non-functional, static and dynamic, electronic and physical. Consequently, methodologies and tools that can provide these abstractions for efficient synthesis, verification and performance evaluation of such heterogeneous, hybrid CPSs become highly desirable. Indeed, as the systems get more and more complex, especially with respect to stringent reliability and safety requirements, a major productivity gain in CPS design becomes essential for many industries.

[1]Sources: (a) www.linkedin.com, by Eric Walz; (b) www.boeing.com; (c) www.commscope.com.

Therefore, systems companies (e.g., avionics and automotive) are increasingly aiming at automating their design processes using the novel computer-aided design (CAD) tools for CPS, same as semiconductor industry did several decades ago to cope with complexity of VLSI design. In turn, electronic design automation (EDA) vendors today pay more attention to system design.

Another concern is the level of interdisciplinary experience required in cyber-physical system design. Today, being a *system designer* requires lots of cross-domain knowledge and simultaneous reasoning about different heterogeneous requirements. That is, embedded and physical system engineers have to go beyond their primary domains in order to understand each other, cooperate, create design specifications, run the design space exploration, synthesis, performance analysis, verification and validation of CPSs. Apart from gaining new necessary skills, this also prompts the development of novel methodologies and supporting tools for facilitating specification, solving and debugging of CPS design problems, as well as easing up the communication between different working groups and companies.

It has been argued in the recent decades that the design of complex heterogeneous systems has to be carried out in a hierarchical, *compositional* way as a sequence of refinement steps between different levels of abstraction [114, 115, 98, 96]. This dissertation focuses on high-level *architectural* aspects of CPS design. In this context, we regard the ***cyber-physical system architecture*** as a network of interconnected components that represent both electronic and physical parts of the system. We tackle the *concept design* stage that deals with the selection of these components and connections, while meeting a set of system requirements, such as timing, reliability or power consumption. This stage is essential, because being able to capture important system aspects as early as possible in the design flow allows us to provide certain correctness guarantees to subsequent steps, thus reducing the chances of errors, requirement violations and lengthy redesign cycles. At the same time, *architecture exploration* is a search over a huge space of potential configurations, which grows with the number of requirement types being captured. Therefore, it asks for finding proper abstractions and formalisms to express the requirements, meaningful partitioning of design concerns, so that exploration problems are tractable, and efficient tools to conduct the search.

## 1.2 Motivation and Objectives

Cyber-physical system design is heavily influenced by decisions made in the early stages of the design, when their impact is still hard to foresee. In particular, architecture exploration today is the domain of experienced architects, that often have to take risky decisions based solely on their heuristic evaluations and accrued knowledge. Such *ad hoc* design practices often result in severe safety violations, countless redesign iterations, missing the time-to-market and financial losses. The major bottleneck for fostering the design process is the lack of comprehensive frameworks for scalable, multi-dimensional architecture exploration under heteronegenous CPS requirements (e.g., connectivity, safety, reliability, timing, energy, workload and others). High-fidelity simulations and prototype testbeds are typically leveraged to achieve reasonable accuracy in performance and cost estimations of an architecture that has been already worked out. However, using these approaches for design exploration and synthesis can become very time-consuming and, therefore, limited in the number of evaluated configurations. Clearly, CPS design would substantially benefit from methodologies and tools for automatic generation of system architectures with correctness guarantees.

In general, one can classify architecture exploration approaches to three groups that can be distinguished by the ability of their members to tackle the following questions:

1. *Does the current architecture meet the requirements?* These approaches do not conduct the actual search, but only evaluate the performance of a given architecture. The most typical techniques are simulation, prototyping and system description languages (e.g., [100, 9]). The exploration can be performed pointwise, i.e., by evaluating a set of selected points in the design space.

2. *Is there any architecture that meets the requirements?* This group targets ***feasible*** solutions that satisfy the given constraints. It is mainly represented by symbolic constraint safisfaction techniques, further discussed in Section 1.4.

3. *Which is the best feasible architecture with respect to a cost function?* In contrast with the previous group, the ultimate goal of these methods is to identify the ***optimal*** architecture which satisfies the given requirements, while minimizing a cost function. Therefore, corresponding approaches deal with optimization problems.

The first group includes a number of widely adopted languages and instruments (e.g., [100, 119, 107, 30, 86]), that are able to capture a variety of high-level concerns, both from cyber and physical realms. However, they are commonly focused on high-accuracy performance evaluation and analysis and have limited design exploration capabilities. In contrast, the two remaining groups rather focus on design space exploration and automated synthesis. However, there are still only few tools that target feasible [101, 90] and optimized [12, 108, 40, 87] conceptual designs in the CPS domain. Being efficient in one particular application, some of these tools suggest limited extension capabilities to other classes of systems. Furthermore, very few of them focus on *usability*, i.e., on providing languages and interfaces for low-effort specification of complex design problems. This is a significant concern, since it aims at keeping the underlying formalisms transparent to the designers and allows them to focus on the problem domain.

In general, techniques from groups 2 and 3 operate on the same search space, however, optimization tools tend to be more attractive, since they intend to find the *best* architecture among feasible ones. For example, apart from determining a valid interconnection of components of an aircraft electrical power system, these techniques can minimize the total component cost and/or weight [99, 103]. Similarly, in addition to synthesizing a resilient topology for a wireless network, in an optimization problem it is possible, for instance, to maximize the lifetime of components [108].

Considering all aforementioned points together, ***the goal of this dissertation is to devise an optimization-based methodology for the exploration of cyber-physical system architectures.*** In particular, we aim to provide a representation of a CPS architecture and a general mathematical formulation that would capture different system-level requirements (e.g., connectivity, timing, energy consumption, routing, reliability) and allow designers to cast exploration problems in different application domains. We further intend to develop an *extensible* framework to incorporate the proposed formulation and algorithms, thus facilitating the use of the methodology among system designers.

Our approach is inspired by the seminal work on reliability-driven optimized architecture design in the avionics domain [99, 12]. In these works, the authors leverage the mixed integer linear programming (MILP) techniques for encoding and solving exploration problems. Our purpose in this thesis is to *generalize* the techniques and the algorithms from [99, 12] to make them applicable to *different* CPS domains. Furthermore, we aim to make existing formulations *richer* so that more complex designs are supported, without making the exploration process expensive. This requires the development of new *scalable* encoding algorithms and efficient approximation techniques.

Apart from the static properties, CPS architectures inherently possess various characteristics that have to be evaluated in their dynamics. Such characteristics can be related both to the physical part (e.g., thermal flows, mass flows) and to the cyber part (e.g., timing behaviors), as well as to the environment (e.g., signal attenuation). Dynamic properties have to be evaluated in the continuous domain, which entails using different computation models in the same design. While state-of-the-art simulators for CPS [107, 30, 47] offer the capabilities of combining versatile models of computation, the situation is different for optimization problems. Expressions of dynamic properties, that are typically nonlinear, are possible in some formulations. Yet, despite the higher accuracy of design capture, the corresponding problems may soon become exceedingly complex and, consequently, impractical. Therefore, in this work we *partition* the architectural space and study its part related to static properties of a CPS. We abstract the dynamic properties (e.g., flow rates, voltage levels, delays) by using approppriate metrics and approximations, such as steady-state and worst-case values, and intend to provide efficient algorithms for their encoding as a part of the exploration problem. Being motivated by existing composite, holistic design methodologies proposed for CPS, such as [96] or [2], we offer a complementary approach that can be applied in the early design stages, i.e., at high levels of abstraction and at first steps in the hierarchy of the design process. Further evaluation, sizing and optimization of dynamic properties can be done on one of the subsequent stages and/or levels of abstraction.

## 1.3 Main Contributions

This dissertation provides contributions to the area of design space exploration for cyber-physical systems. We focus on the concept design of CPS architectures, since the more concerns are accounted for on the high level, the less errors are made on the lower ones. As a result, we develop a methodology for high-level architecture exploration of cyber-physical systems that performs an efficient search within a space of candidate CPS architectures to find a feasible one with respect to a set of requirements while minimizing an objective function. We fortify the methodology with a supporting framework and demonstrate the efficiency and the scalability of the two on the case studies from different CPS domains. Therefore, as summarized on Figure 1.2, our contributions can be classified to three groups: Theory and Algorithms, Tools, and Applications. Each of them is discussed below.

### 1.3.1 Theory and Algorithms

We provide a mathematical formulation of the exploration problem as an optimized mapping problem, in which all components and interconnections of the system architecture are associated with (mapped to) corresponding entities (e.g., devices, wires, links) from pre-defined libraries. Similarly to the seminal works [99, 12], we employ a generic graph-based representation of an architecture as a network of components. However, we significantly extend and generalize previous works in the following directions. First, we offer a new *problem formulation* that separates the *topology selection problem*, i.e., whether a "virtual" component should be used in the architecture and how it is connected to others, from the *mapping problem*, i.e., which "real" library component best implements the "virtual" one [62]. This separation of concerns results in a mixed integer linear program encoding of the exploration problem that is more general and more efficient than the previously proposed one [99, 12].

We further introduce a *generic basis* of the exploration problem - a common semantic domain that includes a set of binary decision variables denoting the presence/absence of edges and paths between the nodes of the architecture graph, and the mapping of graph elements to library components. We then associate these variables to each other with a set of generic MILP constraints and show how they can be used to instantiate a variety of CPS requirements, such as interconnection, reliability, timing, flow, workload, energy consumption, routing and others. In other words, exploration problem formulations for different CPS domains can be built and customized atop the proposed basis. It is also used to establish the dependency between the two problems (topology selection and mapping), i.e., both of them are expressed using the same decision variables and can be jointly solved as a single optimization problem.

On the algorithmic part, we generalize the two techniques proposed in [12] for solving optimization problems to use them in different applications. The first technique is a monolithic optimization with all specified constraints (and possible approximations). The second one is an iterative scheme, in which smaller sub-problems are solved at every iteration and heuristic functions are applied for exact analysis and learning new constraints.

We further propose an algorithm for generating more compact, yet approximate, encoding of network paths between pairs of source and destination components. It relies on the Yen's K-shortest path routine [128] to propose a smaller number of candidate network paths, as opposed to their exhaustive enumeration. This restricts the search to a limited subset of most promising solutions and leads to savings of orders of magnitude in terms of problem complexity and optimization time, at a small cost in terms of optimality. Differently from the works that apply heuristic approaches for synthesizing large-scale architectures [108, 104], our algorithm allows to leverage the empirical advances of state-of-the-art MILP solvers for dealing with otherwise impractical problems.

Finally, based on the proposed separation of concerns, the generic basis of the exploration problem and the algorithms, we form up a methodology for CPS architecture exploration. The methodology allows us to jointly select system topology and its implementation, and consists of four main steps. On the first step, architecture requirements are written down using an extensible set of patterns, which significantly simplifies the specification. Also the structure of the exploration problem and the platform library are initialized. Second step deals with *encoding* the specification into a mixed integer linear program. The architecture *template*, i.e., the set of all possible topology implementations, the library of components and the association between the two are all established using the aforementioned generic basis. The formulation is then filled in with the application constraints and the cost function, all in the form of MILP expressions on top of the basis. The *solving* stage then leverages one of the algorithms mentioned above (monolithic or iterative) to generate the system architecture. After the optimization process (and also between the iterations), *analysis* techniques (e.g., reliability, timing, workload) are applied for static verification and performance evaluation. All steps of the methodology are further detailed in Chapter 3.

### 1.3.2 Tools

We introduce ArchEx 2.0 [62], an extensible framework for formulating and solving CPS architecture exploration problems with the proposed methodology. It builds on a prototype toolbox for optimized selection of reliable architectures of aircraft electrical power systems proposed in [12]. With respect to its predecessor, ArchEx 2.0 has been significantly re-engineered, and a number of novel features have been added. In particular, its software infrastructure now relies on a set of abstract classes and reusable data structures that can be customized to instantiate architecture exploration problems in different domains of CPS. Existing concepts (e.g., reliability constraints, interconnection constraints) have been generalized, moreover, many other design concerns (e.g., timing, workload, energy) are now also supported, all based on the aforementioned generic basis. The framework is also augmented with new encoding algorithms (e.g., for network path encoding) and analysis techniques.

To lower the problem formulation effort, we have implemented a pattern-based formal language for requirement specification. It encompasses an extensible set of patterns that can be used to express requirements on the architecture, which are then automatically translated into mixed integer linear constraints. This provides large savings in terms of development time and guarantees the correctness of the generated specification. Furthermore, designers can customize the framework to implement new patterns, thus supporting broader categories of cyber-physical systems.

One notable application domain of ArchEx 2.0 is wireless networking, which is becoming the primary communication infrastructure in networked CPS. In [63] we have implemented and evaluated a large extension of the framework to support topology synthesis problems for wireless networks. This extension leverages the same semantic domain (basis), in particular, by actively using network paths in the formulation, a map of network deployment area and a set of channel models to accurately determine the operating conditions of network designs.

In the scope of wireless design problems, we also contribute to the improvement of conventional channel models used in network simulators and other related tools. In [64] we have conducted a statistical characterization of the 2.4 GHz radio channel in indoor office environments in order to verify the accuracy of the log-distance [110] channel model. Based on a huge measurement campaign of collecting the signal strength data in different realistic indoor spaces and node placement scenarios, we were able to propose a set of improvements for the log-distance model (and other affiliated ones) that could improve the accuracy of signal path loss estimation. We report on the results in Chapter 6 and implement them in ArchEx 2.0. To a greater extent, the impact of these improvements can be observed in simulation tools, which is future work.

### 1.3.3 Applications

We apply our methodology and framework to several different case studies to demonstrate the expressiveness, the usability, the extensibility and the scalability of our approach. Two of them refer to the avionics domain. We first try out a simplified example of an aircraft fuel management system to demonstrate the design flow with ArchEx 2.0. We then run an extensive numerical evaluation on a case study of aircraft electrical power distribution network with stringent safety and reliability constraints, previously introduced in [99]. With respect to previous work, we are able to efficiently generate more complex and realistic architectures, while achieving a better performance and smaller complexity of the problem formulation.

The methodology is then applied to another industrial domain, factory automation. We use our approach to generate architectures for reconfigurable manufacturing systems subject to balance, workload, timing and reliability constraints. Same as for the power distribution network case study, we evaluate both monolithic and iterative techniques of solving optimization problems, discussing their advantages and drawbacks in the scope of the factory automation case study and related design requirements.

Finally, we conduct extensive experimentation in the wireless domain. We leverage ArchEx 2.0 for synthesizing topologies and node placement for data collection and localization wireless sensor networks. In particular, in these applications we study the capabilities of the proposed approximate path encoding algorithm. Our results confirm the applicability of our optimization-based methodology to large-scale network designs, which are a crucial component of today's IoT applications.

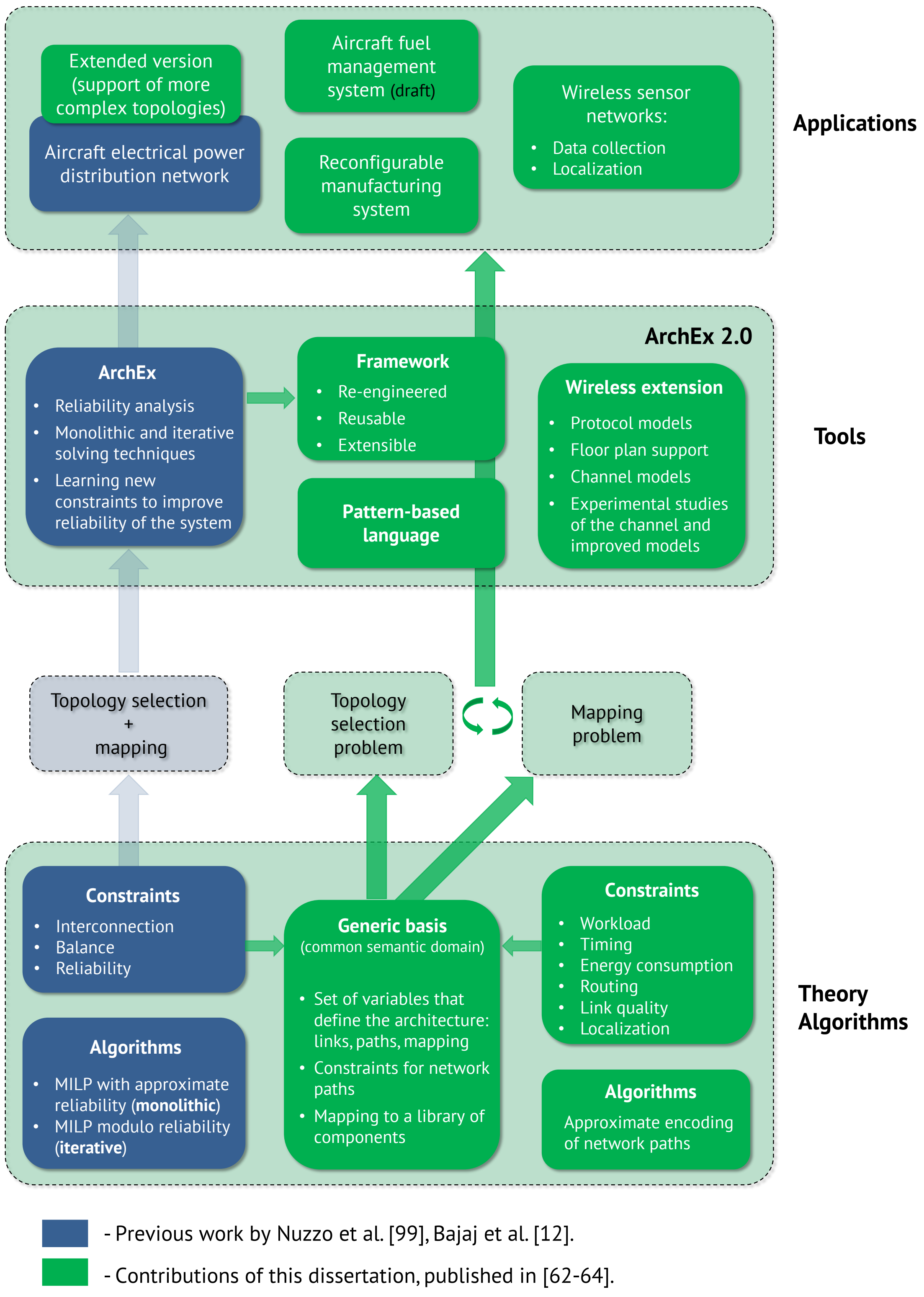

Extended version (support of more complex topologies)
Aircraft electrical power distribution network
Aircraft fuel management system (draft)
Reconfigurable manufacturing system
Wireless sensor networks:
• Data collection
• Localization
Applications
ArchEx 2.0
ArchEx
• Reliability analysis
• Monolithic and iterative solving techniques
• Learning new constraints to improve reliability of the system
Framework
• Re-engineered
• Reusable
• Extensible
Pattern-based language
Wireless extension
• Protocol models
• Floor plan support
• Channel models
• Experimental studies of the channel and improved models
Tools
Topology selection + mapping
Topology selection problem
Mapping problem
Constraints
• Interconnection
• Balance
• Reliability
Algorithms
• MILP with approximate reliability (monolithic)
• MILP modulo reliability (iterative)
Generic basis (common semantic domain)
• Set of variables that define the architecture: links, paths, mapping
• Constraints for network paths
• Mapping to a library of components
Constraints
• Workload
• Timing
• Energy consumption
• Routing
• Link quality
• Localization
Algorithms
Approximate encoding of network paths
Theory
Algorithms
- Previous work by Nuzzo et al. [99], Bajaj et al. [12].
- Contributions of this dissertation, published in [62-64].

Figure 1.2 – Summary of the contributions provided by the dissertation.

## 1.4 Related Work

The term “cyber-physical system architecture” can assume different meanings in different contexts, i.e., it can include a variety of aspects, ranging from the number, type and dimensions of components, including both the embedded system and the physical plant, to software, e.g., control algorithms. As mentioned above, in this dissertation we are taming the concept architecture design of CPS, while relying on the notion of the architecture that only involves components and their interconnections, as formally defined in Chapter 2. Therefore, this section is dedicated to the discussion of existing techniques and tools adapted to or suitable for the exploration of these high-level architectures, while some other aspects, e.g., control design, are out of scope of this work. Yet, some techniques that we touch upon are applicable to different design phases, however, we only discuss them in the context of architecture exploration.

### 1.4.1 CPS Design Methodologies

System architecture selection and evaluation is pivotal in well-known design methodologies for embedded systems [114, 16, 72], which were recently extended to CPS design [98, 59, 2]. Indeed, in platform-based design [114], the set of all architectures that can be built out of a library (collection) of components forms a design *platform*, while the exploration is carried out as a mapping of functional requirements onto available implementations at each level of abstraction. In turn, contract-based design [16, 115] provides a formal framework of assume-guarantee contracts that is used for verifying the compatibility of components in an architecture as well as the consistency of the specification. Actor-oriented design [72] advocates for co-simulation of different aspects of the architecture and the control algorithm defined using a set of computation models, at different abstraction layers.

More recently, a compositional design methodology for CPS has been proposed [98, 96], based on platform- and contract-based paradigms. In this approach, requirements for system architecture and control protocol are partitioned according to their contracts $\mathcal{C}_T$ and $\mathcal{C}_C$, and then separately designed, while it has been shown in [99] that if both $\mathcal{C}_T$ are $\mathcal{C}_C$ are satisfied, then the assembled system is guaranteed to satisfy the system contract $\mathcal{C}_S$. A simulation-based design space exploration is then performed using continuous-time or hybrid behavioral models to jointly assess the architecture and the controller at a lower abstraction level. The exploration methodology presented in this thesis can be directly applied at the corresponding design stage (architecture selection) to search for the optimal implementation that satisfies the contract $\mathcal{C}_T$ of the architecture. In other words, it is proposed as one step in the hierarchy of a holistic cross-layer design flows, as illustrated on Figure 1.3 and Figure 1.4.

Similarly, as also shown on Figure 1.3, our approach can be used at the top level of the V-model [2], that is applied in some industrial domains, e.g., avionics and automotive. This model suggests a top-down design process, starting from high-level system-wide architecture (exactly, what our methodology is developed for) and moving through the subsystem design to

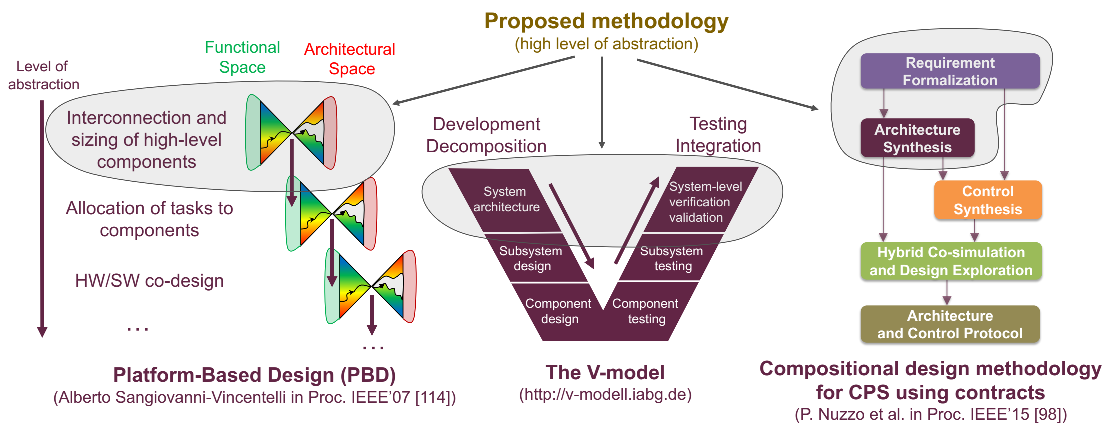

Proposed methodology
(high level of abstraction)
Level of abstraction
Functional Space
Architectural Space
Interconnection and sizing of high-level components
Allocation of tasks to components
HW/SW co-design
...
Platform-Based Design (PBD)
(Alberto Sangiovanni-Vincentelli in Proc. IEEE'07 [114])
Development Decomposition
Testing Integration
System architecture
Subsystem design
Component design
System-level verification validation
Subsystem testing
Component testing
The V-model
(http://v-modell.iabg.de)
Requirement Formalization
Architecture Synthesis
Control Synthesis
Hybrid Co-simulation and Design Exploration
Architecture and Control Protocol
Compositional design methodology for CPS using contracts
(P. Nuzzo et al. in Proc. IEEE'15 [98])

Figure 1.3 – Impact of the proposed architecture exploration methodology: our techniques can be applied at a high level of abstraction within holistic flows used in CPS design.

individual components. These steps are followed by a bottom-up integration and verification process, i.e., the last step is verifying the system as a whole. Different subsystems (e.g., mechatronic, electronic) are typically designed and manufactured by different suppliers. To ensure their correct integration at later stages and to avoid possible errors, requirements for these subsystems have to be provided early in the design by synthesizing a high-level architecture with correctness guarantees. They can then independently follow their own technology-oriented design processes, with certain design decisions being constrained by the capabilities of the top-level architecture.

A holistic model-based design methodology for CPS, proposed in [59], suggests a set of codependent steps that could be iteratively taken to progress towards the final implementation of the system. Although the approach primarily focuses on hierarchical modeling of heterogeneous components for analyzing the behavior and the performance of CPS architectures, automated synthesis of the latter is also anticipated. Starting from a concept design phase, the authors propose to select a proper set of computation models, assemble the models of physical processes and hardware and evaluate the two via simulation. The process is followed by software synthesis and formal verification of the system at each level of abstraction. Our techniques can augment several steps of the proposed design flow with generating correct-by-construction architectures at top levels of abstraction. This is beneficial for the proposed simulation-based design exploration since the architectural models would have formal guarantees and bounds on a set of system-level properties.

### 1.4.2 Architecture Exploration

As mentioned in Section 1.2, one can classify existing architecture exploration approaches and methods into several groups that use different engines/functions for evaluating system properties and generating architectures. First, *simulation-based* group focuses on the *analysis* of architectures, i.e., on the evaluation of system performance and behavior. The goal of state-

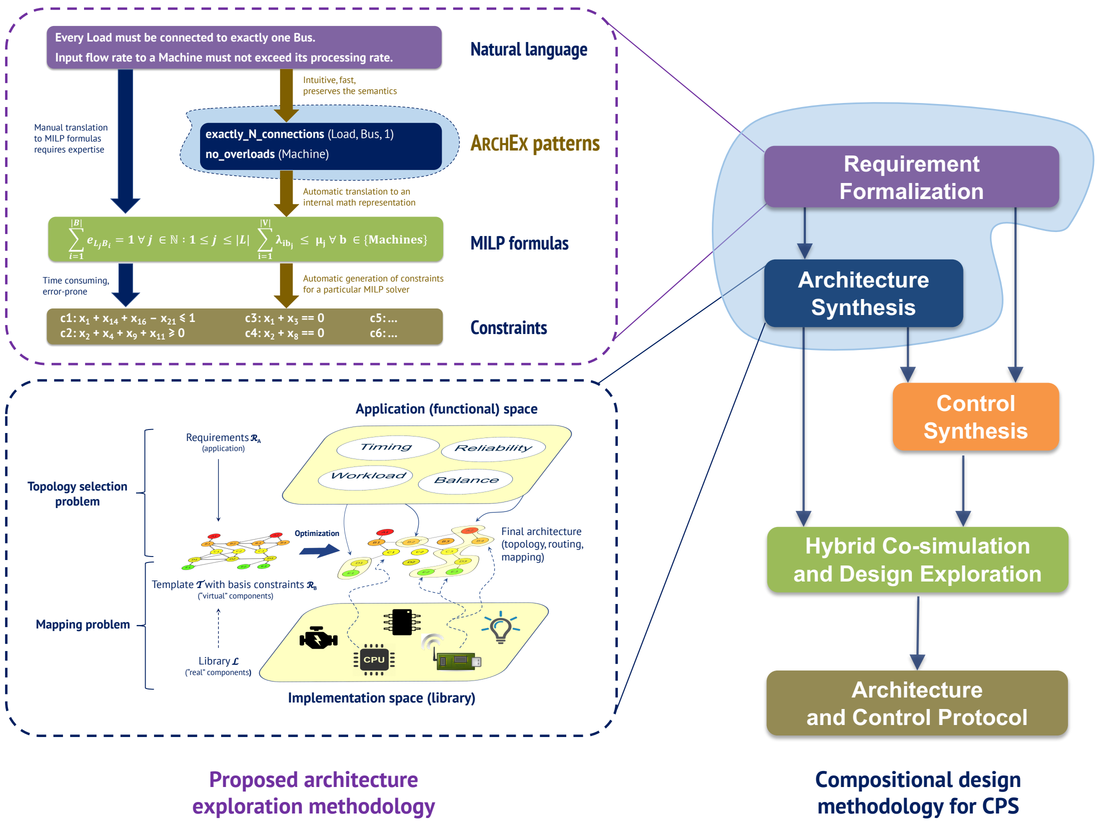

Every Load must be connected to exactly one Bus.
Input flow rate to a Machine must not exceed its processing rate.
Natural language
Intuitive, fast, preserves the semantics
Manual translation to MILP formulas requires expertise
exactly_N_connections (Load, Bus, 1)
no_overloads (Machine)
ARCHEX patterns
Automatic translation to an internal math representation
$\sum_{i=1}^{|B|} e_{L_j B_i} = 1 \ \forall j \in \mathbb{N} : 1 \leq j \leq |L| \quad \sum_{i=1}^{|V|} \lambda_{ib_j} \leq \mu_j \ \forall b \in \{\text{Machines}\}$
MILP formulas
Time consuming, error-prone
Automatic generation of constraints for a particular MILP solver
c1: $x_1 + x_{14} + x_{16} - x_{21} \leqslant 1$
c2: $x_2 + x_4 + x_9 + x_{11} \geqslant 0$
c3: $x_1 + x_3 == 0$
c4: $x_2 + x_8 == 0$
c5: ...
c6: ...
Constraints
Application (functional) space
Requirements $\mathcal{R}_A$ (application)
Timing
Reliability
Workload
Balance
Topology selection problem
Optimization
Final architecture (topology, routing, mapping)
Template $\mathcal{T}$ with basis constraints $\mathcal{R}_B$ ("virtual" components)
Mapping problem
Library $\mathcal{L}$ ("real" components)
CPU
Implementation space (library)
Requirement Formalization
Architecture Synthesis
Control Synthesis
Hybrid Co-simulation and Design Exploration
Architecture and Control Protocol
Proposed architecture exploration methodology
Compositional design methodology for CPS

Figure 1.4 – Proposed architecture exploration approach positioned within the cross-layer cyber-physical system design methodology proposed by P. Nuzzo et al. [98, 99]. Following the platform-based design paradigm, topology selection and mapping problems are jointly solved to synthesize the optimal architecture, while the requirement formalization stage is enhanced by the pattern-based language.

of-the-art *satisfiability-based* approaches is twofold: they can be applied both for verifying the architecture with respect to certain properties, i.e., for analysis, as well as for locating a feasible solution in the design space, i.e., for *synthesis*. Finally, *optimization-based* techniques rather focus on synthesis. They intend to explore the feasible space in order to find an optimal architecture with respect to a given objective. In the following, we overview existing approaches and tools that belong to one or more of these groups.

**Simulation-based approaches**. Several modeling languages have been proposed for specification, design, analysis, verification and validation of complex systems. In particular, SysML [100] is a UML-based language that removes the software-centric restrictions of the latter and is actively used in systems engineering in different industrial domains. The Architecture Analysis and Design Language (AADL) [9], initially developed for the avionics domain, is used to model both the software and the hardware of embedded real-time systems. Both SysML and AADL have built-in constructs, views and diagrams for high-level architecture design, while created models can be verified using conventional techniques, e.g., model checking. MODELICA is a multi-domain modeling language containing different types of subcomponents (e.g., mechanical, hydraulical, electrical, thermal, electronic and others). CPS models written in the MODELICA language can be simulated in many available toolboxes, such as JMODELICA [60] or DYMOLA [36].

CPS architectures can be designed and evaluated with simulation-based tools. In particular, the ones that are capable of capturing both the properties of the embedded systems and of the physical plant, are favorable. For example, in Ptolemy II [107] designers can construct and simulate actor-based models with heterogeneous components by using one of the many provided computation models, e.g., discrete-event, continuous time, synchronous dataflow, finite-state machines, and others. The models can be mixed in a hierarchy controlled by a global scheduler (director). The METRO II framework [30] has been designed based on the PBD methodology with the intent of addressing the challenges posed by heterogeneous CPS. It introduces the techniques of representing non-functional quantities and coordinating different models for performance evaluation. The capabilities of expressing both discrete and continuous systems are also provided by widely adopted commercial toolboxes, such as Simulink [119] and LabVIEW [70].

In general, simulation tools are typically focused on performance evaluation of CPS architectures, rather than on design space exploration. Yet, some approaches are taming the simulation-based exploration. For example, Metronomy [47] integrates the functional modeling capabilities of Ptolemy II and architectural modeling from METRO II via a mapping interface and performs design exploration via co-simulation. Finn et al. combine discrete optimization routine (MILP) that generates architecture candidates of an aircraft environmental control system with continuous sizing and optimization, performed using MODELICA [40]. In [87], topologies for body area networks are synthesized by solving a MILP problem in a loop with running network simulation using the Castalia toolbox [19]. Simulations are used to verify the candidate topologies and to generate (learn) additional constraints for augmenting the original formulation and guiding the MILP solver towards a satisfying solution. This is a kind of “lazy MILP modulo simulation” approach, where the latter is used as a background theory for verifying the correctness of the topology.

Our approach is, in general, complementary to aforementioned simulation-based techniques for CPS design, since it intends to generate correct-by-construction architecture candidates that can be analyzed and verified using simulation models. For example, it can be used in [40] or [87], since it can fully capture the system requirements from these works. At the same time, the proposed methodology is more expressive and reusable across different CPS domains, while MILP formulations from [40, 87] are problem-specific.

In many aspects, model-based approaches are better suited for lower levels of more detailed design, where many high-level decisions have already been made. When few configurations out of a small design space have to be carefully evaluated (e.g., for timing or power consumption behaviors), simulators certainly provide greater accuracy. Still, the concept design stage is also tackled by several approaches [17, 23]. For example, Bhave et al. [17] analyze the consistency of a set of high-level *architectural* models that capture different viewpoints of a CPS. In contrast, Canedo et al. [23] address the *functional* space, i.e., they evaluate different implementations of the same high-level functionality. They propose a functional modeling compiler that automatically generates a set of architectural models from a functional model, which are further converted to simulation models and executed by a dedicated toolbox for performance analysis. In other words, [23] is a simulation-based design space exploration technique. Still, in many application domains, such exploration may become very expensive due to a huge number of candidate architectures to be simulated. In this work we opt, instead, for techniques that perform an efficient search over large high-level design spaces (e.g., wireless network topologies, interconnection networks of production machines) to find optimal solution to be further evaluated and refined, in particular, with simulation models.

**Satisfiability-based approaches**. Many approaches leverage symbolic constraint satisfaction techniques, such as Satisfiability Modulo Theories (SMT) [15, 31], for architecture exploration. The expressiveness of SMT allows it to be applied to rapidly search feasible system configurations. For example, Reimann et al. [112] propose an SMT-based approach for efficient exploration and pruning of the high-level CPS design space under stringent timing constraints. Kumar et al. [68] address the problem of assigning speeds to resources in a real-time system with timing and energy constraints by developing an SMT solver with real-time calculus [123] as a background theory. A number of efficient SMT solvers, such as Z3 [31], openSMT [21] or Yices [35], as well as a standardized description language, SMT-LIB [14], are available.

SMT typically use first-order, linear background theories, e.g., difference logic or linear arithmetic overs reals or integers. Some alternative approaches, however, extend the existing paradigm with non-linear convex theories. For instance, the CalCS toolbox [95] exploits the information from the solution of convex optimization problems to establish the satisfiability of conjunctions of convex constraints. CalCS also provides a formulation that allows it to generate counterexamples, perform conflict-driven learning and approximate non-convex constraints. Gao et al. [41] developed an SMT solver, *dReal*, for nonlinear theories (e.g., trigonometric, exponential) over the reals using efficient relaxation mechanisms. These two approaches are very attractive to be used for capturing various nonlinear properties of cyber-physical systems, however, none of them has been applied to CPS design exploration problems as yet.

More recently, a *Satisfiability Modulo Convex Optimization (SMC)*, was proposed by Shoukry et al. [118]. It uses a lazy combination of SAT solving and convex programming to search for a satisfying assignment. Even though it uses convex optimization as a background theory, it is not an optimization technique, since it rather focuses on feasibility problems. Instead, it intends to leverage state-of-the-art algorithms from both Boolean and convex analysis domains to extend the application of SMT to nonlinear problems. No case studies in the CPS domain have yet been proposed for this novel technique, however, we find the approach attractive, in particular, for architecture exploration problems.

SMT-based design exploration methods provide a clear advantage of being expressive, e.g., with respect to Boolean satisfiability (SAT) techniques. Therefore, a large body of CPS architectural requirements can be naturally captured by an SMT program. Advanced techniques, such as [95, 41] provide even more capabilities by allowing designers to use nonlinear formulas, which is very typical for physical processes and phenomena. The performance of SMT, however, highly depends on the structure of the design exploration problem. For example, if the majority of the variables is Boolean, few calls are made to the theory solvers, and the SAT solver can explore the search space very efficiently. On the other hand, SMT techniques can demonstrate limited scalability on the formulations with large number of real constraints, where they can be outperformed by state-of-the-art optimization algorithms, such as branch-and-cut and its extensions.

The SMT formulation of architecture exploration problems is still a non-trivial process, which requires deep knowledge of underlying mathematical abstractions and understanding the principles of SMT solvers. The problem has been recently addressed by Peter and Givargis [101], who proposed a component-based synthesis method for embedded and cyber-physical system architectures using SMT. They have developed CoDeL, a description language that facilitates the design specification and automatic translation into an SMT program. The main idea of CoDeL is similar to our pattern-based language, provided by ArchEx, since it also significantly lowers the burden of formulating exploration problems. The main difference is that ArchEx uses *optimization* to find *cost-effective* solutions, while CoDeL and SMT-based techniques in general aim at determining a *feasible* one without minimizing a cost function. Recent research efforts of applying SMT for solving optimization problems are discussed in the next section.

Other symbolic techniques, in particular, ordered binary decision diagrams (OBDDs) [91], can be applied for rapidly pruning large high-level design spaces of CPS [90]. Also, symbolic model checking methods have been used for computing feasibility regions of system parameters by expanding unsatisfiable execution traces of the system model to cutting planes [27]. The latter approach has been applied to real-time scheduling, but is very promising also for CPS architecture design.

**Optimization-based approaches**. This group of techniques is related to finding a valid high-level architecture with a minimal cost value (e.g., number or weight of components, energy consumption, idle time). Corresponding design exploration problems are cast as optimization problems. In particular, *mixed integer linear programming (MILP)* is being increasingly adopted to CPS concept design problems. Some existing approaches fully rely on solving MILP problems [106, 103, 45], others use MILP in combination with other techniques, such as heuristic functions for optimization [108], conflict-driven learning functions for pruning the search space by incrementally generating additional constraints for existing formulation [99, 12], simulation [87] and convex optimization [40]. MILP has NP-hard complexity, in contrast with linear programming (LP), for which polynomial algorithms exist. There is no standard commonly adopted solution technique, while existing methods are typically based on LP relaxations and/or heuristics. Yet, MILP is largely used, and the empirical performance of state-of-the-art MILP solvers, such as CPLEX [54], Gurobi [48] or MOSEK [89], is impressive.

An optimization-oriented design methodology following the PBD paradigm was proposed in [103] for aircraft electrical power systems. In particular, a generic nonlinear formulation is proposed for the exploration of power system architectures, i.e., for generator selection and distribution network synthesis. An approximate formulation as a binary optimization problem with all linear constraints, which can be handled by MILP solvers, is also provided. In [99] the design flow from [103] is extended to a more holistic approach, which enables the co-design and synthesis of electrical power system topology and control using contracts. In particular, the architecture selection problem formulation is augmented with reliability constraints. In [12] the authors tackle the exponential complexity of enumerating failure states of the system by proposing two algorithms for solving the exploration problem - MILP with approximate reliability constraints (a monolithic optimization problem) and MILP *modulo* reliability, an iterative technique, in which calls to the optimizer alternate with a heuristic learning function. Our methodology supports generalized versions of both these algorithms, which is further discussed in Chapters 3 and 4.

Overall, as already mentioned in Section 1.2, the methodology presented in this dissertation builds on the works [99, 12] in many aspects. Our additions and extensions include a novel encoding that separates the topology selection problem from the mapping problem (Section 3.1), more general and more efficient mapping mechanism (Section 2.2.1), more types of supported requirements (Section 3.2), extensible and customizable framework (Chapter 4) and improved usability facilitated by the pattern-based formal language (Section 4.2).

The authors of [50] proposed an approach of high-level CPS architecture synthesis that resembles an SMT solver, but with an ILP solver as a core instead of SAT. This allowed them to efficiently solve resource planning and scheduling problem of a large-scale system from the avionics domain. The key idea in [50] is to separate scheduling constraints from connectivity and balance constraints. The developed *ILP Modulo Scheduling* solver was able to quickly progress towards a feasible solution by solving smaller problems and leveraging a learning function for checking the schedulability, finding unsatisfiable cores and deriving new lemmas to be added to the original formulation. This approach was later generalized in [80] to a rigorous *ILP Modulo Theories (IMT)* method. In IMT, certain terms in the MILP encoding can be *interfaces* to background theories, so that theory solvers can be plugged in to verify them. IMT provides a sound and complete optimization procedure for the combination of MILP with stably-infinite theories [80]. The decision procedure, called Branch and Cut Modulo Theory or BC($T$), is an extension of the classical branch and cut, which is the most established family of algorithms for solving ILP instances. An implementation of a BC($T$)-based solver, **Inez**, has also been proposed [79]. In our methodology we use an iterative optimization algorithm, in particular for separation of reliability constraints (MILP modulo reliability, initially proposed in [12]), so that our approach is in part inspired by the IMT.

Finn et al. [40] developed a mixed discrete-continuous optimization scheme for synthesizing CPS architectures, which they evaluated on an aircraft environmental control system design. The authors use a MILP solver (CPLEX) in a loop with a simulator (JMODELICA). The former proposes architecture candidates by optimizing over a discrete space of component parameters (e.g., types of ducts), while the latter runs a convex optimization over a continuous space by using the Nelder-Mead method [92]. The latter is a well known numerical method applied to nonlinear optimization problems, for which derivatives may not be known. Optimization with simulation in the loop has been also proposed by Moin et al. [87] for the optimized design of a human intranet network. A mixed integer linear program generates candidate network architectures under coarse energy constraints, which are then checked by a discrete-event network simulator under reliability constraints. Simulation results generate lower bounds on the power consumption, which are added to the MILP to prune the search space and achieve faster convergence compared to other state-of-the-art methods, such as simulated annealing [3].

Pinto et al. [106] proposed a generic MILP formulation for synthesizing wireless network architectures for building automation and control. The goal is the minimization of the cost of network nodes under a set of topological, link utilization and link quality (end-to-end delay, packet error rate) constraints. More recently, Puggelli et al. [108] extended the formulation from [106] by incorporating different routing patterns (e.g., unicast, multicast) and power consumption constraints for indoor wireless sensor networks. They explore a broader design space and solve a more complex optimization problem. However, according to the results provided in [108], the problem becomes impractical, i.e., no solution is found within a timeout, for networks of more than 50 end devices. Therefore, the authors also propose a polynomial-time heuristic algorithm in place of the NP-hard MILP problem that applies Dijkstra's shortest path routine for synthesis of large networks. Instead, in our work we use an approximate algorithm for encoding network paths to *symbolically* generate compact MILP formulations that can scale to hundreds of nodes. Moreover, with respect to [106, 108], our approach is more general, i.e., it not only supports the formulations from [106, 108], but also other concerns (e.g., lifetime constraints, localization constraints). Overall, our approach is superior both in network size and the dimensionality of the design space, which now includes the sizing of network components (e.g., choosing the transmission power of devices, selecting external antennas and so on). We refer the reader to Section 6.4 for a more detailed comparison of our approach with other related techniques in the wireless network domain.

Other classes of optimization methods include nonlinear convex and non-convex programming. The former is a mature technology with efficient polynomial-time solution strategies, such as interior-point methods. State-of-the-art optimization tools, e.g., CPLEX, Gurobi, MOSEK or SeDuMi [4], can handle convex programs with thousands of constraints and hundreds of variables, in few tens of seconds on a desktop computer. The latter, non-convex problems are among the hardest ones in optimization, while a standard widely-adopted method for solving them is still missing. Several approaches, including local and global optimization methods, exist, as well as the tools, such as IPOPT [55] (open-source) and NPSOL [94] (proprietary). While these nonlinear optimization techniques (especially, convex) are widely applied in other CPS design problems, such as control, very few works (one example is [40]) leverage them for solving high-level architecture exploration problems.

A platform-based framework for the design of interconnection networks, COSI [104], has been proposed for synthesizing high-level network-on-chip architectures. COSI leverages component-based model for representing system architectures and several quantity models for capturing various performance aspects. It allows designers to plug in domain-specific heuristic algorithms for solving mapping problems. A case study of synthesizing networked control systems in building automation using COSI has been presented in [105].

Recently, many efforts have been made in applying the SMT-based techniques to solving optimization problems [116, 28, 93, 117, 74]. For example, Cimatti et al. [28] presented a *Satisfiability Modulo Cost* approach by augmenting an SMT solver with pseudo-boolean constraints representing cost functions. The key idea is to iteratively use the SMT procedure and tighten the admissibility range of one or multiple objectives. Two algorithms have been proposed for running the optimization, one based on branch-and-bound and one based on binary search. The technique reflects the general principle of *Optimization Modulo Theories (OMT)*, i.e., solving SMT problems and updating the cost-related terms of the SMT formula until no better solution is found. The main difference from aforementioned ILP Modulo Theories techniques is, therefore, that in OMT the optimization is performed *externally* from the solver being used. The latter only checks the feasibility of every new solution. In contrast, in IMT optimization is performed *inside* a MILP solver, while the theory solver is responsible for counterexample generation and learning. Finally, the branch-and-cut algorithm used in most of existing MILP solvers estimates the best achievable cost at every step (considering the linear relaxation of the solution). This lower bound can be compared to the best integral solution found so far in order to estimate a gap between the two, which can be used, for example, as a termination criteria for the solver (e.g., when a solution is found, which is only 10% worse than the bound). In OMT, it is not possible to estimate such optimization gap.

Several optimizing SMT solvers have been proposed, for example, MathSAT [22] (implementing the approach from [28]), OptiMathSAT [117], SYMBA [74] and $\nu$Z [18]. They compete in supporting different theories (e.g., linear arithmetic over rationals or integers) and objective functions, conflict-driven learning techniques and performance. None of them currently supports nonlinear cost functions. While the exhaustive comparison of these existing tools is out of scope of this work, we note that none of them has been applied to cyber-physical system design problems so far. OMT techniques are still in their infancy, but we expect their future applications to architecture exploration problems and consider them an interesting direction of future work in the area of CPS design.

## 1.5 Outline of the Dissertation

This dissertation consists of seven chapters which are briefly summarized as follows. In the current chapter we discussed the motivation, the objectives and the contributions of our work and highlighted the phases of cyber-physical system design flows, where our methodology is positioned. We also discussed existing state-of-the-art approaches to high-level architecture exploration of CPS, while more related works for particular case studies are also mentioned in

Chapters 5 and 6 of the dissertation.

In *Chapter 2* we introduce the formal theoretical background of our methodology and the main definitions that we use throughout the thesis. We describe the graph-based representation of CPS architectures, which is employed in this work, and define the generic basis of architecture exploration problems in terms of variables and mixed integer linear constraints used in the proposed formulation. We further demonstrate and exemplify, how the basis can be used to instantiate system requirements.

*Chapter 3* presents the exploration methodology, in which the final architecture is a meet-in-the-middle of the two search spaces, i.e., application space (architecture requirements) and implementation space (library of components) and two respective problems, i.e., topology selection and mapping. We discuss the flow of the methodology, outlining the main steps: specification, encoding, solving and analysis. We then provide the formulations of a variety of CPS requirements that are currently captured in our formulation (e.g., timing, reliability, workload, routing). We further elaborate on the mapping problem and on the solving and analysis techniques. Finally, the chapter is concluded with the algorithmic part of the methodology, including algorithms for approximate encoding of path constraints and iterative optimization techniques.

In *Chapter 4* we introduce ArchEx 2.0, an extensible framework for CPS architecture exploration that we developed in support of the proposed methodology. We pay much attention to the pattern-based language, which plays an important role in the specification, since patterns significantly lower the burden of formulating exploration problems. We then explain the software structure of ArchEx and its extension for wireless network design. Finally, we give a step-by-step example of using the framework for formulating and solving a simple problem for a fuel management system.

In *Chapter 5* we run the numerical evaluation of our approach on two reliability-driven industrial case studies: aircraft electrical power distribution network and reconfigurable manufacturing system. By solving these problems in ArchEx, we demonstrate the efficiency and the scalability of the methodology and show the advantage of using patterns for formulating complex problems.

In *Chapter 6* we focus on a significantly different domain, wireless networks. We then show, how our approach can be applied to topology synthesis and component sizing of wireless sensor networks on two case studies, data collection and localization networks. In addition, by reporting on the results of the channel measurement campaign, we propose several improvements for conventional channel models that can be used in design tools including, but not limited to, network simulators. These models are included in our library in order to map virtual links of the architecture template to real (physical) ones.

Finally, in *Chapter 7* we recap the contributions of this dissertation and summarize the obtained results. Among the conclusions, we also also highlight several promising research directions for future work.

# 2 Preliminaries

*This chapter provides a background for the concepts introduced in this thesis. We give the main definitions related to system architecture and its graph-based representation. We then introduce a set of variables and generic mixed integer linear constraints that we call the "basis" of the exploration problem. This basis is a common semantic domain, in which it is possible to define a variety of requirements for different classes of cyber-physical systems. We further briefly review a set of techniques for expressing complex constraints in a linear form. Finally, we cast the architecture exploration problem as an optimized mapping problem on a reconfigurable graph, where "virtual" components of the architecture are mapped into "real" components from pre-defined libraries to minimize a cost function under correctness guarantees.*

## 2.1 Terminology

### 2.1.1 Architecture, Template, Topology Configuration

We assume that a CPS *architecture* is represented by a network of interconnected components, which are selected from a *library* (collection) $\mathcal{L}$ and comply with a set of *composition rules*. Each component in $\mathcal{L}$ has a set of *attributes* capturing its functional and extra-functional properties. Extra-functional properties include, for instance, energy consumption, processing delay, and cost. Each component has a set of *terminals* parameterized with *terminal variables*. Input and output terminals are used to send and receive signals or the values of terminal variables. Composition rules define which connections are allowed and how terminal variables may be assigned. Components can have different *types*, i.e., different roles or functions in the system.We focus on networks whose components exchange entities or quantities via *flows* (e.g., message flow, power flow, product flow). Therefore, certain components have the role of sources or sinks.

**Definition 2.1** (Architecture). *A system* architecture *is a directed graph $G = (V, E)$, where $V$ is a set of components (nodes) while an edge $e_{ij} \in E$ represents an interconnection from $v_i$ to $v_j$, also written as $e_{v_i v_j}$, with $i, j \in \{1, \ldots, |V|\}$, $|V|$ being the cardinality of $V$.*

Graph $G$ can be represented by its adjacency matrix, and in the following we use $E$ to denote this matrix unless specially noted that $E$ is a set. Edges $e_{ij} \in E$ are interpreted as binary variables that evaluate to one (zero) to indicate the presence (absence) of interconnections between nodes. We say that an edge between $v_i$ and $v_j$ is *instantiated* (or used) if $e_{ij} = 1$. When at least one incoming edge variable $e_{ij}$ or outgoing edge variable $e_{ji}$ for a component $v_i$ evaluates to one, we say that $v_i$ is *connected*. Trivially, connected nodes are instantiated, i.e., used in the final architecture, while the opposite may not hold. Instantiated components are those components that perform a certain function within the system, which is a more general definition with respect to being connected. This definition is futher explained later in this section.

A set of all possible architectures for a given system forms an *architectural space*, which designers explore to find a feasible (or optimal, sub-optimal) instance that meets the specification. To formalize the exploration problem, we resort to the definitions of *template* and *topology configuration* specified as follows:

**Definition 2.2** (Template). *A* template $\mathcal{T}$ *is a reconfigurable architecture, i.e., a graph with a fixed set of nodes $V$ but variable set of edges $E$, which can be altered by the use of the variables $e_{ij}$. The size of the template is determined by the number of nodes in the graph, i.e., $|\mathcal{T}| = |V|$.*

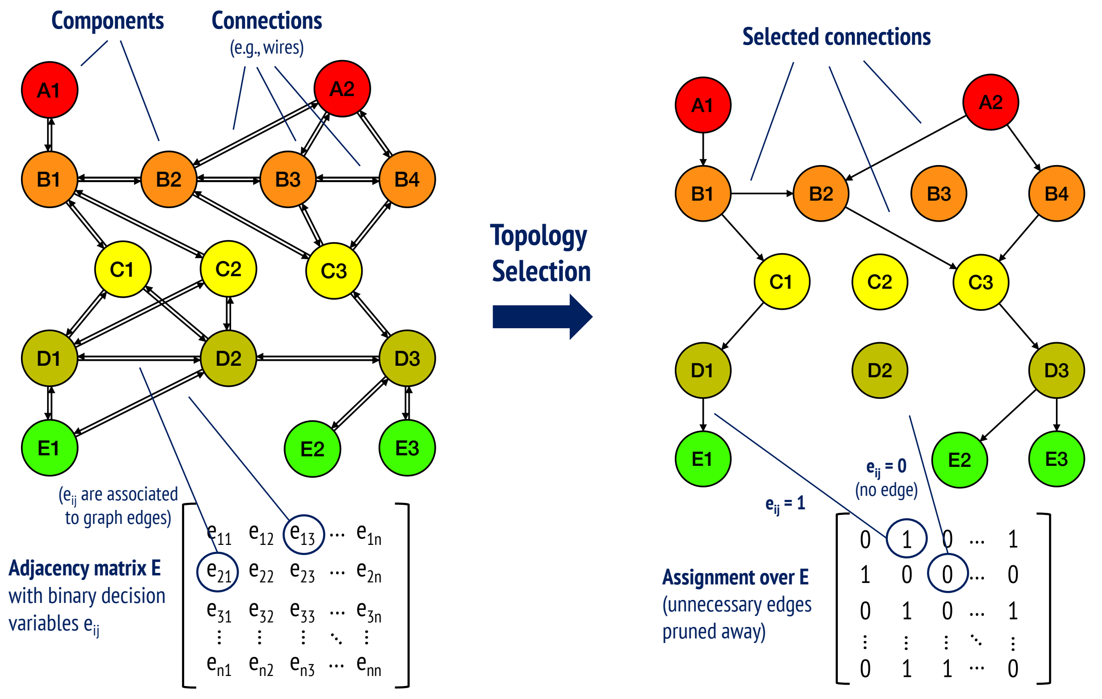
**Definition 2.3** (Topology configuration). *A* topology configuration $E^*$ *is an assignment over the variables in $E$, marking the subset of edges from $\mathcal{T}$ that are instantiated. It can be seen as an instance of $\mathcal{T}$.*

Due to its reconfigurability, $\mathcal{T}$ is one of the inputs of the exploration problem. It has the maximal number of nodes and all possible edges, i.e., those that are not restricted by the composition rules (a restriction is enforced, for example, if components of certain types cannot be directly connected). In turn, a topology configuration is a design decision, i.e., it defines which components are used in the final architecture and how they are connected. The number of nodes and edges in the topology configuration can be smaller than in the template. Some of them may not be necessary for the system to function correctly, i.e., unused, and, therefore, are pruned away to minimize some objective function.

**Example 1** (Topology Selection). *We assume a sample architecture that consists of 5 distinct component types: A,B,C,D and E. Template $\mathcal{T}$ of the architecture is shown on Figure 2.1a, where components of different types have different colors. $\mathcal{T}$ includes all possible connections, while the graph is not complete, because some edges are restricted by composition rules (e.g., direct connections between B and D are not permitted). The total number of nodes in $\mathcal{T}$ is 15. Figure 2.1b represents an assignment over E, i.e., a configuration $E^*$ of the system topology selected from the architectural space. Only 12 nodes are used (all but B3, C2 and D2), and all connections that are not essential for system function are removed.*

### 2.1.2 Mapping

Every component and connection of a system at a given level of abstraction can be implemented in different ways (e.g., by different electrical or mechanical devices, program codes).



(a) Architecture template $\mathcal{T}$ (b) Selected topology configuration $E^*$

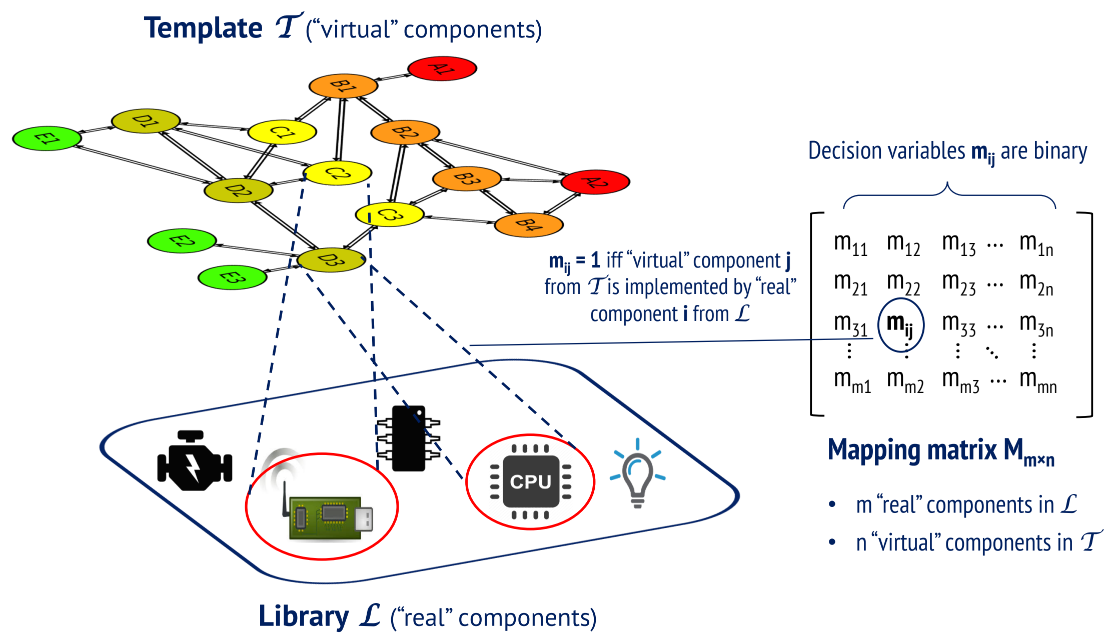
Figure 2.1 – Topology selection example: (a) Template $\mathcal{T}$ with all possible connections between components (note that some edges are missing, because they are restricted by the composition rules), and its adjacency matrix containing binary decision variables $e_{ij}$; (b) A topology configuration $E^*$, i.e., a final system topology selected from $\mathcal{T}$, as an assignment over the decision variables $e_{ij}$.

These implementations, as well as the resulting system, can have different characteristics and performance. The choice of the implementations, or the mapping, is defined below.

**Definition 2.4** (Mapping)**.** *A mapping $\mathcal{M} : V \rightarrow \mathcal{L}$ is a function that associates each graph node $v \in V$ (also called "virtual"component) with some component $l \in \mathcal{L}$ in the library ("real" component or device). We represent the mapping by assigning to each pair $(l_i, v_j)$, with $l_i \in \mathcal{L}$ and $v_j \in V$, a binary variable $m_{ij} \in M$, $M$ being the mapping matrix, such that $m_{ij}$ is one if $v_j$ is mapped to $l_i$ and zero otherwise.*

Clearly, if a component $v_j$ is not associated to any element from $\mathcal{L}$, then it is not used in the final architecture, and vice versa. We can now define *instantiated* components as the ones which are mapped to at least one component in the library $\mathcal{L}$, i.e., $\sum_{i=1}^{|\mathcal{L}|} m_{ij} \geq 1$. In the following, we also use auxiliary Boolean variables $\delta_j$ to define some system requirements: $\delta_j$ is one if component $v_j$ is instantiated (the aforementioned sum of mapping variables for $v_j$ is greater than zero), and zero otherwise.

In general, map $\mathcal{M}$ can also be extended for graph edges $e \in E$, while in our formulation edges are directly mapped to a pre-defined set of connection elements in the library, e.g.,


Template 𝒯 ("virtual" components)
A1
B1
A2
B2
B3
B4
C1
C2
C3
D1
D2
D3
E1
E2
E3
Decision variables $m_{ij}$ are binary
$m_{ij}$ = 1 iff "virtual" component j from 𝒯 is implemented by "real" component i from ℒ
CPU
Mapping matrix $M_{m\times n}$
m "real" components in ℒ
n "virtual" components in 𝒯
Library ℒ ("real" components)

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & \cdots & m_{1n} \\ m_{21} & m_{22} & m_{23} & \cdots & m_{2n} \\ m_{31} & m_{ij} & m_{33} & \cdots & m_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{m1} & m_{m2} & m_{m3} & \cdots & m_{mn} \end{bmatrix}$$

Figure 2.2 – Mapping of "virtual" components from template $\mathcal{T}$ of the architecture to "real" devices from a library $\mathcal{L}$.

switches, wires, or wireless links. That is, their mapping is fixed, so if an edge is used in the final architecture then its corresponding library element is already known. Both nodes and edges in the graph are labeled with types, terminal variables, and attributes corresponding to those from the library $\mathcal{L}$, and their values are determined through the mapping as further demonstrated in Section 2.3.1 of this chapter.

**Example 2** (Mapping)**.** *Associating "virtual" components of template $\mathcal{T}$ to "real" components from library $\mathcal{L}$ ($|\mathcal{T}| = n, |\mathcal{L}| = m$) is done using the mapping matrix $M_{m\times n}$ as shown on Figure 2.2. For example, component C2 from $\mathcal{T}$ is mapped to a radio transceiver, while D3 is associated to a processor. Corresponding values $m_{ij} \in M$ evaluate to one. We can then say that C2 and D3 are instantiated in the architecture.*

### 2.1.3 Paths, Routing, Functional Flow

**Definition 2.5** (Path)**.** *A network* path *(route) $\pi(v_0 \to v_n)$ is a sequence of distinct nodes $\{v_0,\dots,v_n\}$ such that $e_{v_i v_{i+1}} \in E$ for each $i$. We write $|\pi|$ to denote the length of $\pi$. For a pair of nodes $q = (v_s, v_d)$, where $v_s$ and $v_d$ are, respectively, source and destination nodes, we define $\pi^q := \pi(v_s \to v_d)$.*

For computing network paths, edges $e_{ij} \in E$ can be labeled with binary variables $y^{\pi}_{ij}$, which evaluate to one if $e_{ij}$ connects the nodes $(v_i, v_j)$ in $\pi$ and zero otherwise. Therefore, every path $\pi$ has an associated vector $y^{\pi}$ with $|y^{\pi}| = |E|$. In turn, nodes $v \in V$ can be labeled with binary variables $w^{\pi}$: $w^{\pi}_i$ is one if node $v_i$ belongs to path $\pi$ and zero otherwise. We call $y^{\pi}$

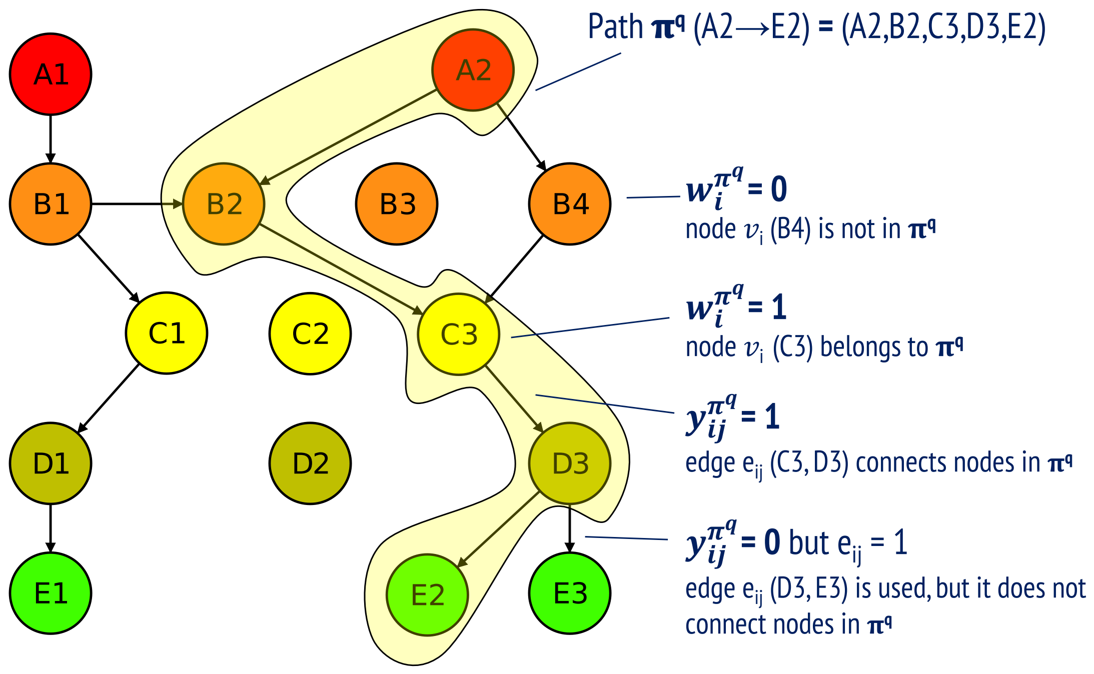

Path $\pi^q$ (A2⟶E2) = (A2,B2,C3,D3,E2)
A1
A2
B1
B2
B3
B4
C1
C2
C3
D1
D2
D3
E1
E2
E3
$w_i^{\pi^q} = 0$
node $v_i$ (B4) is not in $\pi^q$
$w_i^{\pi^q} = 1$
node $v_i$ (C3) belongs to $\pi^q$
$y_{ij}^{\pi^q} = 1$
edge $e_{ij}$ (C3, D3) connects nodes in $\pi^q$
$y_{ij}^{\pi^q} = 0$ but $e_{ij} = 1$
edge $e_{ij}$ (D3, E3) is used, but it does not connect nodes in $\pi^q$

Figure 2.3 – Example of a network path $\pi^q$, $q = (A2, E2)$, and corresponding path variables $w^{\pi^q}$ and $y^{\pi^q}$ for nodes and edges of the graph.

and $w^{\pi}$ *path variables* and associate them to each other and to variables $e_{ij}$ of the topology configuration using a set of generic constraints introduced in Section 2.2.2 of current chapter.

**Example 3** (Path variables)**.** *Figure 2.3 illustrates a path* $\pi^q = (A2, B2, C3, D3, E2)$*, where* $q = (A2, E2)$*. For the nodes visited along the path (e.g., C3), path variable* $w_i^{\pi^q} = 1$*,* $i$ *being the numerical index of the node* $v \in V$*, while remaining nodes (e.g., B4) have* $w_i^{\pi^q} = 0$*. Similarly, edges that connect nodes in* $\pi^q$*, for instance,* $(C3, D3)$*, have* $y_{ij}^{\pi^q} = 1$*, while path variables for other edges evaluate to zero. In particular, it is possible that an edge is instantiated, i.e., edge variable* $e_{ij} = 1$*, while this edge, e.g.,* $(D3, E3)$*, does not belong to* $\pi^q$*, so that* $y_{ij}^{\pi^q} = 0$.

We note that path variables are created only for those pairs of nodes which must be connected by a path as required by the user (it is possible to define several paths between the same pair in case if more than one replica is required, e.g., for reliability), because of the exponential growth of problem complexity for enumerating all possible paths. At the same time, all possible edges from $\mathfrak{T}$ are associated with variables $y_{ij}^{\pi}$ for every required path $\pi$, i.e., any edge from $\mathfrak{T}$ can be used in any $\pi$. In other words, all possible paths are symbolically encoded in each $\pi$, and the corresponding complexity is also exponential. We discuss the techniques of drastically lowering this complexity and achieving more compact encodings in Section 3.5.1.

All *required* paths for a pair $q$ are stored in a set $\Pi^q$. Depending on routing requirements, $|\Pi^q| = 0$ if no paths are required for $q$ and $|\Pi^q| > 0$ if one or more routes are needed. We group the vectors $y^{\pi^q}$ and $w^{\pi^q}$ into a set $R = \{y^{\pi^q}, w^{\pi^q} | q \in Q, \pi^q \in \Pi^q\}$, $Q$ being the subset of pairs of nodes in $V$ for which a path is required.

**Definition 2.6** (Routing)**.** *A routing $R^*$ is an assignment over the set $R$ of variables $y^{\pi^q}$ and $w^{\pi^q}$, marking the subsets of edges and nodes* [1] *from $\mathfrak{T}$ that are used in system paths (routes).*

$R^*$ can be informally seen as a joint routing table for all components of the system, as well as its logical topology. It defines the choice of network routes between pairs of source (sending) and destination (receiving) nodes of a network, which is one of the design decisions for high-level architectures being considered in this work.

We call $P$ a *partition* over $V$, such that all components belonging to the same subset in $P$ have the same type. We also write $t_v$ to denote the type of component (node) $v$. Components $v_i$ and $v_j$ of the same type $t$ can also have different *subtypes* $s_{v_i}$ and $s_{v_j}$. Connections and paths between components with different subtypes can be allowed or restricted based on library composition rules or user-specified interconnection constraints. We call two components *compatible* if their subtypes allow them to be connected by a path. We call path $\pi$ a *valid* path if it guarantees that the signal from the source of $\pi$ can be accepted by its sink. Therefore, if source and sink are not compatible, $\pi$ must include some intermediate components (e.g., converters, modulators, filters) that alter the signal so that it becomes acceptable by the sink.

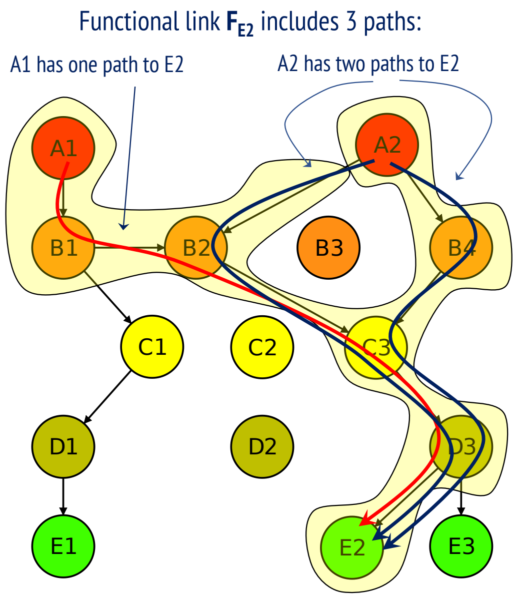
**Example 4** (Valid Paths)**.** *We assume an electrical power distribution network (EPN), where power is delivered from generators to loads via a set of intermediate components (e.g., buses). Compatible components of an EPN must have same voltage levels represented by subtypes: high voltage (HV) and low voltage (LV). Paths between HV (LV) generators and HV (LV) loads are valid. For a path connecting an LV generator to a HV load to be valid, a step-up transformer must be added in between.*

Let $S_1$ and $S_n$ in the partition $P$ include, respectively, all the sources and the sinks of the network. Then, a *functional link $F_i$* is the set of all valid paths from any source in $S_1$ to a sink $v_i \in S_n$. Such links are essential for a system to operate correctly. For instance, in a power distribution network from the example above, they represent the paths between electrical loads and power sources.

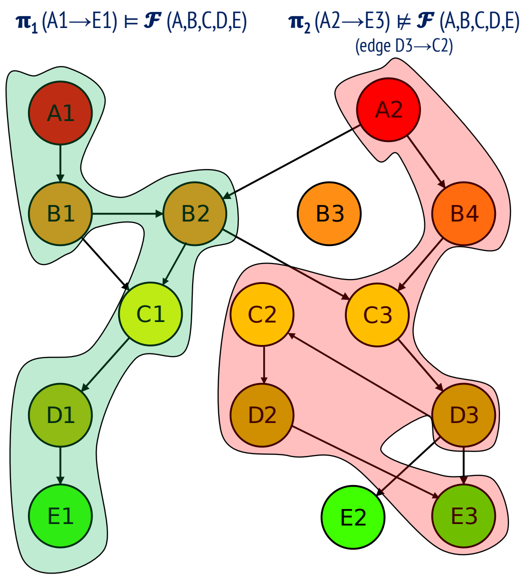
**Definition 2.7** (Functional Flow)**.** *A functional flow $\mathcal{F}$ is an ordered sequence of component types $(t_1, \ldots, t_n)$ that are needed to implement a functional link between a source and a sink. We also write $v_i \preceq_{\mathcal{F}} v_j$ if $t_{v_i}$ precedes $t_{v_j}$ in $\mathcal{F}$, and $v_i \sim_{\mathcal{F}} v_j$ if $v_i$ and $v_j$ have the same type.*

We say that a path $\pi$ complies with $\mathcal{F}$, written $\pi \models \mathcal{F}$, if the ordering of component types in $\pi$ does not violate the ordering in $\mathcal{F}$, i.e., $(v_i \sim v_{i+1}) \vee (v_i \preceq v_{i+1}) \quad \forall i \in \mathbb{N} : 1 \leq i \leq |\pi| - 1$. Multiple

[1]Using $w^{\pi^q}$ alone is not enough to compute a path $\pi^q$ (and $R^*$ in general), since it does not give any information about the ordering of nodes in $\pi^q$. At the same time, variables $y^{\pi^q}$ explicitly compute the edges of $\pi^q$, and the nodes can also be determined using this information. Therefore, node variables $w^{\pi^q}$ can be seen as auxiliary and, in general, redundant for defining $R$. However, as we show later in this thesis, they are extensively used for encoding application requirements, such as timing or power consumption. Therefore, it is useful to define them once in $R$, otherwise, the overall complexity of the encoding will increase.



(a) Functional link $F_{E2}$



(b) Functional flow $\mathcal{F}$ = (A,B,C,D,E)

Figure 2.4 – (a) Example of a functional link $F_{E2}$, i.e., a set of valid paths from sources (A1 and A2) to sink E2; (b) Example of a functional flow $\mathcal{F}$ = (A,B,C,D,E) and two paths: $\pi_1$ complies with $\mathcal{F}$ and $\pi_2$ violates $\mathcal{F}$ because of a “backward” connection from succeeding node D3 to preceding node C2.

functional flows $\{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$ can be defined, each representing an ordering of components for implementing a certain functionality within the same system. For instance, assuming that EPN generators from Example 4 provide AC power, AC and DC loads can have different sequences of intermediate components in their functional links (e.g., DC loads need rectifiers, while AC loads do not). In a manufacturing system different types of products can be processed differently (with different machinery etc). Every path $\pi$ belonging to some functional link $F$ of the system must comply with one of the functional flows.

**Example 5** (Functional Link)**.** *A functional link $F_{E2}$ connecting the sources A1 and A2 to the sink E2 is demonstrated on Figure 2.4a (covered by a yellow trace). It consists of three paths:* $(A1, B1, B2, C3, D3, E2)$, $(A2, B2, C3, D3, E2)$ *and* $(A2, B4, C3, D3, E2)$.

**Example 6** (Functional Flow)**.** *The functional flow $\mathcal{F}$ of the system is a sequence (A,B,C,D,E). On Figure 2.4b, path* $\pi_1 = (A1, B1, B2, C1, D1, E1)$ *complies with $\mathcal{F}$, while path* $\pi_2 = (A2, B4, C3, D3, C2, D2, E3)$ *violates it, because for the edge* $(D3, C2)$ *neither* $D3 \sim C2$ *nor* $D3 \preceq C2$ *holds.*

Informally, with the definitions above, the architecture exploration methodology deals with the selection of the system topology, including components interconnection and routing, and with sizing of each component, i.e., mapping it to a particular implementation from a domain-specific library. The resulting architecture must be feasible with respect to a set of constraints (rules). Some of the constraints come with the structure of the given formulation and ensure that it is consistent, while others are application requirements. In other words, all design problems based on the presented formulation, must satisfy the former, while the latter are specific to a particular domain and design. This allows us to define a common subset of variables and constraints as a fundament of the exploration problem. In the following sections, we discuss this subset and the way it is used for defining higher-level application requirements. In this work, we focus on the *optimized* exploration and selection of CPS architectures, and, therefore, the chapter is finalized with the problem statement as an optimization problem.

## 2.2 Basis of the Exploration Problem

In our formulation, requirements on the architecture are defined using the variables that represent the topology configuration ($e_{ij}$), the mapping ($m_{ij}$) and the paths ($y^{\pi}_{ij}$ and $w^{\pi}_{i}$). Together, they form the *basis B* of the exploration problem, i.e., a *common semantic domain*, in which we formalize an exploration problem for any type of system within our methodology. We group the basis variables into a set $B = E \cup M \cup R$. A set of constraints $\mathcal{R}_B$ that associates these variables to each other and ensures the overall consistency is also included in the basis. Below, we provide their mixed integer linear formulations.

### 2.2.1 Mapping Constraints

We denote as $M^k$ the mapping matrix for type $k$, where entries $m^k_{ij} = 1$ iff a virtual component (graph node) $v_j \in P_k$ is implemented by component $l^k_i \in \mathcal{L}_k$. $\mathcal{L}_k$ and $P_k$ are the subsets of $\mathcal{L}$ and $V$ including all the elements of type $k$ in $\mathcal{L}$ and $V$. We also recall that $E$ is the adjacency matrix of $\mathcal{T}$, i.e., its entries $e_{ij} = 1$ if there is a connection from node $v_i$ to node $v_j$, and 0 otherwise. Then, the mapping constraints for type $k$ assume the following form:

$$\bigvee_{i=1}^{|\mathcal{L}_k|} m^k_{ij} \geq (=) \bigvee_{i=1}^{|V|} (e_{ij} \vee e_{ji}) \qquad \forall j \in \mathbb{N} : 1 \leq j \leq |P_k|, \tag{2.1a}$$

$$\sum_{i=1}^{|\mathcal{L}_k|} m^k_{ij} \leq 1 \qquad \forall j \in \mathbb{N} : 1 \leq j \leq |P_k|. \tag{2.1b}$$

Constraints (2.1a) state that each component of type $k$ that is instantiated must be mapped to one of the components in $\mathcal{L}_k$. They can take one of the two forms: "soft" or "strict". While the former (with inequality) also allows the unconnected components to be associated with elements from $\mathcal{L}$, the latter (with equality) is a tighter version and allows only connected components to be mapped. In general, the strict constraint has to be used; soft mapping constraint is a form of relaxation that can be applied for systems, where the goal of the exploration problem is only to select the nodes. Example 7 illustrates both cases. An example of a design that uses soft mapping (a localization network) is introduced in Chapter 6. Note that disjunction (Boolean OR) operators in (2.1a) are nonlinear, since the result of the operation has to be binary, i.e., they cannot be replaced with summation. A standard linearization technique is applied to obtain linearity (see Section 2.4.1).

Constraints (2.1b) ensure that the mapping is nonambiguous, i.e., that virtual components are never mapped to more than one library component. Similar constraints are enforced for all the types in $\mathcal{T}$. Some components (e.g., of a certain type) can have *fixed* mapping, i.e., known a-priori. This can be done by explicitly setting the corresponding values from $M^k$ to 1 using equality constraints or replacing them with numeric values (ones).

**Example 7** (Mapping Constraints). *Figure 2.5 illustrates a topology configuration based on a template $\mathcal{T}$ with $n$ "virtual" components, which are mapped to a library $\mathcal{L}$ of $m$ elements, as described by an assignment over the mapping matrix $M$. Several graph nodes (A2,B3,B4,C2 and D3) and corresponding columns of $M$ demonstrate different mapping scenarios. Component A2 is mapped to a single element $l_2 \in \mathcal{L}$, while B4 is associated to both $l_1$ and $l_3$. The latter is an invalid mapping, and such cases are forbidden by Constraint* (2.1b). *Node B3 is not instantiated, i.e., does not have any mapping (and is not connected). C2 is mapped to $l_1$ even though it is not connected, which is an example of using a "soft" variation of Constraint* (2.1a). *Finally, D3 is connected, i.e., used in the architecture, but does not have a corresponding library element, which is also a violation that is prevented by* (2.1a).

Overall, the encoding approach in this thesis facilitates the exploration of different implementation alternatives. A change in $\mathcal{L}$ only affects the mapping constraints, which makes the presented formulation more general than the one in [99, 12], as the mapping constraints are not hard-coded as a part of the interconnection constraints. Moreover, this approach is more efficient. Let $\ell$ be the number of library options available to implement each component and $n$ be the number of nodes in the template $\mathcal{T}$. In previous formulations [99, 12], for solving an equivalent mapping problem, i.e., for selecting one of $\ell$ implementations for a node $v_i$, the latter has to be represented by $\ell$ nodes each representing a particular mapping decision. Therefore, the actual size of the adjacency matris $E$ of decision variables is $\ell n \times \ell n$, which is quadratic in $\ell$. In our approach, $E$ is an $n \times n$ matrix, while a mapping matrix $M_{\ell \times n}$ is also introduced, so the total number of variables amounts to $n^2 + n\ell = n(n+\ell)$, which is, instead, linear in $\ell$. Furthermore, as shown in the following chapters, many application constraints use edge variables $e_{ij} \in E$, so increasing the size of $E$ will highly affect their complexity[2] , while manipulating the size of $M$ has a much smaller impact. Experimental results in Section 5.1.5 confirm the efficiency of the proposed mapping approach.

### 2.2.2 Path Constraints

As discussed above, every path $\pi$ within the network of components is encoded using two sets of variables, $y^\pi$ and $w^\pi$, representing, respectively, edges and nodes used in $\pi$. To ensure a valid assignment over these variables, which does not contradict with the topology configuration (variables $e_{ij} \in E$), a set of generic path constraints is added to the formulation for every declared path $\pi$. In our approach, variables $y^\pi$ play the primary role, because they are sufficient to determine both the nodes of $\pi$ and their ordering. Given the source and

[2]Moreover, increasing the template size leads to a similar increase in the number of path variables $y^\pi_{ij}$ and $w^\pi_i$, so potential redundancy in $\mathcal{T}$ may be very harmful.

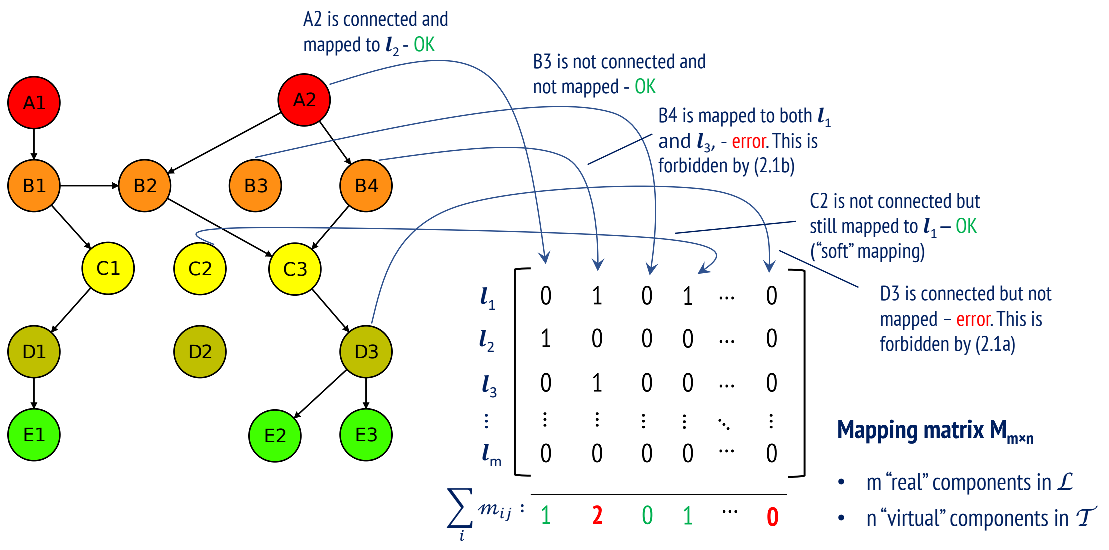

A2 is connected and mapped to $l_2$ - OK
B3 is not connected and not mapped - OK
B4 is mapped to both $l_1$ and $l_3$, - error. This is forbidden by (2.1b)
C2 is not connected but still mapped to $l_1$ – OK ("soft" mapping)
D3 is connected but not mapped – error. This is forbidden by (2.1a)
A1
A2
B1
B2
B3
B4
C1
C2
C3
D1
D2
D3
E1
E2
E3
$l_1$ 0 1 0 1 ⋯ 0
$l_2$ 1 0 0 0 ⋯ 0
$l_3$ 0 1 0 0 ⋯ 0
$l_m$ 0 0 0 0 ⋯ 0
$\sum_i m_{ij}$ : 1 2 0 1 ⋯ 0
Mapping matrix $M_{m \times n}$
m "real" components in $\mathcal{L}$
n "virtual" components in $\mathcal{T}$

Figure 2.5 – Examples of correct and invalid mapping scenarios.

destination (sink) nodes $v_s$ and $v_d$, $s$ and $d$ being the indices of the adjacency matrix $E$, we define following constraint for the path $\pi(v_s \to v_d)$:

$$C(y^\pi)^T = z^\pi, \tag{2.2}$$

which is a balance equation that ensures that $v_s$ and $v_d$ are connected by a path, i.e., edges from $y^\pi$ together form a direct walk from the source to the sink.[3] Here, $C$ is the incidence matrix of the template $\mathcal{T}$ and $z^\pi$ is a column vector of length $|V|$, with $z^\pi(s) = 1$, $z^\pi(d) = -1$, and the remaining values of $z^\pi$ are zero (see illustration from Example 8). To compute $C$ all possible edges in $\mathcal{T}$ are considered. Entries $c_{ij} \in C$ are numeric: $c_{ij} = 1(-1)$ if edge $j$ is leaving (entering) node $i$, and zero otherwise. $C$ can be precomputed offline for a given template $\mathcal{T}$ based on its structure and composition rules of $\mathcal{T}$ and library $\mathcal{L}$. In such a way, restricted connections are not added to $C$ and, therefore, constraint (2.2) ensures that generated paths are valid paths.

Next, we define the relations between the variables $e_{ij} \in E$ of the topology configuration, and path variables $y^\pi_{ij}$ and $w^\pi_i$ using the following expressions:

$$y^\pi_{ij} \le e_{ij} \qquad \forall i,j \in \mathbb{N} : 1 \le i,j \le |V|, \tag{2.3a}$$

$$w^\pi_i = \bigvee_{j=1}^{|V|} (y^\pi_{ij} \vee y^\pi_{ji}) \qquad \forall i \in \mathbb{N} : 1 \le i \le |V|. \tag{2.3b}$$

Constraint (2.3a) says that $y^\pi_{ij}$ is true only if nodes $i$ and $j$ share an edge. Constraint (2.3b) relates variables $w^\pi$ to $y^\pi$: node $i$ belongs to path $\pi$ if at least one of its incoming or outgoing

[3]Note that in Constraint (2.2) $y^\pi$ is a vector of path variables, i.e., their flattened representation, and every entry $y^\pi_k$ of the vector corresponds to a variable $y^\pi_{ij}$ for some source $v_i$ and some sink $v_j$

edges connects the nodes in $\pi$. Generally, both nodes and edges of $\pi$ can be determined by computing only $y^{\pi}$, as defined by Constraint (2.2), while node variables $w^{\pi}$ can be seen as auxiliary. However, many application requirements, such as timing, use the information about the components used in network paths, so node variables are extensively used. Therefore, it is useful to define them in the basis.

Finally, in the design methodology presented in this thesis we focus only on *simple* paths, i.e., paths that do not contain repeated vertices and loops. Assignments over $y^{\pi}$ generated using only the Constraint (2.2) may represent paths that contain loops (see Example 9). Therefore, we add an additional set of path constraints that forbid the loops in every path $\pi$:

$$e_{ii} = 0 \qquad \forall i \in \mathbb{N} : 1 \le i \le |V|, \tag{2.4a}$$

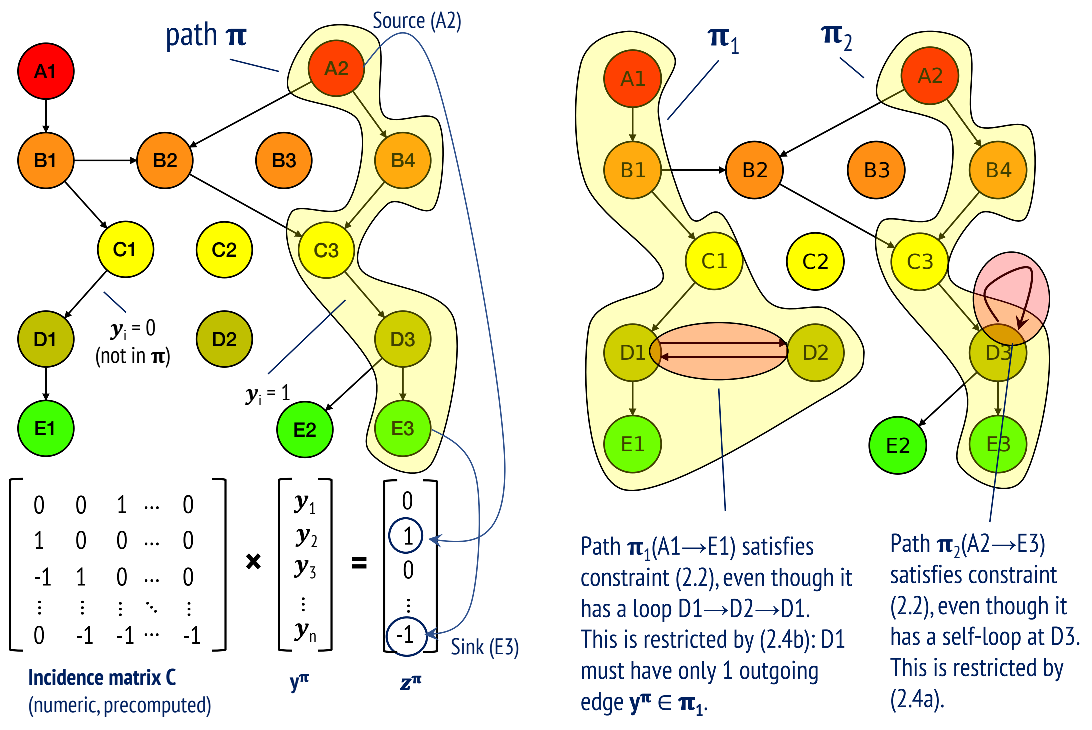
$$\sum_{j=1}^{|V|} y^{\pi}_{ij} \le 1, \quad \sum_{j=1}^{|V|} y^{\pi}_{ji} \le 1 \qquad \forall i \in \mathbb{N} : 1 \le i \le |V|. \tag{2.4b}$$

Constraint (2.4a) imposes that no node in $V$ is connected to itself (this applies to the whole topology configuration, hence, variables from $E$ are used), while two constraints from (2.4b) require that every node in $\pi$ has at most one predecessor and at most one successor. In general, a path that has a subtour of two or more nodes that are connected between each other and not connected to other nodes of $\pi$ is still valid in the scope of given constraints, i.e., certain kinds of loops cannot be restricted with the given formulation. However, in practice such cases are resolved by the virtue of satisfying the application constraints and minimizing number of nodes and edges used in the graph, which is typical for the cost function. There are alternative ILP formulations of simple paths, such as the one in [102], which tackle the issue above, but they lead to a significant increase of the problem complexity.

**Example 8** (Balance Equation)**.** *Using the balance equation* (2.2) *to generate a path* $\pi(A2 \to E3)$ *is demonstrated on Figure 2.6a. Column vector* $z^{\pi}$ *has only two non-zero values, 1 and -1, corresponding to, respectively, source A2 and sink E3 of* $\pi$. *A satisfying assignment over* $y^{\pi}$ *instantiates following edges for* $\pi$*: (A2,B4), (B4,C3), (C3,D3) and (D3,E3), which allows to determine both the nodes and the order, in which they appear in* $\pi$.

**Example 9** (Loop Avoidance)**.** *Figure 2.6b illustrates two paths,* $\pi_1(A1 \to E1)$ *and* $\pi_2(A2 \to E3)$ *that satisfy the Constraint* (2.2)*, but have loops. In particular,* $\pi_1$ *has a bidirectional connection between nodes D1 and D2, which results in a loop* $D1 \to D2 \to D1$. *Note that D1 in this case has two incoming and two outgoing edges in* $y^{\pi}$. *Such paths can be restricted by using Constraint* (2.4b)*. In turn, path* $\pi_2$ *has a self loop at D3, i.e., an edge, connecting D3 to itself. Using Constraint* (2.4a) *allows the self-loops to be avoided in all paths and in the overall topology configuration.*

We note that the presented path constraints enumerate all possible paths between a source-destination pair, which has exponential complexity. This is further exacerbated when the enumeration has to be done for all possible pairs, because listing them has exponential complexity as well. To cope with the latter issue, the paths in our formulation are declared on


path π
Source (A2)
A1
A2
B1
B2
B3
B4
C1
C2
C3
D1
D2
D3
E1
E2
E3
$y_i = 0$ (not in π)
$y_i = 1$
Incidence matrix C (numeric, precomputed)
$y^\pi$
$z^\pi$
Sink (E3)
$\pi_1$
$\pi_2$
Path $\pi_1$(A1⟶E1) satisfies constraint (2.2), even though it has a loop D1⟶D2⟶D1. This is restricted by (2.4b): D1 must have only 1 outgoing edge $y^\pi \in \pi_1$.
Path $\pi_2$(A2⟶E3) satisfies constraint (2.2), even though it has a self-loop at D3. This is restricted by (2.4a).

$$\begin{bmatrix} 0 & 0 & 1 & \cdots & 0 \\ 1 & 0 & 0 & \cdots & 0 \\ -1 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -1 & -1 & \cdots & -1 \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ -1 \end{bmatrix}$$

(a) Loopless path $\pi$

(b) Paths $\pi_1$ and $\pi_2$ with loops

Figure 2.6 – Path constraints: (a) Using balance equation $C(y^\pi)^T = z^\pi$ for generating paths; (b) Examples of paths with loops satisfying the balance equation.

the need basis, i.e., only between components that must be connected due to some application requirements (e.g., routing). We also address the former problem by proposing an approximate encoding of network paths as discussed in Chapter 3.5.1.

## 2.3 Using the Basis

We have presented the basis of the exploration problem as a set $B$ of decision variables, and a set $\mathcal{R}_B$ of integer linear constraints. An assignment over $B$ that satisfies $\mathcal{R}_B$ provides the necessary and, in some cases, sufficient information about the system architecture selected from the design space: how the components are used, how they are connected (direct links and paths), and how they are implemented. In this section we demonstrate how to use the variables from $B$ as building blocks for defining the properties and requirements of cyber-physical systems.

### 2.3.1 Associating Template and Library Attributes

Both nodes and edges of the architecture template $\mathcal{T}$ can be labeled with attributes corresponding to those from the library $\mathcal{L}$, for example, latency, failure probability or power capacity. Mapping constraints from $\mathcal{R}_B$, discussed in Section 2.2.1, enforce that every node $v_j \in \mathcal{T}$ is mapped to at most one element $l_i \in \mathcal{L}$. However, the actual mapping will be selected as a result of the optimization process, and, therefore, it is not known a-priori. At the same time, most of architecture requirements (apart from $\mathcal{R}_B$) are defined over the attributes of $\mathcal{T}$ and naturally depend on the mapping. Hence, these attributes have to be expressed in a form that takes all possible mappings into account. This can be done by leveraging the mapping variables $m_{ij} \in M$ from the basis. Let the nodes $v \in V$ of $\mathcal{T}$ be labeled with an attribute $A$ (e.g., delay) and let $A_j^{\mathcal{T}}$ be the value of this attribute for node $v_j$. Then we can write:

$$A_j^{\mathcal{T}} = \sum_{i=1}^{|\mathcal{L}|} m_{ij} A_i^{\mathcal{L}} \qquad \forall j \in \mathbb{N} : 1 \le j \le |V|, \tag{2.5}$$

where $A_i^{\mathcal{L}}$ is the vector of values of the attribute $A$ for the components in $\mathcal{L}$. In the sum (2.5), at most one of the Boolean variables $m_{ij}$ will evaluate to true, thus making the value of $A_j^{\mathcal{T}}$ equal to a corresponding value from $A^{\mathcal{L}}$ and forcing all other terms to zero. If the component is not instantiated, i.e., does not have a mapping, then the value of $A_j^{\mathcal{T}}$ will be zero. The same can be applied to graph edges.

Overall, when an attribute of a node (or an edge) is used for formulating a constraint or a cost function, it can be substituted with Expression (2.5). Alternatively, every new label can be defined as a decision variable, and then (2.5) can be used as a constraint that associates its value to $\mathcal{L}$. More generally, each attribute in $\mathcal{T}$ can be a linear combination of other existing attributes, each of them being associated to a corresponding element (and value) from the library using the basis variables $m_{ij}$ as shown above.

### 2.3.2 Formalizing Application Requirements on Top of the Basis

In general, a requirement on the architecture can be expressed as a bound on a certain property (e.g, end-to-end delay, failure probability of a link, power provided by a set of generators etc). In turn, many system properties can be defined as functions $f(.)$ over a set of attributes that are assigned to components and connections of the architecture (graph nodes and edges in our formulation). Properties can be system-wide or related to a particular scope, such as a path or a subsystem. Of course, components that are not used in the system or within the selected scope must not be taken into account when calculating a property. This can be determined using the basis variables. Also, as discussed in Section 2.3.1, variables $m_{ij}$ from $B$ are used to associate the attributes $A^{\mathcal{T}}$ of the template to those from the library ($A^{\mathcal{L}}$). Overall, variables $\{e_{ij}, m_{ij}, y_{ij}^{\pi}, w_i^{\pi}\} \in B$ define both the mapping of "virtual" components to implementations, i.e., the values of $A^{\mathcal{T}}$, and their presence or absence in the architecture (or its part). A generic design constraint, therefore, can be expressed in the following form:

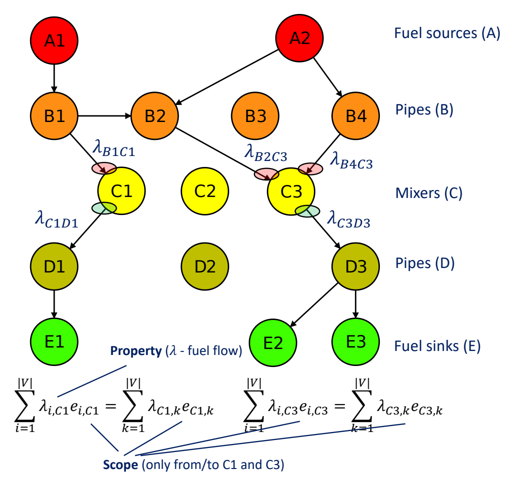
$$f\left(\mathcal{A}^{\mathcal{T}}, \beta, x_A, x_D, x_C\right) \ \ge (\le, =) \ Y^*, \tag{2.6}$$

where $\mathcal{A}^{\mathcal{T}}$ includes a set of attributes $\{A_1^{\mathcal{T}}, \ldots, A_n^{\mathcal{T}}\}$ of the template $\mathcal{T}$ that are used to compute the property $f(.)$, $\beta \in B$ are the basis variables that define the scope of the property, $x_A \in X_A$ are auxiliary variables, e.g., added to the formulation for linearizing complex constraints (further explained in Section 2.4), $x_D \in X_D$ and $x_C \in X_C$ are, respectively, discrete and continuous parameters that are problem-specific, and the right-hand side $Y^*$ is the numeric bound (lower, upper or exact). $X_C$ and $X_D$ include decision variables, which are either system-level parameters or template attributes that do not have corresponding ones in the library. For example, $x_C$ can be a duty cycle of a wireless communication protocol (i.e., a system parameter) or a flow of some continuous-valued quantity (e.g., product flow, air flow) between a pair of components. Similarly, $x_D$ can represent some discrete (binary, integer) choice, such as a number of message retransmissions, a scheduling policy and others.

**Example 10** (Flow Balance Constraint)**.** *Assume Figure 2.7a represents the architecture of a fuel distribution system, where components C1 and C3 are mixers that distribute the input flows to output flows (both inputs and outputs can be multiple). Balance constraints for C1 and C3 state that their fan-in must be equal to fan-out, i.e., no fuel is stored in the components over time. Edges of the graph on Figure 2.7a are labeled with domain-specific attributes $\lambda_{ij}$ (fuel flows), which are real decision variables, so that a static fuel flow distribution is obtained as a part of the solution. To consider only those edges that are used in the topology configuration, flow rates $\lambda_{ij}$ are multiplied by corresponding variables $e_{ij} \in E$. We can express the balance constraint as $f(\lambda, e, x_A) = \sum_{i=1}^{|V|} \lambda_{ij} e_{ij} - \sum_{k=1}^{|V|} \lambda_{jk} e_{jk} = 0$, where $Y^* = 0$, $j = \{idx(C1), idx(C3)\}$ (indices of nodes C1 and C3 in E), $\mathcal{A}^{\mathcal{T}} = x_C = \lambda$, $\beta = e$ (only topology configuration variables from the basis are used), and $x_A \in X_A$ are auxiliary variables added to the formulation for linearizing the products $\lambda_{ij} e_{ij}$ of real and binary variables.*

**Example 11** (Timing Constraint)**.** *In a communication network, end-to-end delay $\tau$ of a path $\pi(A1 \rightarrow E1)$ for delivering a message from A1 to E1 must not exceed the threshold $\tau^*$. To correctly calculate $\tau$ as a sum of delays $\tau_i$ of components along the path, other components (not in $\pi$) must not be considered. This is done by multiplying each $\tau_i$, which are attributes of nodes $v_i \in V$ of the template $\mathcal{T}$, by corresponding basis variables $w_i^\pi$ and summing them together, so that only the delays of components from $\pi$ are considered (rest are zeros). Therefore, by using the variables $w_i^\pi$ from the basis the property (timing) is calculated within a particular scope (path $\pi$). Note that products $w_i^\pi \tau_i$ are nonlinear, hence, linearization is applied, and auxiliary variables $x_A \in X_A$ are added to the formulation. The constraint can be written as $f\left(\tau, w^\pi, m, x_A\right) = \sum_{i=1}^{|V|} \tau_i w_i^\pi \leq \tau^*$, where $\mathcal{A}^{\mathcal{T}} = \tau$, $\beta = \{w^\pi, m\}$ (mapping variables $m \in M$ are used to compute the values of $\tau_i$ of the template), and domain-specific parameters $x_C$ and $x_D$ are not used. This is further explained on the Figure 2.7b, as well as in Section 3.2.5.*

Overall, the presented generic basis allows us to express a variety design of requirements for different classes of CPS. As further shown in Chapter 3, all definitions of the application constraints make use of the variables from $B$. For some of them, auxiliary and domain-specific variables $(x_A, x_C, x_D)$ are used, which also complies with the generic constraint expression (2.6). Finally, all application constraints and auxiliary constraints (e.g., those used for linearization) form the set $\mathcal{R}_A$, i.e., a set of constraints for a specific architecture exploration problem.


(a) Flow balance constraint

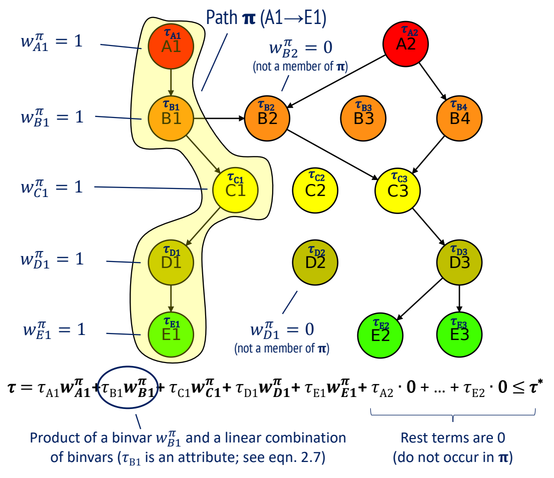

(b) Timing constraint

Figure 2.7 – Examples of defining application constraints $\mathcal{R}_A$ using variables from the basis of the exploration problem: (a) Flow balance constraint for mixer components in a fuel distribution system; (b) End-to-end delay (timing) constraint for a communication network.

### 2.3.3 Objective Functions

Every node and every edge in $\mathcal{T}$ is associated with a cost value. This may represent the monetary cost as well as other cost parameters, such as idle time, energy, weight. We then consider cost functions that can be expressed as the sum of the costs of all the instantiated components (nodes) and connections (edges):

$$\sum_{i=1}^{|V|} \delta_i c_i + \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} e_{ij} \tilde{c}_{ij}, \tag{2.7}$$

where $c_i$ is the cost of component $v_i$, $\tilde{c}_{ij}$ is the cost of the edge $e_{ij}$, and $\delta_i$ is a binary variable equal to one if the component is instantiated and zero otherwise.

To compute certain costs, such as energy consumption, the information about an edge $e_{ij}$ being present in the topology configuration is not sufficient, because it is important to know how many paths (routes) in the system use this edge (link). Same applies to the components. Therefore, another cost function that we consider is using the path variables $y^\pi$ and $w^\pi$ from the basis:

$$\sum_{\pi} \left( \sum_{i=1}^{|V|} w_i^\pi c_i + \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} y_{ij}^\pi \tilde{c}_{ij} \right) \tag{2.8}$$

Typically, either (2.7) or (2.8) is used to formulate the overall objective. Depending on the specific problem, some of the terms in these expressions may be omitted or assigned weights (for example, total costs of nodes and edges may have different impact). Also, in general, a combination of (2.7) and (2.8) can be used. For example, if the cost of nodes is constant and depends only on their presence or absence in the topology, while every edge can contribute several times depending on how it is used, then the final cost is composed from the first part of (2.7) and second part of (2.8).

Finally, we note that the cost functions presented above are expressed using the basis variables, but, as discussed in Section 2.5, the underlying expressions for node and edge costs $c_i$ and $\tilde{c}_{ij}$ may also depend on some non-basis variables that are specific to the exploration problem.

## 2.4 Linearization

Our methodology leverages mixed integer linear programming to solve architecture exploration problems. Therefore, all constraints and cost functions in the formulation must be linear. That is, functions $f(.)$ that compute system properties must consist of linear operations over their arguments, which are decision variables. If a property cannot be expressed in linear form per se, then some linearization method has to be applied. In general, such methods replace the original nonlinear expression with a set of auxiliary variables and MILP constraints that associate these variables to results of original nonlinear operations. The variables are added to the set $X_A$ mentioned above.

We note that the techniques presented in this section do not cover all possible cases, in which the linearization may be required. However, situations, in which these techniques can be applied, frequently occur in definitions of application requirements within the methodology presented in this thesis. Hereafter, if one of them (e.g., product of variables) appears in the constraint formulation, we refer to this section instead of listing the linearization variables and constraints for every nonlinear expression.

### 2.4.1 Standard Techniques

Linear operations with decision variables only include summation and multiplication by a constant. That is, a significant part of operations used to express optimization constraints (even simple ones) is in fact nonlinear. Such operations are also extensively used in our methodology for expressing a variety of application requirements and include, for instance, products of decision variables, logical operations (AND, OR, NOT, XOR etc), absolute values and so on. Most of these can be linearized using a set of well-known techniques [126]. Below we summarize the ones that are frequently used throughout the thesis:

- **Logical AND** (product of Boolean variables $x_1$ and $x_2$, conjunction, lower bound, `Min`): the product $x_1 x_2$ is replaced with an auxiliary Boolean variable $y$. Following constraints are added to associate $y$ to the operands: $y \geq x_1 + x_2 - 1$, $y \leq x_1$, $y \leq x_2$. This can also be generalized for any number of operands $x_1 \dots x_n$: $y \leq x_i \ \forall i \in \mathbb{N} : 1 \leq i \leq n$, $y \geq \sum_i x_i - (n-1)$.
- **Logical OR** (disjunction, upper bound, `Max`): $x_1 \vee x_2$ is replaced with a Boolean variable $y$, and constraints $y \leq x_1 + x_2$, $y \geq x_1$, $y \geq x_2$ are added. Can be generalized to disjunction of $n$ Boolean variables $x_1 \dots x_n$: $y \geq x_i \ \forall i \in \mathbb{N} : 1 \leq i \leq n$, $y \leq \sum_i x_i$.

- **Logical NOT** (negation): $\neg x$ can be written simply as $1 - x$.
- **Logical implication** (if-then): if $x_1$ and $x_2$ are atomic Boolean variables, then $x_1 \Rightarrow x_2$ can be simply rewritten as $x_1 \leq x_2$. Otherwise, an auxiliary Boolean variable $y$ can be introduced, such that $y = x_1 \rightarrow x_2 = \neg x_1 \vee x_2$, and linear constraints $y \leq 1 - x_1 + x_2$, $y \geq 1 - x_1$, $y \geq x_2$ are added.
- **Logical XOR** (exclusive OR): to express $x_1 \oplus x_2$, one can replace the operation with $y$ (Boolean) and constraints $y \leq x_1 + x_2$, $y \geq x_1 - x_2$, $y \geq x_2 - x_1$, $y \leq 2 - x_1 - x_2$.
- **Product of real and Boolean variables**: let $z$ be a real decision variable, such that $0 \leq z \leq U$, and $x$ be a Boolean variable. Then $z \cdot x$ can be replaced with a real variable $y$, such that $y \leq Ux$, $y \leq z$, $y \geq z - (1-x)U$ and $y \geq 0$. If the upper bound $U$ is unknown, then some big number can be used instead (this is also called the “big-M” approach). If the lower bound is less than zero, than a more complex formulation is also available [126].

### 2.4.2 Approximate Encodings

Some complex properties (e.g., bit error rate of a wireless link, advanced delay models) cannot be expressed as linear functions over component attributes neither by itself nor by using the techniques above. Common examples include division by a real variable as well as power and exponential functions. Some of them can be expressed as linear if they are used on a small interval (locally), while for some others efficient, yet approximate, linear encodings have been proposed (e.g., [12]). In this work, we use a simple approximation of nonlinear functions that can be used whenever none of the aforementioned can be applied.

The main idea of the approximation is to discretize the nonlinear function to be computed. That is, instead of computing the function in its exact formulation, it can be replaced with as set of intervals of its arguments, each resulting in a particular value of the function. These intervals compose a lookup table, which can then be encoded into MILP using a set of implication (if-then) constraints. Let a system property be represented by a nonlinear function $f(x)$ of a real variable $x$ on an interval $L \leq x \leq U$. We can then discretize $f(x)$ to a set of $n$ intervals $\Delta_i$, such that $\Delta_i \leq \frac{(U-L)}{n}$ $\forall i \in \mathbb{N} : 1 \leq i \leq n$ (the last interval $\Delta_n$ may be a remainder, i.e., smaller than the rest; we also add $\Delta_0 = 0$). Then, the approximate encoding of $f(x)$ will assume the following form:

$$(L + \Delta_{i-1}(i-1) + \varepsilon \leq x \leq L + \Delta_{i-1}(i-1) + \Delta_i) \Rightarrow (y = Y_i) \qquad \forall i \in \mathbb{N} : 1 \leq i \leq n, \tag{2.9a}$$

$$y_{min} \leq y \leq y_{max} \qquad \forall y \in Y, \tag{2.9b}$$

where $Y_i$ is the value of $f(x)$ associated with the interval $i$, $y_{min}$ and $y_{max}$ are the numerical bounds for the real variable $y$, and $\varepsilon$ is a small value. Every possible value of $x$ on the specified interval $[L, U]$ will activate one of the implications (2.9a) and force $y$ to take a corresponding value $Y_i$ from the lookup table. The intervals have to be disjoint, otherwise, since strict inequalities cannot be used in MILP, the values of $x$ that are equal to the beginning or the end of some interval may activate two constraints simultaneously. Consequently, $y$ will be constrained to take two different values at the same time leading to infeasibility of the formulation. We use a small value $\varepsilon$ to make two neighboring intervals disjoint, so that none of the values of $x$ can cause such inconsistency. Uncertain situation can still occur if $L+\Delta_{i-1}(i-1) \le x \le \varepsilon$, which does not fall under any of the implications (2.9a). In such case, $y$ is allowed to take any value within the bounds, specified by Constraint (2.9b). However, the value of $\varepsilon$ can be selected based on the domain-specific knowledge to minimize the probabiliy of such uncertainties.

Intuitively, the degree of linear approximation of $f(x)$ using the method above can be altered by changing the number of intervals $n$ and, consequently, the number of constraints. Larger values of $n$ lead to a more accurate representation of the function and may be necessary especially if its behavior is not monotonic. At the same time, this also affects the problem complexity due to additional constraints. Moreover, implications (2.9a) also require linearization, so even more constraints will be added for every interval. Furthermore, if $x$ represents some attribute of a node $v_i$ or edge $e_{ij}$, then, in general, the computation of $f(x)$ has to be performed for every node or edge from the template $\mathcal{T}$, so the size of $\mathcal{T}$ also highly affects the complexity introduced by our approximation.

Finally, we note that the presented encoding of the linear approximation using a lookup table can be generalized so that $x$ is some expression in existing decision variables, and intervals $\Delta_i$ have different lengths. Moreover, instead of being numeric, values $\{Y_1 \dots Y_n\}$ can represent *piecewise linear functions*. Our approach is a simplified special case of the latter. It has an advantage of being generic and applicable to a variety of functions and related system properties, and a drawback of potential accuracy loss. It proposes a tradeoff, since the user can make the approximation more accurate for a price of increased complexity of the formulation.

## 2.5 Problem Statement

Within the computational framework presented in this chapter, we have outlined the generic constituents of the CPS architecture: topology configuration, mapping and routing. In addition, system architecture may include sets $X_C$ and $X_D$ of, respectively, continuous and discrete parameters, which are problem-specific. By putting it all together, we cast the cyber-physical system architecture exploration problem as an optimization problem as follows:

- **Given** a template $\mathcal{T}=(V,E)$ and a library $\mathcal{L}$ of components
- **Find** an assignment $B^*$ over the basis variables (topology configuration $E^*$, a map $M^*$ and a routing $R^*$), and the values $X_C^*$ and $X_D^*$ of problem-specific parameters
- **Such that** all application constraints $\mathcal{R}_A$ and basis constraints $\mathcal{R}_B$ are satisfied.

More formally:

$$\begin{aligned} \min_{\beta \in B,\, x_D \in X_D,\, x_C \in X_C} \quad & \mathcal{C}(\beta, x_D, x_C) & \\ \text{s.t.} \quad & r_B(\beta) \le 0 & \forall r_B \in \mathcal{R}_B, \\ & r_A(\beta, x_D, x_C) \le 0 & \forall r_A \in \mathcal{R}_A \end{aligned} \tag{2.10}$$

In (2.10), cost function $\mathcal{C}$ is also defined both over the basis and problem-specific variables, using the expressions (2.7) or (2.8), where variables $x_c \in X_c$ and $x_d \in X_d$ can be involved in calculating node costs $c_i$ and edge costs $\tilde{c}_{ij}$. For instance, if $\mathcal{C}$ is the battery lifetime of a wireless sensor node, then every message that the node is sending or receiving, i.e., every incoming and outgoing link to/from the node, contributes to draining the battery. The number of these messages over a period of time depends on the duty cycle of the MAC protocol, which is a real decision variable from $X_C$ specific to the problem of wireless network topology selection.

In general, $\mathcal{C}$ can include several independent objectives, so that the provided formulation becomes multiobjective. The optimization-based methodology presented in this thesis, however, leverages mixed integer linear programming techniques. Typically, a single cost function is used in MILP, while multiple objectives can still be addressed by collapsing them to a single function as a weighted sum of different concerns. In the latter case, designers are able to explore the tradeoffs by assigning and manipulating the weights. Finally, we note that all $r_A \in \mathcal{R}_A$, $r_b \in \mathcal{R}_B$, and $\mathcal{C}$ must be *linear* in their arguments.

The decision variable set of the exploration problem is $D = (E \cup R \cup M) \cup (X_A \cup X_C \cup X_D) = B \cup X$, where $X$ includes the auxiliary and application-specific variables. The final assignment over $D$, determined as a result of the optimization process, provides an optimal architecture, i.e., a network topology, in which a subset of the nodes and edges in $\mathcal{T}$ is used, and the mapping of nodes to components in $\mathcal{L}$. It also includes the parameter values, specific to a given system.

For certain classes of systems, such as communication networks, routing $R^*$ is an inherent part of the design (e.g., configuration of the routing protocol). An example of wireless network design with routing requirements is provided in Chapter 6. Some other systems, such as an electrical power system, may require that the paths between certain components exist (e.g., for reliability purposes), while the exact structure of these paths, i.e., path variables and, consequently, routing, may not be of primary importance. In such cases, to lower the complexity, the goal of the exploration problem can be reduced to finding only $E^*$ and $M^*$ in the basis, and $X$. Also, some architectures can be fully defined using only the presented basis, and in such cases $X = X_A$ (e.g., if some constraints require linearization), or even an empty set.

# 3 Exploration Methodology

*In this chapter we present the methodology for architecture exploration to select correct-by-construction configuration and interconnection of system components taken from pre-defined libraries. We leverage an extensible set of requirement patterns for creating the system specification. We then translate this specification to a set of mixed integer linear constraints. Using the basis of the exploration problem introduced in Chapter 2, we are able to instantiate a variety of design requirements, such as connectivity, balance, timing, reliability and energy consumption. We then discuss the mapping problem, which is separated from the topology selection problem. Interconnection (topology) and implementation (mapping) of the architecture are then jointly optimized using either monolithic or iterative optimization techniques. We also present two algorithms that allow the methodology to scale to large designs. The first one is an approximate technique for encoding generic path constraints, which drastically reduces the problem complexity by guiding the solver to a promising part of the design space. The second algorithm is an iterative approach for solving optimization problems.*

## 3.1 Overview

In Chapter 2 we have established a computational framework, which leverages a graph-based representation of a CPS architecture and defines the exploration process as an optimized selection of system components from a library and system topology (connections and routes between components) from a template. We have also outlined the generic basis of the exploration problem by encoding both the topology and the mapping into a mixed integer linear program. The goal of this chapter is to develop a CPS architecture exploration methodology within the proposed framework and on the top of the defined basis.

Conceptual representation of the methodology is shown on Figure 3.1. Following the principles of platform-based design [114], we separate the *topology selection problem*, i.e., whether a "virtual" component should be used in the architecture and how it is connected to others, from the *mapping problem*, i.e., which library component best implements the "virtual" one. The former requires the exploration of the *application* (functional) space that incorporates design requirements defined over the nodes, edges and paths of the architecture template $\mathcal{T}$. The latter is related to the selection from the *implementation* space, i.e., from a library $\mathcal{L}$ of available "real" components and connections that captures their related characteristics and provides design

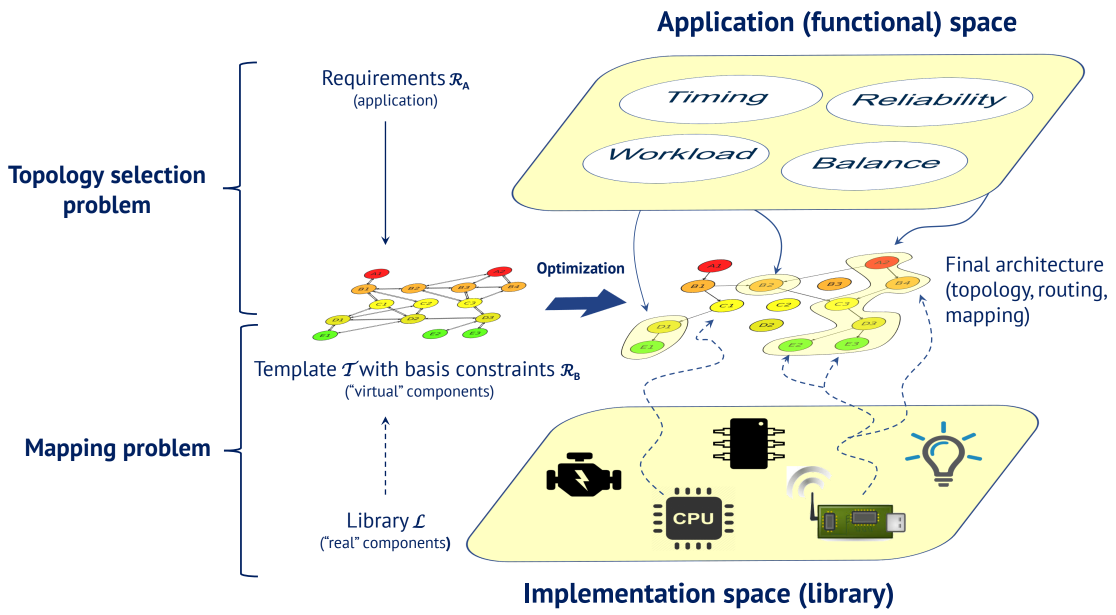

Application (functional) space
Timing
Reliability
Workload
Balance
Topology selection problem
Requirements $\mathcal{R}_A$ (application)
Optimization
Final architecture (topology, routing, mapping)
Template $\mathcal{T}$ with basis constraints $\mathcal{R}_B$ ("virtual" components)
Mapping problem
Library $\mathcal{L}$ ("real" components)
CPU
Implementation space (library)

Figure 3.1 – Separation of concerns within the proposed architecture exploration methodology: *topology selection* problem (how many components to use and how to connect them) is decoupled from the *mapping problem* (which elements from the library implement the topology in a best way).

alternatives. The final architecture is a meet-in-the-middle of the two spaces and problems, which is optimized with respect to a cost function. We establish the dependency between the two problems by defining them within the same formal basis by using the variables and constraints discussed in Sections 2.2 and 2.3. This results in a mathematical formulation, in which the decisions made in both spaces affect each other and the design constraints. We use this formulation to cast an optimization problem for jointly selecting the system topology and its implementation, and provide a cost-effective and correct-by-construction solution.

In this work, we focus on steady-state, discrete abstractions of CPS, i.e., we select the system architecture by evaluating the static properties, and abstract the dynamic behavior. However, as discussed in Section 1.4.2, our approach is complementary and its results can be used on further stages of design space exploration, such as simulation, to evaluate the system dynamics. We have chosen mixed integer linear programming as an optimization engine because its expressiveness allows us to capture a variety of design concerns (as we show later in this chapter), while state-of-the-art MILP solvers, such as CPLEX [54] or Gurobi [48] are able to efficiently explore the search space. We have extended existing formulations [12, 99, 108] with support for component sizing (library mapping) and continuous-based decision variables. Moreover, we improve the usability of MILP-based techniques by introducing a pattern-based formal language that facilitates the writing of the design specification.

Figure 3.2 illustrates the flow of the proposed methodology, which has four main steps. First of them is the *design specification*, which includes the general organization of the system (types and maximum number of components, composition rules, functional flows) and application requirements. Library $\mathcal{L}$ is also initialized as a collection of elements of corresponding types, which represent real components that can be implementation alternatives for the architecture. The specification is created using patterns, which are short expressions that are intuitively associated with corresponding requirements and automatically translate them to MILP constraints. We further discuss patterns in Chapter 4, where we introduce ARCHEX, a tool that supports our methodology, and demonstrate their effectiveness in Chapters 5 and 6 of this dissertation.

Specification part is followed by *encoding* of the specification into a mixed integer linear program and formulation of the optimization problem. First, the generic basis introduced in Section 2.2 is initialized, i.e., the template $\mathcal{T}$ is created and its elements are associated with the ones from library $\mathcal{L}$ using mapping constraints. Also, system routes required in the specification are defined using the path variables and constraints from the basis. Depending on the problem, each path can be symbolically encoded either as a full enumeration of all possible sequences of nodes in $\mathcal{T}$ or using an approximate algorithm that is introduced in Section 3.5.1. The basis is then used to translate requirement patterns used in the specification to corresponding categories of MILP constraints. As discussed in Section 2.3.2, application requirements are defined on system properties (e.g., latency, reliability) within a particular scope (e.g., a link, a route, a subsystem of components). Attributes (e.g., delay or failure probability of a component), which form the properties, as well as the scope of the latter are computed using basis variables. Properties themselves can also include domain-specific and auxiliary variables as a part of their expressions. For some of the properties, such as reliability, approximate algorithms are also applied to obtain a MILP formulation. We demonstrate the encoding of a variety of CPS requirements in Section 3.2.

Alltogether, set of basis constraints $\mathcal{R}_B$, set of application constraints $\mathcal{R}_A$ and cost function $\mathcal{C}$ compose the optimization problem, which is passed to the next, *solving* stage of the methodology. Here, we use one of the two methods. The *eager* optimization method solves a *monolithic* problem that includes all the optimization constraints. In contrast, the *lazy* method leverages a coordination of the specialized solvers. In this paradigm, the MILP solver is called *iteratively* on smaller problem instances including only a subset of constraints (e.g., interconnection constraints) to generate a candidate architecture, and then heuristic functions are applied for analyzing this architecture and learning new constraints if the latter does not yet meet the requirements. Both approaches are further discussed in Section 3.4.

Finally, during the *analysis* stage, generated architectures are evaluated with respect to certain properties. For example, reliability analysis verifies the functional links of the system to determine for every sink the probability of being disconnected from all sources (e.g., the probability of an electrical load on an aircraft to get unpowered). Static timing analysis (STA) algorithm traverses the architecture graph to compute end-to-end delay of certain paths. Workload analysis can be applied for estimating the steady-state load at the processing nodes of the system as well as calculating their idle rates.

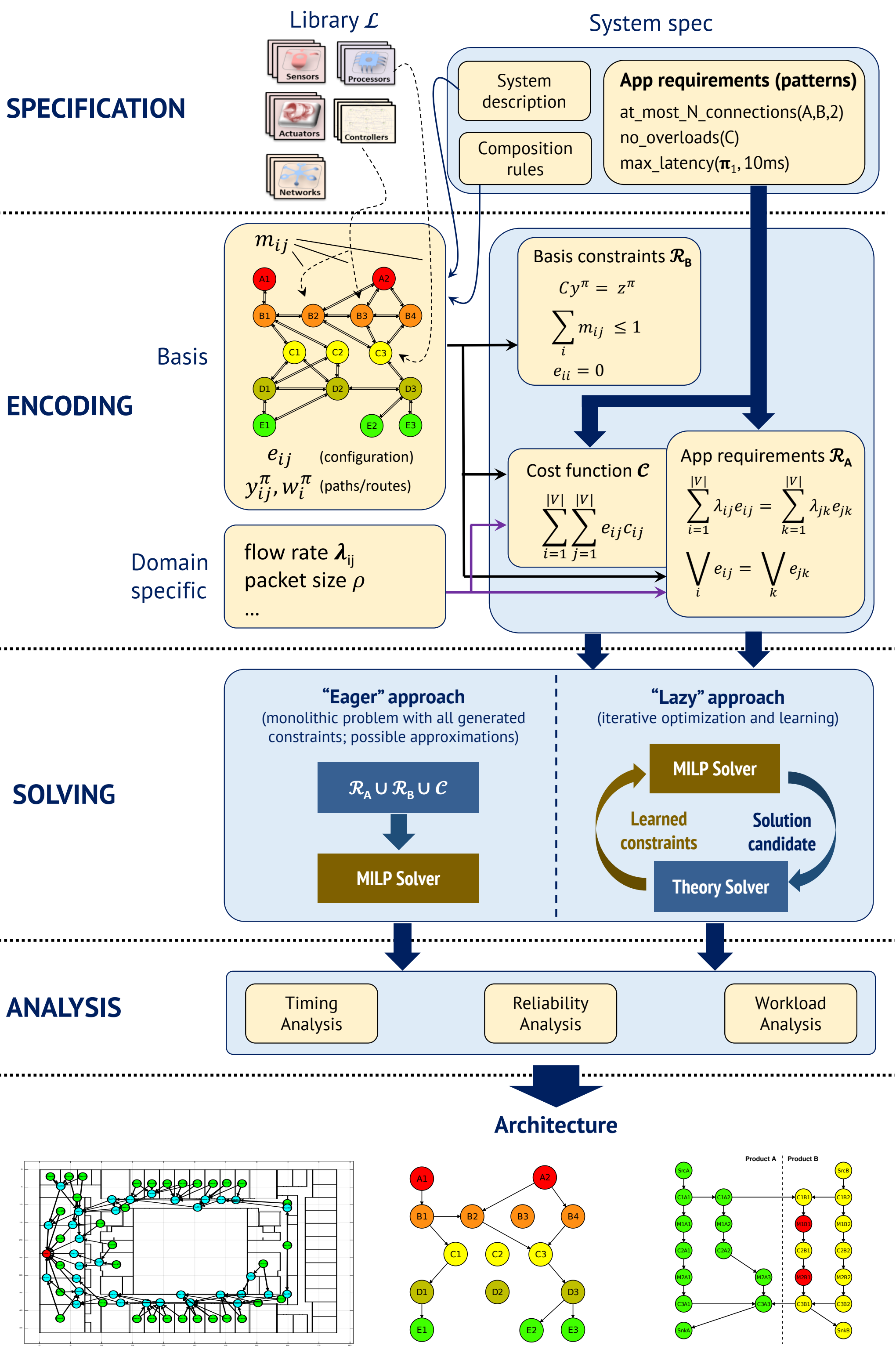

Library $\mathcal{L}$
Sensors
Processors
Actuators
Controllers
Networks
System spec
System description
Composition rules
App requirements (patterns)
at_most_N_connections(A,B,2)
no_overloads(C)
max_latency($\pi_1$, 10ms)
SPECIFICATION
ENCODING
Basis
$m_{ij}$
$e_{ij}$ (configuration)
$y_{ij}^{\pi}, w_i^{\pi}$ (paths/routes)
Domain specific
flow rate $\lambda_{ij}$
packet size $\rho$
...
Basis constraints $\mathcal{R}_B$
$C y^{\pi} = z^{\pi}$
$\sum_i m_{ij} \leq 1$
$e_{ii} = 0$
Cost function $\mathcal{C}$
$\sum_{i=1}^{|V|}\sum_{j=1}^{|V|} e_{ij}c_{ij}$
App requirements $\mathcal{R}_A$
$\sum_{i=1}^{|V|} \lambda_{ij}e_{ij} = \sum_{k=1}^{|V|} \lambda_{jk}e_{jk}$
$\bigvee_i e_{ij} = \bigvee_k e_{jk}$
SOLVING
"Eager" approach
(monolithic problem with all generated constraints; possible approximations)
$\mathcal{R}_A \cup \mathcal{R}_B \cup \mathcal{C}$
MILP Solver
"Lazy" approach
(iterative optimization and learning)
MILP Solver
Learned constraints
Solution candidate
Theory Solver
ANALYSIS
Timing Analysis
Reliability Analysis
Workload Analysis
Architecture
Product A
Product B

Figure 3.2 – Flow of the proposed architecture exploration methodology: design specification is encoded into a mixed integer linear program and solved using one of the two approaches (monolithic or iterative). The outcome is verified using exact analysis techniques.

In sum, our methodology takes several subsequent steps to convert a design specification into a correct-by-construction architecture that is optimal with respect to an objective function (e.g., monetary cost, power consumption, reliability). We decouple the topology selection (which nodes are used, how they are connected) from the mapping (how the nodes are implemented), which brings more flexibility to the problem formulation. At the same time, in contrast with solving the mapping problem on a fixed topology (or vice versa), in our approach both the topology and the mapping are *jointly* selected within the same optimization problem. Therefore, it can be guaranteed that the solution is the optimal combination of the two.

## 3.2 Application Requirements

The top-down phase of the proposed methodology deals with selecting a system topology (how the components from the template $\mathcal{T}$ are connected) that meets a set of application requirements, both functional and non-functional. These requirements are defined at the application level (top part on Figure 3.1) As discussed in Section 2.3.2, we express design constraints as bounds on the properties of a system within a particular scope (e.g., for components of a certain type). The properties are defined as functions over a set of attributes of the architecture template $\mathcal{T}$, as well as over basis, auxiliary and problem-specific variables. We have shown that the basis variables $\beta \in B$ are necessary both for the mapping, i.e., computing the template attributes $\mathcal{A}^{\mathcal{T}}$ from library attributes $\mathcal{A}^{\mathcal{L}}$ using mapping variables $m \in M$, and for defining the scope of computing the property (and the constraint) using topology configuration variables $e \in E$ and path variables $\{y^{\pi}, w^{\pi}\} \in R$. In the following, we use these principles to formulate several different classes of design constraints that, in general, are applicable to various classes of cyber-physical systems and networks.

### 3.2.1 Number of Components

While the template $\mathcal{T}$ of the architecture has the maximum number of components of every type as well as all possible connections between them, this does not necessarily hold for the topology configuration $E^*$. Unless explicitly required to be instantiated, certain component types may be completely absent in the final topology. Let $m_{ij}^k$ be entries of the matrix $M^k$, which defines the mapping of components of type $k$ to the library. To ensure that at least (at most, exactly) $N$ such components are instantiated, one can use the following constraint:

$$\sum_{j=1}^{|P_k|} \left( \bigvee_{i=1}^{|\mathcal{L}_k|} m_{ij}^k \right) \geq (\leq, =)\ N, \tag{3.1}$$

where $P_k$ and $\mathcal{L}_k$ are subsets of, respectively, $V$ and $\mathcal{L}$ that include all the elements of type $k$. Disjunction in (3.1) determines which components in $P_k$ are used (it can be linearized with standard techniques shown in Section 2.4.1), and the sum provides their final number.

Similarly, the user can require a certain amount of specific components (i.e., with a particular subtype $s$) to be present in the architecture. For example, an electrical power system of an aircraft has a certain number of generators (power sources), and at least one of them must be a battery (emergency power supply) that has to be used in case if all primary generators (engines) fail. In this case, a submatrix $M^{k,s}$ of $M^k$ has to be used, which only includes the rows that are associated with library components $l^{k,s}$ having the subtype $s$, and then Constraint (3.1) can be applied.

### 3.2.2 Interconnection

Interconnection constraints are used for enforcing valid connections between components or limiting the number of allowed connections. In cyber-physical systems they can arise from different contexts. For example, from the functional viewpoint, both the inputs and the outputs of a manufacturing machine must be connected to conveyors, so that production units processed by the machine can move along the production line. Similarly, in a power distribution network, electrical loads must stay connected to some power source. At the same time, generators must not be connected in parallel, i.e., to the same electrical bus, for safety concerns. Such requirements can be enforced using linear arithmetic constraints.

The first group of constraints is used to set a bound on the number of connections between the components of certain types. Let $P$ be a partition over the set $V$ of vertices of the template $\mathcal{T}$ (explained in Section 2.1.3), and $A$, $B$ and $C$ be sets in $P$. We can then write the following expressions:

$$\sum_{j=1}^{|B|} e_{a_i b_j} \geq (\leq, =)\ N \qquad \forall i \in \mathbb{N} : 1 \leq i \leq |A|, \tag{3.2a}$$

$$N\delta_i \leq (\geq, =) \sum_{j=1}^{|B|} e_{a_i b_j} \qquad \forall i \in \mathbb{N} : 1 \leq i \leq |A|, \tag{3.2b}$$

where $e_{a_i b_j}$ is an edge from node $a_i$ to node $b_j$ (and similarly $e_{b_j c_k}$). Constraint (3.2a) prescribes that there exist at least (at most, exactly) $N$ connections from a node in $A$ to a node in $B$. One can note that using $(=, \geq)$ in (3.2a) with $N \geq 1$ forces every node from $A$ to be instantiated, according to our definitions. Instead, it may be required to enforce the constraint only for the nodes from $A$ that are used and, at the same time, not to affect the decision of using them. In this case, Constraint (3.2b) can be applied, which uses auxiliary variables $\delta_i$ that, as shown in Section 2.1.2, evaluate to true if the component $v_i$ is instantiated and zero otherwise. This constraint is a linearized form of a forced implication: if the left hand side is true, i.e., the component $v_i$ is used, then the right hand side also holds. The latter can also be true if $\delta_i = 0$, however, this is equivalent to a non-instantiated component with connections, which will be restricted by the mapping constraint (2.1a) from the basis.

Another group of constraints defines the “if-then” relationships between incoming and outgoing connections of nodes using the following expressions:

$$\bigvee_{i=1}^{|A|} e_{a_i b_j} \leq \bigvee_{k=1}^{|C|} e_{b_j c_k} \quad \forall j \in \mathbb{N} : 1 \leq j \leq |B|, \tag{3.3a}$$

$$\bigvee_{i=1}^{|A|} e_{a_i b_j} + \bigvee_{k=1}^{|C|} e_{b_j c_k} \leq (\geq)\ 1 \quad \forall j \in \mathbb{N} : 1 \leq j \leq |B|. \tag{3.3b}$$

Constraint (3.3a) states that if node $b_j$ has a connection to any node in $A$, then it must also have a connection to at least one node in $C$. In turn, Constraint (3.3b) requires that if $b_j$ has (does not have) a connection to any node in $A$, then it must not (must) have a connection to a node in $C$ (i.e., "If A then not B / If not A then B"). In both constraints, disjunction operators that calculate the least upper bound of the variables, can be linearized using standard techniques.

A *bidirectional connection* between nodes $v_i$ and $v_j$ implies that they are connected by two oppositely directed edges $e_{ij}$ and $e_{ji}$, i.e., a signal can flow both from $v_i$ to $v_j$ and vice versa. Such pairs of connections can represent, for example, contactors between electrical buses, which allow the power to flow in both directions depending on the system's dynamic state. Following expressions are possible:

$$e_{ij} = e_{ji} \qquad \forall i, j \in \mathbb{N} : 1 \leq i \leq |A|, 1 \leq i \leq |B|, \tag{3.4a}$$

$$e_{ij} \leq 1 - e_{ji} \qquad \forall i, j \in \mathbb{N} : 1 \leq i \leq |A|, 1 \leq i \leq |B|. \tag{3.4b}$$

Constraints (3.4a) and (3.4b) state, respectively, that the components from $A$ must and must not have bidirectional connections to components from $B$.

Finally, our formulation provides additional flexibility and expressiveness by using *subtypes* that can be assigned to components as attributes. A subtype corresponds to a feature or a characteristic of a component, possibly in another dimension of classification. For example, components of an electrical power system can be grouped by following types: generator, AC bus, rectifier, DC bus, load. At the same time, these components can have either high or low voltage levels, which is their key characteristic that defines, in particular, a subset of composition rules, such as allowing or restricting the direct connections between high and low voltage devices. In other words, different voltage levels represent different subtypes[1].

Component subtypes can be used to add problem-specific composition rules to the formulation, extending the ones provided with the library. Let components from sets $A$ and $B$ (subsets of the partition $P$) be labeled with subtypes $s_1$ and $s_2$, which are incompatible. Following expression forbids connections between components $v_i \in A$ having subtype $s_1$ and components $v_j \in B$ having subtype $s_2$:

$$\sigma_i^{s_1} + \sigma_j^{s_2} - 1 \leq 1 - e_{ij} \qquad \forall i, j \in \mathbb{N} : 1 \leq i \leq |A|, 1 \leq i \leq |B|, \tag{3.5}$$

[1]We note that different component types can have same subtypes, e.g., both generators and loads can have several voltage levels, which have to be considered when connecting these devices.

where $\sigma_i^{s_1}$ and $\sigma_j^{s_2}$ are auxiliary boolean variables, which evaluate to one if components $v_i$ and $v_j$ are mapped to library elements that have, respectively, subtypes $s_1$ and $s_2$, and zero otherwise. One can obtain their values using the mapping variables $m \in M$. For example, for a component $v_i \in A$, $\sigma_i^{s_1}$ can be computed as $\bigvee^k m_{ki}$, with $k$ including the indices of elements from $\mathcal{L}$ that have the subtype $s_1$. A linear encoding for this computation can be obtained using the techniques summarized in Section 2.4.1.

### 3.2.3 Balance and Workload

A set of constraints is used to enforce conservation laws or *balance equations* in physical systems, e.g., by requiring that the maximum power provided by a source in $\mathcal{T}$ is greater than or equal to the maximum power required by the sinks connected to it (with connectors, buses, paths etc). Let the node $b$ have an intermediate type in the functional flow $\mathcal{F}$, which is neither a source nor a sink, and let $A$ and $C$ be the sets of, respectively, direct predecessors and direct successors of $b$. Then, a "local" balance equation at the terminals of $b$ can be written as

$$\sum_{i=1}^{|A|} x_{a_i} e_{a_i b} \geq (=) \quad \sum_{j=1}^{|C|} y_{c_j} e_{bc_j}, \tag{3.6}$$

where $x_{a_i}$ is the input value imposed by $a_i$ and $y_{c_j}$ is the output value assigned to $c_j$. Such a "local" assertion can be used, for instance, to express the guarantees of the node $b$. Alternatively, given a set of sources $A$ and a set of sinks $Z$, we can directly write a "global" balance equation for each source node as

$$y_{a_i} \geq (=) \quad \sum_{j=1}^{|Z|} x_{z_j} \eta_{a_i z_j} \quad \forall\ i \in \mathbb{N}:\ 1 \leq i \leq |A|, \tag{3.7}$$

where the variables $\eta_{a_i z_j}$ are entries of the walk indicator matrix $\eta$, which evaluate to one if there exists a path (direct walk) from source $a_i$ to sink $z_j$, and 0 otherwise (the concept of walk indicator matrix was previously introduced by N. Bajaj et al in [12]; variables from $\eta$ can be related to edge variables $e_{ij}$ using a set of linear arithmetic constraints). Assertions as in (3.7) can be used to express the guarantees of a source node. Finally, to require that a quantity that propagates through the system (e.g., production units, electric charge) is not stored in the intermediate nodes along the path, i.e., everything generated by the source reaches the sink, one can turn inequalities in expressions (3.6) and (3.7) into equalities.

**Example 12** (Flow Balance)**.** *We recall the Example 10 of a fuel distribution system. Let each edge $e_{ij}$ of $\mathcal{T}$ be associated with a real variable $\lambda_{ij}$ that expresses the fuel flow through that edge (e.g., a connecting pipe). The input flow into the components of the fuel system must be equal to the output flow (except sources and sinks). According to* (3.6)*, we can express the flow balance constraint for a node $v_j$ as $\sum_{i=1}^{|V|} \lambda_{ij} e_{ij} = \sum_{k=1}^{|V|} \lambda_{jk} e_{jk}$. With this MILP constraint, the flows for output edges $e_{jk}$ will be assigned in a way that no fuel is stored in the intermediate components. Moreover, flow variables $\lambda_{ij}$ will be forced to zero if the corresponding edge variable is zero.*

A particular concern in system architecture design that makes an extensive use of flow variables and balance constraints is managing the *workload* of the system. For example, nodes in $\mathcal{T}$ can represent processor cores or industrial machines. For the architecture to be feasible, such nodes must be implemented in a way that they are able to handle certain steady-state values of the input load. In our formulation, they can be labeled with a *throughput* $\mu$. If the cumulative flow into a component is higher than its throughput, then it cannot handle the input traffic, i.e., the system is *overloaded* and some messages (or production units, requests) are discarded. This can deteriorate the quality of service of a system as well as the safety. To avoid overloading, we can bound the incoming workload for node $v_j$ as follows:

$$\sum_{i=1}^{|V|} \lambda_{i v_j} \leq \mu_j, \tag{3.8}$$

requiring that a valid input is processed before the next one arrives. Otherwise it will be discarded (assuming a deterministic flow). If $m^k_{ij}$ is the element of the mapping matrix $M^k$ associated with $v_j$ and component $l^k_i$ in $\mathcal{L}_k$, $k$ being the type of $v_j$ (e.g., a processor), and $\mu^{\mathcal{L}_k}$ is the vector of throughputs for the components in $\mathcal{L}_k$, then, according to (2.5), we have $\mu_j = \sum_{i=1}^{|\mathcal{L}_k|} m_{ij}\mu_i^{\mathcal{L}_k}$.

In general, stochastic input flows can also be considered, so that $\lambda_{ij}$ represents the mean value of some probability distribution (e.g., exponential) or some worst-case value. While the dynamic behavior of a CPS has to be studied with other techniques, such as simulation, our methodology allows us to provide guarantees for some static (steady-state) combination of parameters that represent system dynamics. For instance, one can set up a tight constraint on the workload considering some heavy traffic (large values of $\lambda$) and generate architectures that are able to cope with such inputs, which also guarantees that smaller loads can be handled by the system. Similarly, one may find out that a feasible architecture does not exist, so that relaxing the workload or some other constraint may be required.

### 3.2.4 Reliability

In safety-critical applications, reliability requirements prescribe that a functional link must be guaranteed with a certain probability for a system to operate correctly, that is, the probability for a sink to be disconnected from all sources should be less than a desired threshold. In general, to formulate a reliability constraint, one needs to compute the probability of composite failure events in the system, starting from the failure probability of components. Several assumptions are made: when the component fails, no recovery is possible and the adjacent links are no longer usable, and failures in different components are independent. A symbolic constraint can then be recursively computed to enumerate all possible failure events. As discussed in [12], such exact computation has exponential complexity, as well as the enumeration of all possible topology configurations of the template $\mathcal{T}$. In other words, this soon leads to problem formulations that are untractable. Moreover, such symbolic constraints are highly nonlinear and extremely complex, so using MILP is not possible.

Instead, to capture this class of constraints, we leverage the efficient mixed integer linear encoding techniques based on the approximate reliability computations proposed by P. Nuzzo in [96]. These estimations still have the correct order of magnitude and stay within an explicit theoretical bound on the approximation error. They are based on the notion of degree of redundancy $h_{ij}$, which is a number of components of type $j$ used in at least one path of a functional link $F_i$. With this notion, the failure probability of $F_i$ can be estimated as follows:

$$\tilde{r}_i = \sum_{j \in I_i} h_{ij} p_j^{h_{ij}} \tag{3.9}$$

where $I_i$ is the set of all component types that jointly implement $F_i$, $p_j$ is the failure probability of any of the components of type j. Values of $h_{ij}$ can be computed as $\sum_{k=1}^{|P_j|} \eta_{ki}$ with $P_j$ being the subset of $V$ containing all nodes of type $j$ and $\eta_{ki}$ being the values of the walk indicator matrix $\eta$ (introduced in [12] and previously mentioned in Section 3.2.3). The reliability constraint can then be defined simply as

$$\tilde{r} < r^* \quad \forall i \in F_i. \tag{3.10}$$

Expression (3.9) is nonlinear but there exists a linear encoding using auxiliary binary variables and constraints. We refer the reader to [96] for details.

### 3.2.5 Timing

A typical timing requirement specifies an upper bound or deadline on the latency (delay) for performing a certain action (propagating a signal from a source to a sink, delivering a message, completing a task). The total delay depends on the propagation or processing delays of the individual components. Such properties in our formulation can be captured with the basis variables, since they define, in particular, how the signals propagate through the components of the network. Therefore, we assume that each node and each edge in $\mathcal{T}$ is labeled with a delay $\tau$ and consider a simplified delay model, where the delay of a cascade of components is equal to the sum of the delays of each component and connection. Let $\Pi^{ab}$ be the set of all paths from source $v_a$ to sink $v_b$. We can ensure that the propagation delay of each path $\pi$ in $\Pi^{ab}$ does not exceed $\tau^*$ by requiring

$$\sum_{i=1}^{|V|} \tau_i w_i^{\pi} + \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} \tau_{ij} y_{ij}^{\pi} \leq \tau^* \quad \forall \pi \in \Pi^{ab}, \tag{3.11}$$

where $\tau_i$ is the delay of node $v_i$ in $\mathcal{T}$, which can be computed using the formula (2.5). Path variables $w_i^{\pi}$ and $y_{ij}^{\pi}$ ensure that the non-relevant delays, i.e., delays of components and connections that do not belong to the path $\pi$, will be discarded (such timing constraint has been illustrated in Example 11). Node delays $\tau_i$ can be, for instance, computation or processing delays, while edge delays $\tau_{ij}$ can be interpreted as communication delays (e.g., time required for sending a packet over a wireless link). Depending on the specific problem, some terms in (3.11) may be omitted.

Certain classes of CPS, such as manufacturing systems, also possess another important timing characteristic, namely the *idle rate* of components, which is the difference between the processing rate and the input flow rate. This quantity has to be either bounded or minimized, because idle state of the system may be extremely economically inefficient (silicon foundries are good examples). Let $\mu_j$ be the processing delay of component $v_j$, and $\lambda_{ij}$ be the input flow rate to component $v_j$ originating from component $v_i$. We can then use the following constraint to limit the total idle rate of an architecture to be below a required value $\varphi^*$:

$$\sum_{j \in Idx(PU)} \left( \mu_j - \sum_{i=1}^{|V|} \lambda_{ij} \right) \leq \varphi^*, \tag{3.12}$$

where $Idx(PU)$ is a set of indices corresponding to the nodes in $V$ that are labeled as processing units, i.e., can have an idle state.

Constraint (3.11) can be also enforced conditionally based on other system properties. For example, if the input flow of some production units or details to the manufacturing line exceeds a certain threshold, i.e., the line is heavily loaded, then it should be subject to a tighter timing constraint, so that all units can be processed and the system does not slow down. This may require the processing machines to reconfigure or set higher priority to such inputs, while still being able to handle other incoming requests. Such conditional timing constraints can be set up as simple implications:

$$\left( \sum_{i=1}^{|V|} \lambda_{i,v_a} \geq \lambda^* \right) \Rightarrow (T = 1), \tag{3.13}$$

where $v_a$ is a source node of $\pi$, $\lambda^*$ is the input rate threshold, after which the timing constraint must be enforced, and $T$ is an auxiliary Boolean variable that evaluates to one if the Constraint (3.11) holds, and zero otherwise (consequently, forcing $T$ to 1 entails that (3.11) must hold). Expression (3.13) can be easily linearized. In general, such principle can be used also for activating other types of constraints (e.g., interconnection) based on a specified condition.

### 3.2.6 Routing

In a network of components, sources have to be connected to sinks by a set of paths (routes), which must satisfy a set of requirements. These are *routing constraints* that require, in particular, the presence (or absence) of certain paths, specify their structure (e.g., nodes of certain type or subtype must be visited), set an upper or lower bound on their number and length, enforce the difference between them (e.g., fully disjoint, partly disjoint, equal etc).

As discussed in Section 2.1.3, in our formulation every path (route) $\pi$ is defined by the sets of path variables $y^\pi$ and $w^\pi$, which symbolically encode, respectively, the edges and the nodes of the template $\mathcal{T}$ that implement $\pi$. Moreover, for the sake of decreasing the complexity of the optimization problem, they are created only for routing requirements, i.e., for paths between those nodes that must be connected within a given system. For example, if there is no connection required between the source A1 and sink E2 of the template $\mathcal{T}$ shown on Figure 2.1a, then the path $\pi(A1 \rightarrow E2)$ is not declared in the problem formulation as well as the path variables and constraints for $\pi$. Let $Q$ be the set of source-destination pairs of $\mathcal{T}$. Instead of enumerating all possible pairs, $Q$ only stores those pairs that are specified by the user in the requirements. If several replicas of a certain path are required, then $Q$ can also contain duplicates of the same pair.

We have discussed in Section 2.2.2 that only simple (loopless) paths are considered in our formulation. Therefore, a set of path constraints (2.2)-(2.4b) is automatically enforced for every path declared by the user. These constraints are a part of the routing requirements. Hereafter, we formalize an additional set of linear arithmetic constraints that can be set up for a path, but are *optional.* In particular, one can require the two paths to be different from each other, for example, when the network topology has to include several routes between the same source and sink. It can be expressed as follows:

$$\sum_{i=1}^{|V|} (w_i^{\pi_1} \oplus w_i^{\pi_2}) \geq N^*_{diff}, \tag{3.14}$$

where $N^*_{diff}$ is the number of differences (in terms of nodes) that paths $\pi_1$ and $\pi_2$ must have. Similar expression can be defined with the edge variables $y^{\pi}$. Such constraints allow designers to flexibly specify some degree of distinction between routes, ranging from a single node (or edge) being different in $\pi_1$ and $\pi_2$ to the maximum difference. They are commonly defined for replicas of the same route to force at least one difference between them (otherwise it may happen that they will be equal, because the optimizer tries to minimize the system cost, which is valid but does not make sense from the functional/routing point of view). The XOR operation in Constraint (3.14) can be written in linear form by introducing an auxiliary Boolean variable $z_i$ for each $i$ and adding following constraints: $z_i \leq w_i^{\pi_1} + w_i^{\pi_2}$, $z_i \geq w_i^{\pi_1} - w_i^{\pi_2}$, $z_i \geq w_i^{\pi_2} - w_i^{\pi_1}$, $z_i \leq 2 - w_i^{\pi_1} - w_i^{\pi_2}$.

It may also be required that two routes are independent (completely disjoint). This is a common concern in routing, which increases the network resiliency: when the primary (best) route is not available, an alternative one can be used. The constraints can assume the following form:

$$y_{ij}^{\pi_1} + y_{ij}^{\pi_2} \leq 1 \quad \forall i,j \in \mathbb{N} : 1 \leq i,j \leq |V|, \tag{3.15a}$$

$$w_i^{\pi_1} + w_i^{\pi_2} - 1 \leq 0 \quad \forall i \in \mathbb{N} : 1 \leq i \leq |V|, i \neq \{a,b\}, \tag{3.15b}$$

where $a$ and $b$ are numerical indices of source and sink nodes in $E$. Constraint (3.15a) states that all edges of $\pi_1$ and $\pi_2$ must be disjoint, i.e., there are no edges $e_{ij}$ such that both $y_{ij}^{\pi_1}$ and $y_{ij}^{\pi_2}$ evaluate to one. Constraint (3.15b) is a slightly stricter version, which requires all nodes of $\pi_1$ and $\pi_2$ to be different (except the source and the sink).

**Example 13** (Path Difference)**.** *Figure 3.3a-c illustrates the constraints enforcing the difference between two paths $\pi_1$ and $\pi_2$. On Figure 3.3a, the paths distinct from each other with one node*

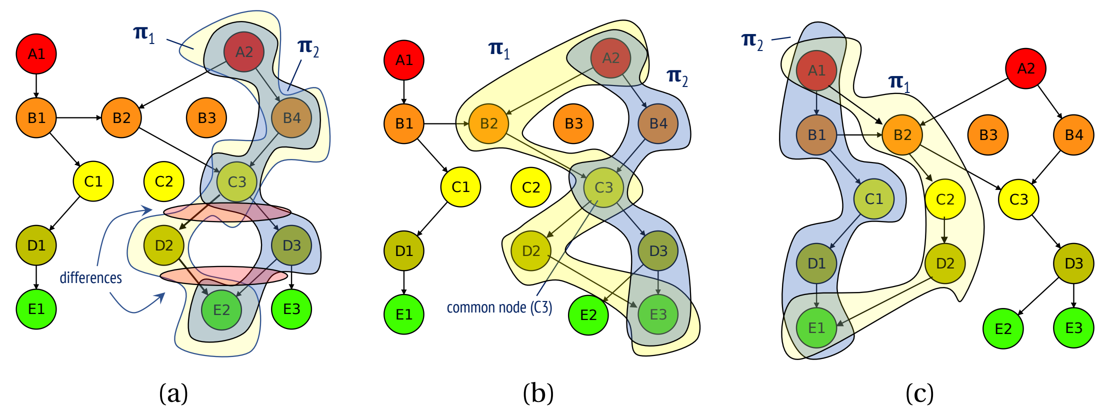


Figure 3.3 – Example of path difference constraints: (a) Two paths with difference in one node (D2 vs D3); (b) Paths with disjoint edges; (c) Paths with disjoint nodes.

*(D2 and D3 for, respectively, $\pi_1$ and $\pi_2$), i.e., Constraint* (3.14) *is applied with $N^*_{diff} = 1$. Paths on Figure 3.3b have disjoint edges, which satisfies* (3.15a)*, while they still share a common node (C3). Finally, paths on Figure 3.3c are completely disjoint both in terms of nodes and edges as required by* (3.15b).

It is also possible to set up a bound $N^*_{hops}$ on the number of hops of the path $\pi$:

$$\sum_{i,j} y^{\pi}_{ij} \leq (\geq, =)\ N^*_{hops}. \tag{3.16}$$

Similarly, the max (min, exact) number of nodes in $\pi$ can be required.

Alltogether, the routing constraints (including the path constraints from the basis) allow great flexibility in defining the system paths and setting up the restrictions for them. In a broad sense, they can be seen as interconnection constraints, however, we classify them in a separate group as having slightly broader, system-wide scope, while the former are more associated with direct connections between components. We further exemplify them in Chapter 6.

### 3.2.7 Link Quality

One of the important domains for applying the presented architecture exploration methodology is wireless communication, which is a crucial part of networked embedded and cyber-physical systems. The three following categories of design requirements focus primarily on this domain.

Many Quality of Service (QoS) metrics of a wireless network (e.g., latency, energy consumption, packet loss) depend on the link quality (LQ), which can be expressed using different metrics. Overall, constraints that specify a bound on the LQ, play a significant role in synthesizing wireless network topologies. Such constraints are typically defined in the scope of network routes, i.e., they are end-to-end and can differ from route to route. Therefore, they extensively use the path variables $y^{\pi}$ and $w^{\pi}$ from the basis of the exploration problem, which further demonstrates their importance in problem formulations.

One of the most used metrics for defining the LQ is *received signal strength* (RSS) of wireless links. RSS is measured by the receiving radio and depends on several factors including the signal propagation in the wireless channel as well as transmitting and receiving device characteristics. In our formulation, edges of the network template $\mathcal{T}$ can be labeled with real decision variables $RSS_{ij}$ which can be computed using the following constraint:

$$RSS_{ij} = PL_{ij} + tx_i + g_i + g_j \qquad 1 \le i, j \le |V|. \tag{3.17}$$

Constraint (3.17) computes the RSS of every link $e_{ij}$ between a transmitter (TX) $v_i$ and receiver (RX) $v_j$ as a sum of the link path loss $PL_{ij}$, TX and RX antenna gains $g_i$ and $g_j$, and TX power $tx_i$. The value of $PL_{ij}$ can be analytically estimated using a channel model, such as the log-distance model [110]. The rest are component attributes and can be computed by associating them to corresponding values from $\mathcal{L}$ using the formula (2.5). For example, let $g^{\mathcal{L}}$ be the vector of antenna gains for the components in $\mathcal{L}$ and $m_{ij}$ be the element of the mapping matrix $M$ associated with node $v_j$ and component $l_i \in \mathcal{L}$. Then, we have $g_j = \sum_{i=1}^{|\mathcal{L}|} m_{ij} g_i^{\mathcal{L}}$. Similarly, one can compute other attribute values, e.g., TX power.

We model the noise in the wireless channel by associating edges $e_{ij}$ of the network graph (wireless links) with numerical values $\gamma_{ij}$. These values can model the backround noise caused by different reasons, for example, by an interfering communication device (e.g., WiFi router) located in the area. The noise can be estimated analytically or through measurements, which can bring different levels of accuracy for channel modeling. Using the values $\gamma_{ij}$, another important LQ metric, namely Signal-to-Noise ratio (SNR), can be computed simply as

$$SNR_{ij} = RSS_{ij} - \gamma_{ij}. \tag{3.18}$$

Further, such metrics as Bit Error Rate (BER), Packet Error Rate (PER) and the number of expected transmissions, also known as ETX, can be considered as LQ constraints. The corresponding attributes $BER_{ij}$, $PER_{ij}$ and $ETX_{ij}$ of edges $e_{ij}$ can be computed as follows:

$$BER_{ij} = \frac{1}{2}\operatorname{erfc}\left(\sqrt{SNR^{*}_{ij}}\right), \tag{3.19a}$$

$$PER_{ij} = 1 - (1 - BER_{ij})^{N}, \tag{3.19b}$$

$$ETX_{ij} = \frac{1}{1 - PER_{ij}}, \tag{3.19c}$$

where $SNR^{*}_{ij}$ is the normalized signal-to-noise ratio of the link (“SNR per bit”) and $N$ is the packet length. We note that (3.19a) defines the BER assuming a QPSK modulation and can be computed differently for other types of modulation. Overall, it can be clearly observed that all the expressions above are highly nonlinear in the real variables involved. Nevertheless, they can still be encoded as MILP constraints by using a linear approximation in a form of a lookup table that we introduced in Section 2.4.2. In current case, several intervals of SNR of length 0.5 dB or 1 dB each can be considered, and corresponding values of BER, PER or ETX (depending on the ones used in the formulation) can be precomputed for each interval. Similar approach is used in some existing network simulators, e.g., Castalia [19]. We omit the resulting linear expressions for brevity.

Finally, all the LQ metrics presented above are related to the edges of the template $\mathcal{T}$. Using the basis variables $y^{\pi}$ it is easy to set up link quality requirements as bounds on these metrics. We exemplify them for RSS and BER constraints for a route $\pi$ that can be written as follows:

$$RSS_{ij} y_{ij}^{\pi} \geq RSS^{*} \qquad 1 \leq i, j \leq |V|, \tag{3.20a}$$

$$BER_{ij} y_{ij}^{\pi} \leq BER^{*} \qquad 1 \leq i, j \leq |V|, \tag{3.20b}$$

where $RSS^{*}$ and $BER^{*}$ are, respectively, the minimum RSS and the maximum BER of a signal transmitted over every link of $\pi$. Similarly, these constraints can be defined for other LQ metrics. Also, the bounds can be enforced only on certain links within a route or on a particular link, which can be a member of several routes. In the former case, the scope of the constraints can be easily manipulated by using only dedicated variables $y_{ij}^{\pi}$, while in the latter case it is more convenient to use edge variables $e_{ij} \in E$.

We also note that, unlike network simulators, the path loss values $PL_{ij}$ of wireless links are computed only once, i.e., a static case is assumed, in which every signal transmitted from $v_i$ to $v_j$ has the same attenuation. This is a simplification, and in practice signal behavior is affected by various random factors, such as multipath propagation, sporadic interference and so on. Path loss can be computed using different channel models, and it is important to carefully select and calibrate the proper one. Also, channel measurements and surveys can be used, as done, for example, in [108]. Overall, for obtaining the LQ guarantees on generated topologies, one can consider worst-case scenarios with respect to the channel model used (e.g., worst possible path loss). We further discuss this issue in Section 6.3, where we propose a set of improvements for conventional models of the wireless channel.

### 3.2.8 Energy Consumption

Energy consumption is a common concern in designing networked CPS. It can affect the utilization cost of the system as well as its lifetime, or the lifetime of some of its components (especially if they are battery powered). In general, to compute the energy consumed by a system or its parts, one has to estimate the respective amounts consumed by its components and connections (nodes and edges in our formulation). For diffenent systems, these energies can be computed differently, based on the type of the physical plant, a network protocol and so on. In this thesis, we exemplify this type of requirements for the wireless network domain.

Wireless devices consume energy primarily by sending and receiving signals (messages, packets) over the wireless channel, i.e., the radio is the main consumer. However, other compoments can also highly affect the energy consumption and node lifetime in the long run.

Therefore, it is required to consider them as well during estimations. Every component in the network template $\mathcal{T}$ can be labeled with the current drawn by its hardware (e.g., radio, CPU, sensors) in different operating modes. In this example, we distinguish between the radio TX and RX current $c^{TX}$ and $c^{RX}$, while all the remaining current values for active and sleep modes are cumulatively denoted by, respectively, $c^{active}$ and $c^{sleep}$. We also label the wireless links $e_{ij}$ with the following values: bit rate $b_{ij}$ of links (assumed to be constant) and energies $\epsilon_{ij}^{TX}, \epsilon_{ij}^{RX}$ consumed for sending/receiving a data packet over these links. The latter are real decision variables and can be computed as follows:

$$\epsilon_{ij}^{TX} = ETX_{ij} \cdot U \cdot c_i^{TX} \cdot \frac{N}{b_{ij}} \qquad \forall i,j \in \mathbb{N} : 1 \leq i,j \leq |V|, \tag{3.21a}$$

$$\epsilon_{ji}^{RX} = ETX_{ji} \cdot U \cdot c_j^{RX} \cdot \frac{N}{b_{ij}} \qquad \forall i,j \in \mathbb{N} : 1 \leq i,j \leq |V|, \tag{3.21b}$$

where $N$ is the packet length, $U$ is the voltage (also set up as a constant) and $ETX_{ij}$ is the expected number of transmissions of a packet necessary for it to be received without error at its destination. $ETX_{ij}$ depends on the path loss and interference and can be computed as discussed in Section 3.2.7. Values of $c_i^{TX}$ and $c_j^{RX}$ are obtained from the library mapping using the formula (2.5), while their product with the real variable $ETX_{ij}$ can be linearized as shown in 2.4.1. For simplicity, we omit the impact of packet acknowledgements as well as other system packets stipulated by a particular protocol. For a more accurate computation, they can also be added to formulas (3.21a)-(3.21b).

We now assume a collision-free Time Division Multiple Access (TDMA) protocol, in which the nodes wake up only within a few dedicated time slots within each superframe SF for sending and receiving packets, and that the schedule of the protocol is fixed. There are $n$ slots in SF, the duration of each is $t^{slot}$, so the superframe duration is $t^{SF} = n \cdot t^{slot}$. The total energy $\epsilon_i^{radio}$ consumed by the node $v_i$ for communication within a superframe, i.e., for sending and receiving packets over all routes where $v_i$ is involved, can be expressed as

$$\epsilon_i^{radio} = \epsilon_i^{TX} + \epsilon_i^{RX} = \sum_{\pi \in \Pi} \Big( \sum_{j=1}^{|V|} \epsilon_{ij}^{TX} y_{ij}^{\pi} + \sum_{j=1}^{|V|} \epsilon_{ji}^{RX} y_{ji}^{\pi} \Big). \tag{3.22}$$

In (3.22) $\Pi$ is the set of all routes declared by the user in the specification, i.e., the contributions of every path $\pi$ is considered by the virtue of using the variables $y_{ij}^{\pi}$, which belong to the basis of the architecture exploration problem introduced in Section 2.2. Accurate computation of energy consumption is one of many examples of their utility. We note that if the period $T$ of sending packets for some route $\pi$ is longer than $t^{SF}$, i.e., a packet is not sent in every superframe on this route, than the contribution of $\pi$ has to be scaled by a fraction $\frac{t^{SF}}{T}$.

Energies $\epsilon_i^{active}$ and $\epsilon_i^{sleep}$ consumed in, respectively, active and sleep modes of $v_i$ within SF, are calculated as

$$\epsilon_i^{active} = U \cdot c_i^{active} \cdot t^{slot} \cdot k, \tag{3.23a}$$

$$\epsilon_i^{sleep} = U \cdot c_i^{sleep} \cdot t^{slot}(n-k), \tag{3.23b}$$

where $k$ is the number of time slots in which $v_i$ must either transmit or receive assuming each TX and RX requires a separate slot.

Now, considering a node $v_i$ of the wireless network, the following energy consumption constraint can be imposed:

$$\epsilon_i^{radio} + \epsilon_i^{active} + \epsilon_i^{sleep} \leq \epsilon^*. \tag{3.24}$$

Constraint (3.24) sets up an upper bound on the energy consumed within a superframe. Dividing both sides of (3.24) by the superframe duration $t^{SF}$ provides a power consumption constraint, which may be more convenient to use in certain contexts.

Additionally, we propose a more intuitive way for expressing the energy consumption requirements, i.e., by specifying a lower bound on the node lifetime:

$$\frac{B_i}{\epsilon_i^{radio} + \epsilon_i^{active} + \epsilon_i^{sleep}} \cdot t^{SF} \geq L^*, \tag{3.25}$$

where $B_i$ is the battery capacity of the node $v_i$ (assumung it has limited energy resource), and $L^*$ is the minimum required lifetime. The left hand side expresses the actual lifetime of $v_i$, which is computed as the number of superframes that $v_i$ can stay alive with respect to the energy consumed multiplied by the duration of a superframe $t^{SF}$. Clearly, Constraint (3.25) is in nonlinear form due to the division by a sum of real decision variables. It is easy to obtain an equivalent linear formulation by inverting (3.25), since the remaining terms are constants. The overall lifetime of a wireless network typically depends on a subset of critical nodes, i.e., when these nodes are down, the network cannot function anymore. The network lifetime constraint, therefore, can be specified by setting a lower bound on the lifetime of this subset using the formula (3.25).

We model all types of interference using variables $\gamma_{ij}$ that represent the channel noise for links $e_{ij}$, while the path loss values of links are provided by a channel model. A more complicated encoding can be obtained by taking the packet collisions into account. In our formulation, collisions can affect the number of retransmissions and the computation of the $ETX_{ij}$ attributes of links, which, in turn, has an influence on energy. To capture their effect, an analytical model, such as the additive interference model [42] can be used. More simple, links can be labeled with probabilities of collisions, which can be mean values of a certain distribution. These labelings will be considered when computing ETX. Overall, a variety of improvements in the expessions for LQ attributes is possible, and we omit them here, remarking that they can be encoded similarly to currently shown (3.19a)-(3.19c) by using a lookup table.

Finally, similar requirements can be obtained for other classes of network protocols, such as contention-based Carrier Sense Multiple Access (CSMA). The main difference is related to computing the energies consumed in different states of components. Instead of the lengths of a slot and a superframe, parameters, such as *duty cycle* (the overall length of the period of repeating sleep and active states of the node, similar to a superframe duration) and *epoch* (period for sending packets) should be used. For event-based systems, epoch can be represented by a mean value of a probability distribution of events that require the nodes to communicate when they occur.

### 3.2.9 Localization

Wireless localization and tracking is nowadays being applied in various areas: industrial, retail, healthcare, building automation. Objects to be localized can be both static (e.g., crates, goods, medical machinery) or mobile (e.g., workers, hospital personnel or patients, robots etc). Systems that localize and track mobile objects in real time are well known as Real-Time Location Systems (RTLS). A variety of technologies and algorithms have been developed/applied for RTLS, such as WiFi, Wireless Sensor Networks (WSN), Radio Frequency Identification (RFID), Bluetooth Low Energy (BLE), Ultra wideband (UWB) and others. Several papers provide comprehensive reviews of existing localization techniques [75, 44, 29].

In this work, we focus on range-based localization systems that estimate distances between anchor nodes and a target node by using received signal strength, time of arrival or other related metrics. Evaluation of such systems is typically performed using a set of locations in the network deployment area, in which the quality of localization (e.g., accuracy, precision) is estimated [51]. This set of “evaluation locations” can be seen as possible locations of a mobile device (e.g., cartesian coordinates). Let $\Lambda^{eval}$ be the array of these locations, while $\Lambda^{\mathcal{T}}$ is the array of locations of nodes in $\mathcal{T}$, so that $|\Lambda^{\mathcal{T}}| = |V|$. Also, let real variables $r_{ij}$ be the entries of the reachability matrix $\mathbf{r}$, $|\mathbf{r}| = |\Lambda^{\mathcal{T}}| \times |\Lambda^{eval}|$, computed as follows:

$$r_{ij} = (RSS_{ij} \geq RSS^*) \wedge \delta_i \qquad 1 \leq i \leq |\Lambda^{\mathcal{T}}|,\ 1 \leq j \leq |\Lambda^{eval}|, \tag{3.26}$$

where $\delta_i$ is a binary variable equal to one if the component $v_i$ is used and zero otherwise (introduced in Section 2.1.2). Constraint (3.26) forces the value of $r_{ij}$ to be true if the mobile node located at $\lambda_j^{eval} \in \Lambda^{eval}$ is *reachable* by $v_i$, i.e., it is able to receive the signal from a node $v_i$ located at $\lambda_i^{\mathcal{T}} \in \Lambda^{\mathcal{T}}$ with signal strength of at least $RSS^*$. Conjunction with $\delta_i$ is linearizable with standard techniques shown in Section 2.4.1. The values of $RSS_{ij}$ can be computed similarly to (3.17). Also, other LQ metrics, such as SNR or BER, can be used in (3.26).

According to the formulation above, we define reachability of a mobile device by an anchor node as a stable link (with an LQ constraint) between the two. We now assume that the location of the target node, i.e., its coordinates, is calculated using trilateration. The latter requires a minimum of 3 (4) distances between the target and the anchors to be estimated for calculating its 2D (3D) location. To satisfy this algorithm requirement, the mobile device must

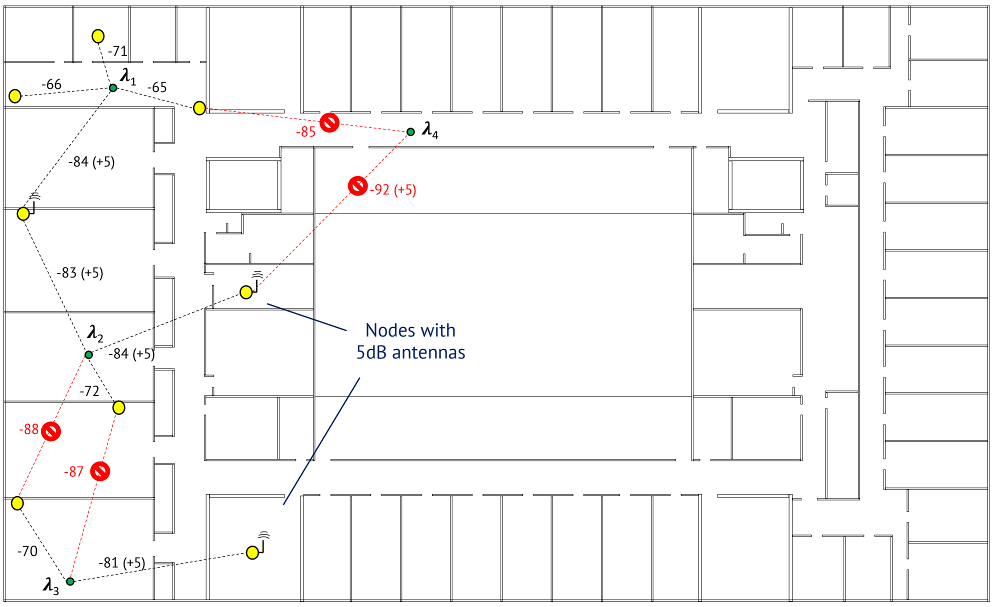


Figure 3.4 – Reachability example: locations $\lambda_1 - \lambda_4$ are reachable by anchor nodes (yellow) of the localization system, if corresponding RSS values are at least -80 dBm. Some links are shown in red color, meaning that communication may still be possible, but our reachability condition is not satisfied. Some links are not shown for simplicity.

be reachable by several anchors in any of its possible locations. We make a simplification by enforcing this requirement on a set of evaluation locations $\Lambda^{eval}$, which can be imposed as follows:

$$\sum_{i=1}^{|\Lambda^{\mathcal{T}}|} r_{ij} \geq N \qquad \forall j \in \mathbb{N} : 1 \leq j \leq |\Lambda^{eval}|. \tag{3.27}$$

Constraint (3.27) requires that every location from $\Lambda^{eval}$ has to be reachable by at least $N$ nodes from $V$. This requirement does not depend on the ranging technique and can guarantee a reliable coverage of the localization area.

**Example 14** (Reachable Anchors)**.** *Consider an indoor wireless localization network deployment on Figure 3.4, where yellow nodes represent network devices (anchors) and green dots are evaluation locations $\lambda_i \in \Lambda^{eval}$. A set of links is also shown with corresponding path loss values. Some anchors are mapped to library devices that have a 5 dB antenna. Transmission (TX) power is 0dBm for all devices. The system computes a 2D location using trilateration, therefore, at least 3 anchors must be reachable at a given location in order to provide the functionality. We impose that location $\lambda_i$ is reachable by a wireless node if the RSS at $\lambda_i$ is at least -80 dBm using Constraint* (3.26) *(assuming that $\lambda_i$ is a mobile receiver). With this requirement, $\lambda_1$ is reachable by 4 nodes. Location $\lambda_2$ is reachable by 3 nodes, while this is only possible with two of them having external antennas, which is a mapping decision. Only two nodes reach location $\lambda_3$, while $\lambda_4$ is unreachable by any node (even though one of the closest ones has an* *antenna). This sample deployment cannot guarantee the proper functionality of the designed localization system. Enforcing Constraint* (3.27) *would help to find a proper physical placement and mapping of anchor nodes in this system.*

Overall, the presented constraints do not explicitly manipulate the accuracy of the localization system. However, the accuracy is indirectly affected by these constraints, since they enforce the possibility of the target node to communicate with anchors via stable links with good RSS and/or SNR (using weak links leads to potential errors in distance estimation). The resulting node placement and mapping (e.g., using antennas or higher TX power on certain devices) will provide such guarantees. As future work, we plan to extend the list of localization constraints by supporting a set of metrics related the quality of trilateration, such as the ones from [127] and [111].

## 3.3 Mapping Specifications to Implementations

The mapping problem within the proposed architecture exploration methodology deals with selecting the library components that implement the "virtual" components of the template. As shown in Section 2.1.2, the mapping problem is encoded using a set of binary variables $m \in M$ so that $m_{ij} = 1$ if virtual component $v_j \in V$ is implemented by a library element $l_i \in \mathcal{L}$. A set of mapping constraints presented in Section 2.2.1 is added to basis constraints $\mathcal{R}_B$ to ensure the correctness of the mapping.

Our formulation of the mapping problem allows it to be separated from topology selection because a different set of variables ($m_{ij}$) is used for encoding it. At the same time, $m_{ij}$ are associated with the variables $e_{ij} \in E$ that define the system topology. However, with such formulation it is possible to solve the mapping problem for a fixed topology by replacing the variables $e_{ij}$ with some constant assignment. Similarly it is possible to perform topology selection with the mapping specified a-priori. That is, the two problems can be solved separately. This separation, as discussed in Section 2.2.1, results in an encoding that is more general than the one from [99, 12].

On the other hand, despite the possibility of separately selecting the topology and the mapping, both of them are parts of the exploration problem and are defined within the same basis. It is shown in Section 2.3.1 that attributes of the nodes of the template $\mathcal{T}$ are computed using the mapping variables $m_{ij} \in M$. At the same time, these attributes are used as arguments to compute system properties and impose application constraints on these properties, like it is demonstrated in Sections 2.3.2 and 3.2. Such interdependence allows for a joint selection and optimization of the two concerns. The solution for such optimization problem is guaranteed to be an optimal combination of the topology and the implementation as opposed to separately optimizing the two. This differs our approach from previous works [99, 12, 108], which select optimal system and network topologies assuming that the implementation characteristics of the nodes are known. For example, as further discussed in Section 6.4, Puggelli et al. in their MILP-based design methodology for wireless sensor networks assume that such device parameters as TX power and antenna gain are fixed. Conversely, by the virtue of the mapping constraints, in our formulation they become additional degrees of freedom of the network. This allows us to explore a much larger portion of the design space, where more cost-effective solutions may be found.

Previous works by Bajaj et al. [12] and Nuzzo et al. [99] solve only the topology selection problem, while the implementation of each component of the template (e.g., a generator with certain characteristics and cost) is hard-coded in the template $\mathcal{T}$. In general, their formulation allows encoding a mapping problem that is equivalent to the one proposed in this thesis, however, as discussed in Section 2.2.1, this would lead to a problem of a much higher complexity. The template $\mathcal{T}$ in these previous works would include nodes that represent particular implementations from the library $\mathcal{L}$. Therefore, the size of $\mathcal{T}$ would be at least $|\mathcal{L}|$ times larger compared to our approach, which would result in a significant growth of variables and constraints when defining application requirements, especially the complex ones, such as reliability. Mapping variables in our formulation introduce some overhead as well, but it is much smaller compared to increasing the size of $\mathcal{T}$. This makes our encoding more efficient with respect to previous work.

## 3.4 Solving and Analysis

Having defined the application requirements $\mathcal{R}_A$ (Section 3.2), the basis constraints $\mathcal{R}_B$ and the cost function $\mathcal{C}$ (Sections 2.2 and 2.3.3) we have all the constituents for casting the architecture exploration problem as an optimization problem as formalized in (2.10). The result of solving this problem is an assignment over the decision variable set $D = B \cup X$, where $B$ refers to variables from the basis (topology configuration, mapping and routing), while $X$ includes the auxiliary and application-specific variables. Below we describe two different techniques of using the MILP solver for finding an optimal assignment for $D$.

The first method, *monolithic* optimization (also called "eager"), simply uses all the optimization constraints from $\mathcal{R}_A$ and $\mathcal{R}_B$ and aims to solve a *single* problem, albeit of a potentially large size. Since some of the constraints may originate from approximations (e.g., reliability constraints discused in Section 3.2.4 or approximate encoding of network paths presented in Section 3.5.1), optimality is only guaranteed within the error bound due to the approximation. If there are no approximate encodings in the problem formulation, then the exact optimal solution is provided. Eager approach can be naturally used for solving any problem within our methodology. Its advantage is the global (possibly approximation-wise) optimality guarantee for the resulting architecture. The main drawback is the potentially high problem complexity, especially for large sizes of template $\mathcal{T}$ and library $\mathcal{L}$, and complex requirement encodings with lots of linearizations and auxiliary variables.

The second technique is an *iterative* (or "lazy") optimization procedure, initially proposed by P. Nuzzo in [96]. In Section 3.5.2 we provide a generalized version of the algorithm from [96], which can be customized for different exploration problems. Instead of formulating a large, "flat" optimization problem, it avoids the expensive generation and manipulation of certain classes of "heavy" constraints in the first place, by leveraging a coordination of specialized solvers inspired by the lazy ILP Modulo Theories [50, 80] or Satisfiability Modulo Theories (SMT) [15, 95] paradigms. Here, the MILP solver is called *iteratively* on smaller problem instances including only a subset of constraints from $\mathcal{R}_A$ (e.g., interconection constraints) to generate candidate architectures. The validity of these architectures is then checked against the remaining constraints using exact analysis methods. If these constraints are violated, a conflict-driven learning function is called between the iterations of the MILP-solver. This function incrementally adds new constraints to the original formulation based on the analysis of previous outcomes and generated counterexamples. Such technique allows pruning the search space and rapidly progressing towards a feasible solution. Solving a small number of simpler problem instances can significantly reduce the execution time with respect to a monolithic approach. However, global optimality is no longer guaranteed.

The learning functions used in the "lazy" approach implement some strategy (heuristic or exact) that performs a set of actions for guiding the solver towards the feasible portion of the design space. These actions may include adding or removing a connection, mapping some node or edge to a particular implementation, enforcing a route between the nodes, and others. The resulting outcome of the function, i.e., the learned constraints, guarantee that the solution of MILP at a next iteration refines the overall system quality, either improving a cost or a certain property being analyzed. On the whole, iterative optimization makes it easier to incorporate a domain-specific knowledge to the exploration problem, since a designer can customize the techniques adopted to improve the quality at each iteration. Another advantage is the high performance of the optimization that is achieved by solving small problems alternating with heuristic learning, as opposed to solving a large NP-hard problem. The absence of optimality guarantees as well as the sophistication of learning algorithms are primary drawbacks.

In the subsequent chapters both solving algorithms are evaluated on different case studies. In particular, in Chapter 5 we demonstrate that iterative approach provides results (in terms of system cost) that are competitive to the monolithic optimization with approximate reliability constraints, while the execution time is several orders of magnitude faster. In Chapter 6 we focus on the approximate encoding of path constraints from $\mathcal{R}_B$ to obtain more compact formulation of large-scale wireless network design problems that can still leverage the advances of the monolithic optimization and MILP overall. An approximate encoding algorithm for network paths is presented in Section 3.5.1.

Finally, the solution generated by either of the approaches is verified using one of the applicable exact analysis techniques, if any, so that designers are able to evaluate the quality of the architecture (apart from the value of the cost function). Moreover, these techniques are used in the "lazy" approach to verify intermediate solutions between the iterations. They compute some functional or non-functional properties. For example, *reliability analysis* calculates the *exact* failure probabilities of functional links in a system by recursively traversing the graph from the sink to sources of the link, evaluating the probabilities of individual failure events and joining them to obtain a total failure probability. A corresponding analysis algorithm has been proposed in [99]. The latency of system paths can be estimated using a static *timing analysis* (STA) routine, which is also a graph traversal algorithm. The information provided by STA can be used to determine critical paths, timing constraint violations as well as for other purposes. The *workload analysis* can be applied to compute the static load of system components and give understanding on how the flows are spread across the system. A *lifetime analysis* can be run to estimate, how long the system can perform its operation. For example, in a wireless network such estimation is related to the energy consumption and can be computed similarly to the corresponding requirement explained in Section 3.2.8.

## 3.5 Algorithms

In the following, we present two algorithms that allow designers to leverage the full power of our architecture exploration methodology, in particular, for large-scale problems and sophisticated design requirements. The first one replaces the exhaustive enumeration of all possible nodes and edges in the encoding of required system paths with a more compact, yet approximate, representation. This allows solving MILP for large systems (hundreds of nodes), while being competitive both to the exact formulation in terms of cost and to a heuristic algorithm in terms of performance. Moreover, the tradeoff between the two can be adjusted. Second algorithm is the generalized version of the MILP-MR (Mixed Integer Linear Programming Modulo Reliability) algorithm originally proposed in [12, 96], which implements the “lazy” solving method of the exploration problem.

### 3.5.1 Approximate Encoding of Network Paths

Every path $\pi$ from routing requirements can be encoded using $n^2$ variables from the set $y^\pi$ and $n$ variables from the set $w^\pi$, $n$ being equal to $|V|$, which correspond, respectively, to all the edges and nodes of the template $\mathcal{T}$. This encoding allows exhaustive exploration of network topologies, since any node and any edge may be a member of $\pi$, but becomes inefficient when either the size of $\mathcal{T}$ or the number of required paths increase. For every $\pi$ at least $n^2 + 5n$ constraints introduced in Section 2.2.2 are added to the optimization problem. Entries of $y^\pi$ and $w^\pi$ are further used in other network constraints (e.g., link quality or energy consumption constraints) and often multiplied by other decision variables. Each product of binary variables must be translated into a linear constraint by introducing auxiliary variables and constraints to the original non-linear formulation using one of linearization techniques from Section 2.4. All of these steps may result in a significant growth of problem size and solver time.

We propose to trade generality with complexity of the exploration problem by implementing an algorithm for a more compact, yet approximate, encoding of network paths. The main idea behind our method is to direct the search toward a smaller number of candidate alternatives instead of considering all possible nodes and edges from $\mathcal{T}$ in every required path. We assign a domain-specific weight criterion for graph edges and execute Yen’s K-shortest path algo-

**Algorithm 1:** Approximate path encoding

```
Given: Network template $\mathcal{T} = (V, E)$
Input: Set $Q$ of pairs $(s, d)$, weight matrix $W$, number of path candidates $K^*$
Output: Set $R = \{y^q | q \in Q\}$ of path variables, set $Cons$ of path constraints
1  $Cons \leftarrow [\,]$
2  $Q^+ \leftarrow$ FINDREPLICAS$(Q)$
3  forall $q \in Q^+$ do
4      $(K, N^{rep}) \leftarrow$ BREAKDOWN$(K^*)$
5      $(y_1^q, \ldots, y_{|E|}^q) \leftarrow 0$; $W' \leftarrow W$; $NewCons \leftarrow [\,]$
6      for $n = 1$ to $N^{rep}$ do
7          $(p_1, \ldots, p_K) \leftarrow$ KSHORTEST$(W', s_q, d_q, K)$
8          for $k = 1$ to $K$ do
9              $v \leftarrow$ GETVARIABLES$(p_k)$
10             $y^q \leftarrow$ ADDVARIABLES$(v)$
11             $NewCons \leftarrow NewCons \vee \bigwedge_{i=1}^{|v|} v_i$
12         $W' \leftarrow$ REMOVEMINDISJOINTPATH$(W', (p_1, \ldots, p_K))$
13     $R \leftarrow R \cup y^q$
14     $Cons \leftarrow Cons \cup NewCons$
15 return $(R, Cons)$
```

rithm [128] to select a number $K^*$ of the path candidates for every network route specified in the requirements. We then symbolically encode the proposed paths using a smaller number of edge variables to obtain the final path constraint, as summarized in Algorithm 1.

The function FINDREPLICAS$(Q)$ extends the input set $Q$ to a set $Q^+$ with a number of copies of each source-destination pair $(s, d)$ corresponding to the required amount of replicas (redundant paths) for this pair. By analyzing the routing requirements for a pair $q \in Q^+$, the function BREAKDOWN$(K^*)$ splits the required number of candidate paths $K^*$ into $N^{rep}$, the required number of *disjoint* replicas for $q$, and $K$, the required number of candidate paths for each replica, such that $N^{rep} \cdot K \geq K^*$ (line 4). Then, $q$ is associated with a vector $y^q$, where $|y^q| = |E|$, and $K^*$ candidate paths are generated for $q$ as follows (lines 6-13). KSHORTEST runs Yen's K-shortest path routine to generate K "best" paths $p_1 \ldots p_K$ in non-decreasing order of cost (line 7), by using the matrix $W$ to assign weights to edges. Every generated path $p_k$ is then processed and a binary variable is assigned to every edge between the nodes of $p_k$ (line 9). The vector $v$ of these variables is added to $y^q$ (line 10). Also, the path constraint $NewCons$ is updated (line 11). It requires that one of the proposed paths has to be selected in the final topology. The path generation procedure above is repeated $N^{rep}$ times. At each iteration, the function DISCONNECTMINDISJOINTPATH identifies a path, which has maximum number of edges in common with other paths, i.e., it is minimally disjoint from others. This path is disconnected from the graph, so that the following iteration will generate at least one path that is completely independent from the previous ones (in practice, restricting the most used edges leads to larger amount of disjoint replicas). This ensures that at least $N^{rep}$ of the $K^*$ proposed paths will be disjoint, as per the routing requirement. Disconnecting a path can be done by manipulating the weights of corresponding edges in a way that the shortest path routine does not consider them on the next call (e.g., by setting their values to infinity). The process is repeated for every $q$ until all path candidates and corresponding constraints are generated, and then the algorithm terminates.

Depending on the concrete problem, entries of the weight matrix $W$, i.e., the weights assigned to graph edges, can be associated to different properties of the system or the environment. For example, in a wireless network they can represent path loss values or signal-to-noise ratios of corresponding links. Moreover, $W$ can be preprocessed to restrict certain connections according to composition rules of the platform library or some design requirements (e.g., "weak" wireless links with large path loss can be discarded). This is done to guarantee that all selected path candidates satisfy other concerns of the specification, such as link quality of network routes.

The worst case number of path variables $y^{\pi}$ needed for every required route is $K^*(n-1)$, rather than $n^2$, assuming that every new path consists of $n = |V|$ nodes and all $K^*$ paths are disjoint. However, the situation is much better in practice, since realistic network paths typically contain only few hops and share common links. Moreover, path constraints (2.2), (2.4a)-(2.4b) can be omitted since the validity of generated paths is guaranteed by the shortest path routine. Further reduction is also achieved in other constraints that use the path variables from the basis, because the latter have to be defined only for nodes and edges which are members of some candidate path. $K^*$ controls the gap between solutions obtained with and without the approximation and can be adjusted to trade optimality with execution time.

Finally, the proposed path encoding algorithm is general and can be applied to any weighted directed graph model, independently of the specific application domain. We run the experimental evaluation of Algorithm 1 on a wireless sensor network case study in Chapter 6. Results confirm that using approximate path encoding significantly improves the scalability of the exploration problem and decrease the complexity and execution time by orders of magnitude. The effect of manipulating the parameter $K^*$ is also studied to explore the tradeoff between optimality and complexity provided by the algorithm.

### 3.5.2 Iterative Optimization and Learning

An iterative MILP-based optimization technique has first been proposed by Hang et al. in [50] and later adopted by P. Nuzzo et al. in [96, 12] as a MILP Modulo Reliability (MILP-MR) algorithm for synthesizing reliable and cost-effective CPS architectures. In Algorithm 2 we generalize MILP-MR with respect to our generic formulation of the CPS architecture exploration problem. It can then be customized for iterative optimization and design exploration of CPS under different concerns.

The exploration problem is defined within the basis $B$ introduced in Section 2 with possible assistance of domain-specific discrete or continuous decision variables $X$ (e.g., flow rates). At each iteration the generated architecture $G^*$ will be verified against the requirement (property) $r^*$ with $r$ being the value of this property in $G^*$. First, $r$ is assigned with initial value (line 1) such that it does not satisfy the requirement (greater, less or not equal to $r^*$ depending on the

**Algorithm 2:** Iterative optimization

```
Given: Basis $B$ of the exploration problem, set $X$ of domain-specific decision variables
Input: Network template $\mathcal{T} = (V, E)$, library $\mathcal{L}$, requirement $r^*$, set $\mathcal{A}$ of component attributes
       (excluding the ones related to $r^*$)
Output: Final architecture $G^*$, i.e., topology configuration $E^*$, mapping $M^*$ and routing $R^*$
1  $r \leftarrow$ InitRequirement($r^*$)
2  ($\mathcal{C}$, $\mathcal{R}_A$, $\mathcal{R}_B$) $\leftarrow$ GenMILP($\mathcal{T}$, $\mathcal{L}$, $\mathcal{A}$)
3  while Verify($r, r^*$) = Unsat do
4      $G^* \leftarrow$ SolveMILP($\mathcal{C}$, $\mathcal{R}_A$, $\mathcal{R}_B$)
5      if $G^*$ = [ ] then
6          return Infeasible
7      $r \leftarrow$ RunAnalysis($G^*$)
8      if Verify($r, r^*$) = Unsat then
9          $\sigma \leftarrow$ GetCounterExample($r, r^*, G^*$)
10         $Cons \leftarrow$ LearnCons($\sigma, G^*$)
11     if $Cons$ = [ ] then
12         return Infeasible
13     $\mathcal{R}_A \leftarrow \mathcal{R}_A \cup Cons$
14 return $G^*$
```

concrete problem). The MILP formulation of the exploration problem is then generated as a combination of cost function $\mathcal{C}$, application constraints $\mathcal{R}_A$ and basis constraints $\mathcal{R}_B$. We note that the generated MILP problem is smaller than the monolithic one, which includes all possible constraints, because constraints on the properties related to $r^*$ (e.g., reliability) are not added to the formulation, i.e., $\mathcal{A}$ does not include corresponding component attributes (e.g., failure probabilities).

The MILP problem is then solved in a loop with analysis and learning routines (lines 3-13). SolveMILP generates minimum cost architectures $G^*$ (line 4). The RunAnalysis routine (line 7) (e.g., reliability analysis, timing analysis) evaluates the solution and computes the current value of the property $r$, which is then compared with the requirement $r^*$ (line 8). If the candidate architecture does not satisfy $r^*$ (`Unsat` is returned by Verify), then a counterexample $\sigma$ is generated (line 9), i.e., a property in a certain scope (e.g., reliability of a particular functional link, latency of some path) that violates $r^*$. The learning function LearnCons then uses $\sigma$ to generate a set of additional MILP constraints that augment the original optimization problem (line 10). This is done by suggesting a set of strategies for improving the existing architecture and guiding the solver towards a feasible one. LearnCons is, therefore, instrumental to efficiently converge towards a satisfying assignment over the decision variables, while minimizing the number of calls to RunAnalysis.

The overall result is, in general, sub-optimal with respect to the monolithic formulation but the satisfaction of the requirement $r^*$ is guaranteed by the exact analysis routine. At the same time, Algorithm 2 can terminate with `Infeasible` when either LearnCons or SolveMILP terminates with `Infeasible`. In the former case, the learning function is no more able to generate additional constraints and, therefore, we infer that no more moves can be made to improve the current architecture, while the requirement $r^*$ is still not satisfied. For example, no more additional network paths can be proposed to increase the reliability of a functional link. In the latter case, we infer that SOLVEMILP fails to find a feasible assignment to the problem, which means either that the original formulation is inconsistent or some constraint recently generated by LEARNCONS is incompatible with existing ones. The rigorous proof of correctness of the algorithm directly follows the same proof for the MILP Modulo Reliability (MILP-MR) algorithm and can be found in [96].

# 4 ARCHEX 2.0: Architecture Exploration Framework

*In this chapter we introduce* ARCHEX 2.0*, an extensible optimization-based framework for cyber-physical system architecture exploration that supports all steps of the proposed methodology. It allows exploration problems to be efficiently formulated and solved as MILP optimization problems by using the algorithms presented in Chapter 3. The software structure of* ARCHEX 2.0 *is modular and amenable to design reuse. We provide a high-level overview of the framework outlining its main components and classes. We also present an extension for wireless network topology design that provides handling of plans and maps of the deployment area, encodings for domain-specific requirements (e.g., link quality) and several channel models for analytical estimation of the path loss of wireless links. We then discuss the requirement patterns that lower the effort of problem formulation by automatically creating the MILP constraints from the specification. These patterns implement the constraints presented in Section 3.2 and thus allow us to capture a variety of design concerns. Finally, we give a short overview of using the framework and exemplify the main steps to be taken by the designer.*

## 4.1 Overview

So far we have established a computational framework for representing cyber-physical system architectures as networks of components that can be implemented by elements taken from domain-specific libraries. One of the early design stages, the *architecture exploration*, deals with selecting a feasible (and, possibly, optimal) architecture from a large space of candidates. The theoretical part of Chapter 2 includes the formulation of such exploration problems as optimization problems. In Chapter 3 we have developed this theoretical background into an optimization-based methodology, which includes writing design specifications, generating their mixed integer linear formulations, solving optimization problems and analyzing the obtained architectures. For applying the methodology to a certain CPS domain, it has to be instrumented with supporting tools that implement the proposed encoding and solving algorithms, and enable efficient co-design with correctness guarantees.

Instead of a domain-specific design tool, in this thesis we opt for an architecture exploration *framework*, i.e., a set of coherent functionalities that can be selectively adapted to different application domains and requirements by reusing and extending them. Therefore, we propose

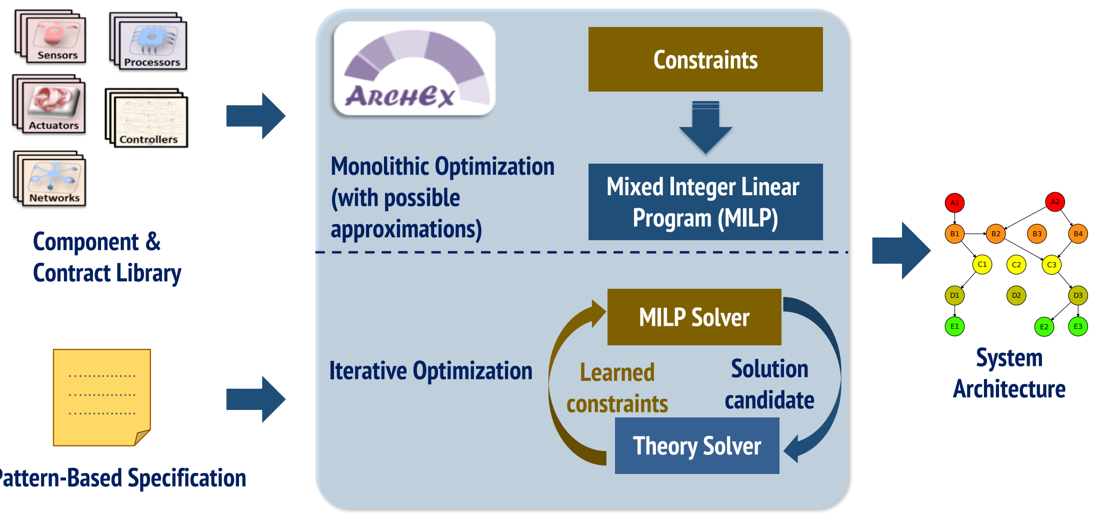


Figure 4.1 – Overview of the ARCHEX 2.0 framework.

ARCHEX 2.0 [62] as a design artifact for formulating and solving architecture exploration problems using a set of generic requirement patterns. It relies on a set of abstract classes and reusable data structures. A developer can customize them to implement new patterns, which can support broader categories of CPS designs, or implement new, possibly more specific, classes to solve particular problems. Initial version of ARCHEX has been proposed in [12] as a prototype toolbox for exploration and synthesis of reliable and cost-effective architectures of aircraft electrical power systems. We have amended the previous version by completely re-engineering the software infrastructure and generalizing existing concepts (e.g., reliability constraints) so that they can be applied to different CPS domains. Also, as discussed in Section 3.3, the mapping problem has been separated from the topology selection problem. Overall, ARCHEX 2.0[1] fully conforms to the theory presented in this thesis: exploration problems are defined within the generic basis presented in Chapter 2. Moreover, we have implemented a large number of patterns corresponding to requirements presented in Section 3.2 (a comprehensive list of existing patterns is shown in Section 4.2) and provided a significant extension for wireless network design. Furthermore, the pattern language can now be used to write specifications as text files, instead of using them in the application code.

The basic overview of the ARCHEX framework is shown on Figure 4.1. The tool accepts the design specification and the library of components and contracts as inputs and then internally translates them to MILP constraints that form an optimization problem. The generated formulation is either solved in full ("eager" approach discussed in Section 3.4) or iteratively with current optimization results being verified by a theory solver (e.g., reliability) that attempts to improve the solution by proposing additional constraints to the original problem. The latter "lazy" technique and the corresponding algorithm have been discussed in Section 3.5.2. Approximate encodings, such as reliability constraints [12, 96] or path constraints (see Sec-

[1] In the following, the version number (2.0) is sometimes omitted for brevity.

tion 3.5.1 and Algorithm 1) may be used to generate monolithic problem formulations that are tractable. That is, the result of the "eager" optimization may be optimal with respect to an approximation of the constraints. In contrast, the "lazy" approach aims at avoiding heavy constraints and approximations in the formulation, while still accounting for them via exact analysis of the architecture and gradually progressing towards a feasible solution using learning functions. In general, this has much smaller complexity, but does not provide guarantees that the obtained architecture is optimal with respect to initial objective.

ARCHEX 2.0 is offered as a MATLAB [82] toolbox and is available online [1]. It currently uses CPLEX [54] as a MILP solver and YALMIP [76] as an intermediate interface that facilitates the problem formulation. The execution flow of ARCHEX together with the main components and methods is illustrated on Figure 4.2 (formulation and solving flows are shown with, respectively, semi-transparent blue and green arrows). The input to the toolbox consists of two text files: specification and library. The specification file includes several parts:

- General information about the system (component types, functional flows) and the exploration problem (e.g., name, description, cost of an edge etc).
- Structure of the template, i.e., the maximum number of components of each type.
- Composition rules between the system components that allow/restrict certain connections in the architecture template.
- Problem-specific parameters that are system-wide, e.g., parameters of a communication protocol, environment characteristics (temperature, pressure, humidity) and others.
- Application requirements in terms of patterns.

As we demonstrate in the following chapters, by the virtue of using requirement patterns, the size of the specification is kept small (less than hundred rows even for large problems), while the underlying MILP formulation generated by ARCHEX can be several orders of magnitude larger (thousands to millions of constraints and variables).

The library is organized as a list of records grouped by component types. Each record represents a distinct component, and includes a name and a list of attributes belonging to current (or each) type. Several records can represent the same physical device with different configuration parameters, e.g., a radio transmitter with different TX powers. Therefore, component sizing can be similarly performed in two ways: selecting a physical device that best implements the "virtual" component of the template, and exploring a discrete space of configuration parameters of a chosen implementation. Components in the library can also be labeled with subtypes. For instance, power sources can be partitioned into low- and high-voltage sources while still being of the same type, e.g., AC or DC. Such sub-classification is useful, in particular, for defining interconnection constraints on the architecture, as discussed in Section 3.2.2.

Furthermore, components having the same type and sub-type may be grouped using *tags* based on the problem domain. For instance, as shown in Section 5.1.2, we distinguish between

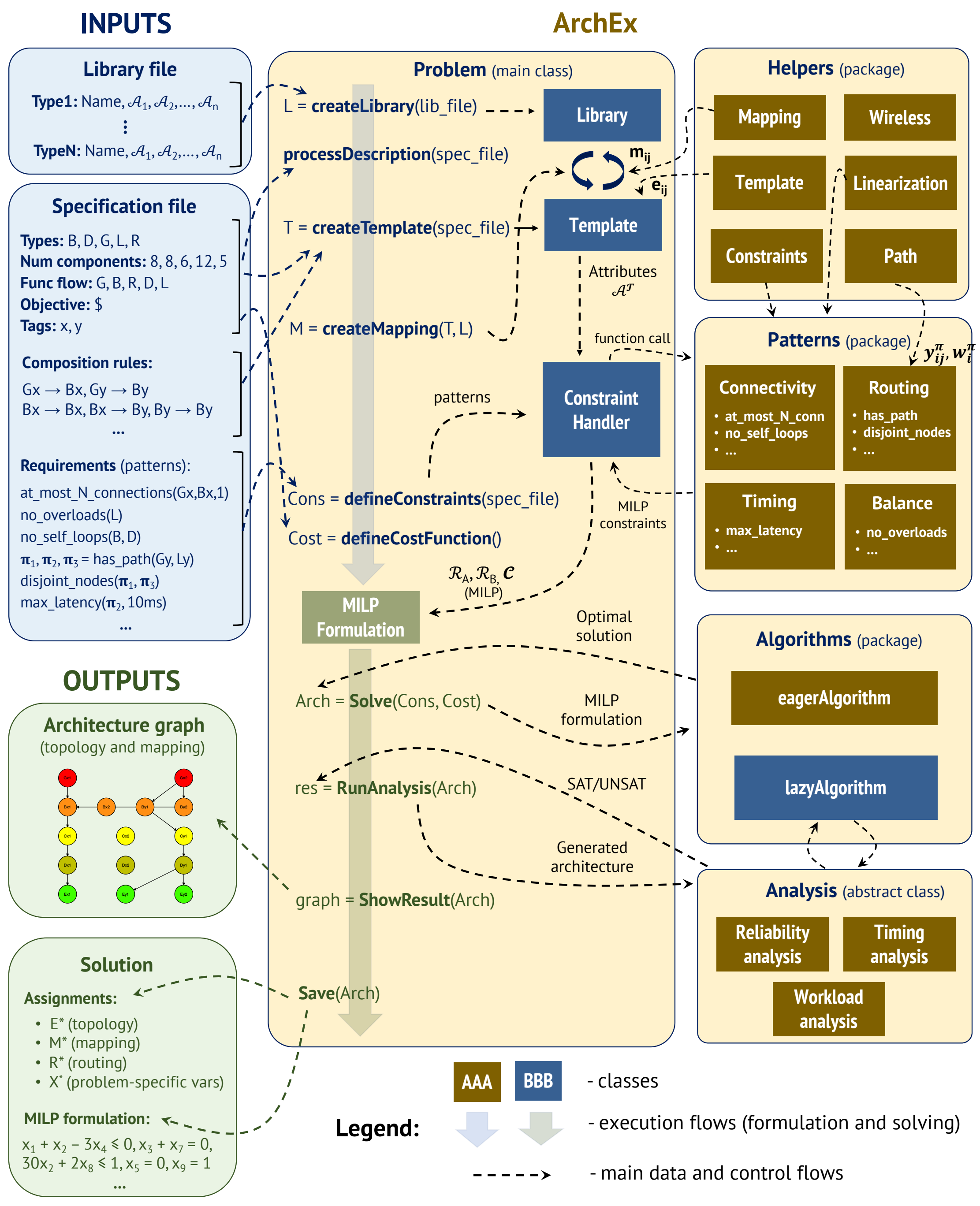


Figure 4.2 – Execution flow of ArchEx: inputs, outputs, sequence of main method calls and related dependencies.

left and right AC buses in the aircraft electrical power network, based on their location, as this poses restrictions on the feasible connections. Therefore, tags represent an additional, problem-specific dimension of classification, and can be used in the patterns together with types and subtypes.

The class *Problem* implements the core functionality of the framework. It is an abstract class that represents the generic exploration problem and has to be inherited by a child class for a particular CPS domain as further discussed in Section 4.3.1. The constructor of the Problem class takes specification and library filenames as arguments and executes a sequence of methods (shown top-down on Figure 4.2) for parsing them and initializing the library and the architecture template, the mapping, the constraints (application and basis) and the objective function. Overall, the whole problem formulation is performed when the object is created.

The *Helpers* package includes a set of static classes, i.e., collections of helper methods, that facilitate the encoding of the exploration problem to a mixed integer linear program. For example, classes *Template*, *Mapping* and *Path* are responsible for encoding the basis of the exploration problem introduced in Chapter 2, and provide corresponding variables ($e_{ij}$, $m_{ij}$, $y^{\pi}_{ij}$ and $w^{\pi}_{i}$) and basis constraints (mapping and path).

Application requirements are read from the specification and classified by type (e.g., routing constraints, timing constraints). The *Patterns* package is a collection of methods that implement the corresponding requirement patterns. When called from the Problem class, they return the MILP formulations of the constraints. They are stored together with the cost function $\mathcal{C}$ defined using expressions (2.7) and (2.8). Patterns can also be invoked interactively by the user if he/she wants to augment the specification without recreating the problem, which lowers the effort and the time for debugging large problem instances.

After the problem instance is created, user can run the optimization using one of the two aforementioned algorithms (“eager” is the default one, while the “lazy” can also be used if a learning function is implemented for the given domain). This is done by calling the `Solve(Cons,Cost)` command of the Problem class, where `Cons` includes the application and basis constraints $\mathcal{R}_A$ and $\mathcal{R}_B$, while `Cost` is the objective function $\mathcal{C}$. After a solution is generated, it is verified against a certain criterion using one of the analysis classes (e.g., reliability, timing, workload). The latter is implemented as a child of the abstract *Analysis* class. As shown in Algorithm 2, iterative optimization scheme also uses analysis methods in a loop with learning constraints and generating new architectures.

The graph representation of the final architecture is shown and saved to disk. Every node in the graph also stores the information about its selected implementation (mapping). Finally, all input and output data, including specification, generated MILP formulation (constraints) and solution (assignments over all decision variables, i.e., basis, problem-specific, auxiliary) is also saved for future reference. Visualization may also include problem specific views, graph layouts and auxiliary information (e.g., floor plan, transmission coverage and so on).

## 4.2 Pattern-Based Requirement Specification

System requirements in specifications are typically expressed in a natural language, e.g., in English. It is challenging to establish and follow a set of structural rules for such expressions so that they could be easily recognized by a design automation tool and translated into the underlying formalism, such as MILP. The reasons include, but are not limited to, word ordering, grammatical and spelling errors, wrong semantics. On the other hand, creating specifications in high-level programming languages, or mathematical formulas that capture groups of requirements, or even at a lower level, i.e., constraint by constraint, can be extremely complicated. First, MILP formulations of CPS architecture exploration problems, even of a small size, may include thousands of constraints and variables (as well as other formalisms, such as SAT/SMT). This soon becomes very time consuming. Moreover, the resulting specification is difficult to review and debug. Second, along with the system domain expertise, designers are required to have significant knowledge of the mathematical formalisms and methods used by particular tools, as well as programming skills.

In this work, we propose *requirement patterns*, an intermediate level between the natural language and the underlying mixed integer linear formulation. Figure 4.3 illustrates the approach. ARCHEX allows the users (designers) to operate on a level of a pattern-based formal language, thus hiding the complex details of the actual MILP representation of the exploration problem. This significantly lowers the burden of problem formulation, because all underlying translations and definitions are done automatically by the tool.

An ARCHEX pattern has a name that reflects the associated requirement and a list of arguments, e.g., the component types or paths to which it applies. Each pattern is used to automatically generate MILP constraints over the input arguments, operating on corresponding subsets of decision variables. The access to the actual problem variables and the internal data structures is transparent to the user, which makes it easier to formulate and solve exploration problems. A system developer can then leverage these higher-level primitives to encode a problem, rather than manually generating the underlying optimization constraints, which is a tedious, error-prone task, often requiring the touch of an optimization expert. In this way, patterns can also contribute to reducing the chances of errors, and therefore the debugging effort, by virtue of their abstract nature.

Table 4.1 shows the list of patterns currently supported by ARCHEX. They are grouped by several categories reflecting corresponding classes of system requirements (e.g., interconnection, balance, timing). Each pattern is an intuitive abbreviation of a requirement, which is close to a natural-language expression but still preserves a formal semantics. For example, to express that "there must be at least one connection between components of type A and components of type B", one can use the pattern `at_least_n_connections(A,B,1)`, which is later translated into a constraint as in (3.2a). Similarly, `in_conn_implies_out_conn` is used to encode balance constraints on the component connections (edges): if there is a certain type of incoming edge then a certain type of outgoing edge must be present. In some patterns there are optional arguments, such as subtypes, that can be omitted. Such arguments are

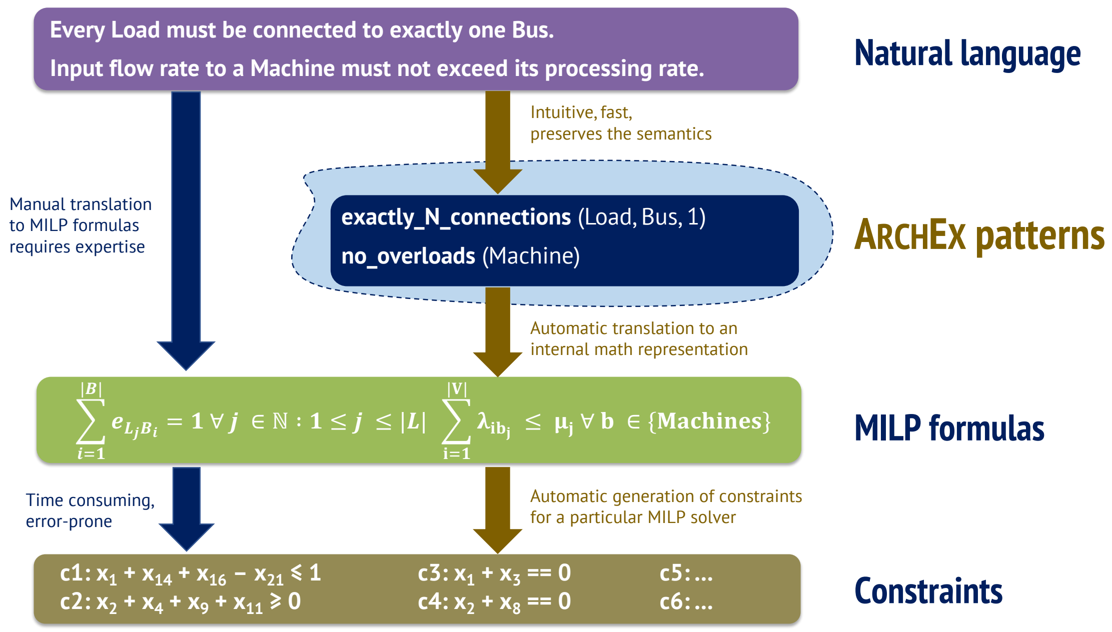

Every Load must be connected to exactly one Bus.
Input flow rate to a Machine must not exceed its processing rate.
Natural language
Intuitive, fast, preserves the semantics
Manual translation to MILP formulas requires expertise
exactly_N_connections (Load, Bus, 1)
no_overloads (Machine)
ARCHEX patterns
Automatic translation to an internal math representation
$\sum_{i=1}^{|B|} e_{L_j B_i} = 1 \; \forall j \in \mathbb{N} : 1 \le j \le |L|$
$\sum_{i=1}^{|V|} \lambda_{ib_j} \le \mu_j \; \forall b \in \{\text{Machines}\}$
MILP formulas
Time consuming, error-prone
Automatic generation of constraints for a particular MILP solver
c1: $x_1 + x_{14} + x_{16} - x_{21} \leqslant 1$
c2: $x_2 + x_4 + x_9 + x_{11} \geqslant 0$
c3: $x_1 + x_3 == 0$
c4: $x_2 + x_8 == 0$
c5: ...
c6: ...
Constraints

Figure 4.3 – Pattern-based formal language as an intermediate level between requirements expressed in a natural language and the underlying MILP representation.

marked with prime (e.g., S′).

Mapping patterns include the ones used internally to encode the mapping constraints from the basis of the exploration problem. Along with that, the pattern `in_flow_implies_mapping_to` is useful in some specifications to impose a particular implementation of a component if it accepts certain type of quantity as input. For example, in a production line, if a machine that processes the product type $A$ also receives units of the product $B$, then it must be reconfigurable, i.e., mapped to one of the elements from the library that supports reconfigurability during manufacturing.

The underlying MILP formulations hidden by patterns can be very sophisticated. For example, `min_network_lifetime` replaces a set of complex constraints introduced in Section 3.2.8, which, together with auxiliary linearization constraints, amounts to thousands of inequalities. At the same time, with this pattern the lifetime requirement can be imposed for the whole network using a single line in the specification file. The difference between the complexity of the specification and the generated MILP formulation rapidly grows with the problem size, and achieves orders of magnitude even for small instances. Therefore, our pattern-based language significantly improves the usability of the framework, which is further demonstrated in the following chapters with experimental results.

| Pattern name | Description |
|---|---|
| NUMBER OF COMPONENTS | |
| `at_least_N_components(T,S',N)`<br>`at_most_N_components(T,S',N)`<br>`exactly_N_components(T,S',N)` | There must be *at least* [*at most, exactly*] `N` component(s) of type `T` with subtype `S`. |
| INTERCONNECTION | |
| `at_least_N_connections`$(T_1,T_2,N)$<br>`at_most_N_connections`$(T_1,T_2,N)$<br>`exactly_N_connections`$(T_1,T_2,N)$ | Every component of type $T_1$ must be connected to *at least* [*at most, exactly*] `N` components of type `T2`. |
| `at_least_N_connections_if_used`$(T_1,T_2,N)$<br>`exactly_N_connections_if_used`$(T_1,T_2,N)$ | Every component of type $T_1$ that is *used* (instantiated) must be connected to *at least* [*exactly*] `N` components of type $T_2$ (this does not force the unused nodes to have edges). |
| `in_conn_implies_out_conn`$(T,T_{in},T_{out})$<br>`no_in_conn_implies_out_conn`$(T,T_{in},T_{out})$<br>`in_conn_implies_no_out_conn`$(T,T_{in},T_{out})$<br>`no_in_conn_implies_no_out_conn`$(T,T_{in},T_{out})$ | If a component of type `T` *has* [*does not have*] an incoming connection from a component of type $T_{in}$, then it *must* [*not*] have an outgoing connection to a component of type $T_{out}$. |
| `out_conn_implies_in_conn`$(T,T_{out},T_{in})$<br>`no_out_conn_implies_in_conn`$(T,T_{out},T_{in})$<br>`out_conn_implies_no_in_conn`$(T,T_{out},T_{in})$<br>`no_out_conn_implies_no_in_conn`$(T,T_{out},T_{in})$ | If a component of type `T` *has* [*does not have*] an outgoing connection to a component of type $T_{out}$, then it *must* [*not*] have an incoming connection from a component of type $T_{in}$. |
| `bidirectional_connection`$(T_1,T_2)$<br>`no_bidirectional_connection`$(T_1,T_2)$ | Connections between components of type $T_1$ and $T_2$ *must* [*not*] be bidirectional (if node *a* of $T_1$ has an edge to node *b* of $T_2$, then *b must* [ *not*] have an edge to *a*). |
| `cannot_connect`$(T_1,S_1,T_2,S_2)$ | Components of type $T_1$ that have subtype $S_1$ cannot be connected to components of type $T_2$ that have subtype $S_2$. |
| PATH AND ROUTING | |
| `p = has_path(src,dst)` | There must be a path between the nodes `src` and `dst`. The path has a symbolic name `p` that can be later used in other patterns in the specification to refer to current path. |
| `disjoint_edges`$(p_1,p_2)$ | Paths $p_1$ and $p_2$ must have disjoint edges. |

| | |
|---|---|
| disjoint_nodes($p_1$,$p_2$) | Paths $p_1$ and $p_2$ must have disjoint nodes (except for the source and the sink). |
| at_least_N_differences($p_1$,$p_2$,N)<br>at_most_N_differences($p_1$,$p_2$,N)<br>exactly_N_differences($p_1$,$p_2$,N) | Paths $p_1$ and $p_2$ must have *at least* [*at most,exactly*] N distinct nodes (from each other). |
| min_nodes_in_path(p,N)<br>max_nodes_in_path(p,N)<br>exact_nodes_in_path(p,N) | There must be *at least* [*at most,exactly*] N nodes in the path p. |
| min_hops_in_path(p,N)<br>max_hops_in_path(p,N)<br>exact_hops_in_path(p,N) | There must be *at least* [*at most,exactly*] N hops (edges) in the path p. |

BALANCE AND WORKLOAD

| | |
|---|---|
| flow_balance(T,S$'$) | The input flow into components of type T and subtype S must be equal to the output flow from these components (fan in equals fan out). This is a strict version of Constraints (3.6)-(3.7). |
| has_sufficient_input($T_1$,$S'_1$,$T_2$,$S'_2$) | Components of type $T_1$ and subtype $S_1$ must provide sufficient quantity (e.g., power, fuel, air) to supply all the components of type $T_2$ and subtype $S_2$. This is a non-strict version of Constraints (3.6)-(3.7). |
| no_overloads(T,S$'$) | Components of type T and subtype S must not be overloaded (input flow must be less than or equal to component's throughput). |

RELIABILITY

| | |
|---|---|
| min_redundant_components(T,S$'$,N) | There must be at least N redundant components of type T and subtype S (minimal degree of redundancy). |
| max_failprob_of_connection($T_1$,$S'_1$,$T_2$,$S'_2$,val) | The failure probability of all functional links between components of type $T_1$ and $T_2$ must not exceed val. |

TIMING

| | |
|---|---|
| max_latency_of_path(p,val,units) | The latency (delay) of the path p must not exceed val units (e.g., seconds, minutes). |
| max_latency_of_component(T,S$'$,val,units) | The latency (delay) of components of type T and subtype S must not exceed val units (e.g., seconds, minutes). |

| | |
|---|---|
| `max_total_idle_rate(T,val,units)` | The total idle rate of components of type `T` (e.g., processing machines, servers) must not exceed `val units` (e.g., parts/min). |
| `in_flow_implies_max_latency(F,T,val,units)` <br> `in_flow_implies_max_latency(F,p,val,units)` | If the input flow rate to *components of type* `T` [*path* `p`] is `F` (e.g., msg/sec), then the total latency of these components [this path] must not exceed `val units` (e.g., seconds, minutes). |

MAPPING

| | |
|---|---|
| `at_most_one_mapping(T)` | Componets of type `T` can be mapped to at most one element in the platform library. |
| `if_conn_then_has_mapping_strict(T)` <br> `if_conn_then_has_mapping_soft(T)` | If component of type `T` is connected, then it must have a mapping to the library. If it is not connected, then it *must not* [*may*] be mapped. |
| `in_flow_implies_mapping_to(`$T_{in}$`,`$S_{in}$`,T,S)` <br> `in_flow_implies_no_mapping_to(`$T_{in}$`,`$S_{in}$`,T,S)` | If component of type `T` has an input flow from a component of type $T_{in}$ with subtype $S_{in}$, then it *must* [*not*] be mapped to a library element with subtype `S`. |

LINK QUALITY (wireless)

| | |
|---|---|
| `min_received_sig_strength(p,val)` | The RSS of every link along the path `p` must be at least `val` dBm (`p` can also be a single link). |
| `min_signal_to_noise(p,val)` | The SNR of every link along the path `p` must be at least `val` dB (`p` can also be a single link). |
| `max_bit_error_rate(p,val)` | The BER of every link along the path `p` must be at most `val` % (`p` can also be a single link). |
| `max_pkt_error_rate(p,val)` | The PER of every link along the path `p` must be at most `val` % (`p` can also be a single link). |
| `min_pkt_delivery_ratio(p,val)` | The packet delivery ratio of every link along the path `p` must be at least `val` % (`p` can also be a single link). |
| `max_retransmissions(p,val)` | The maximum number of `val` retransmissions is allowed for every link of the path `p` (ETX constraint). |

ENERGY CONSUMPTION (wireless)

| | |
|---|---|
| `min_network_lifetime(val,units)` <br> `min_node_lifetime(V,val,units)` | The lifetime of the *network* [*node* `V`] must be at least `val units`, e.g., days or years (first pattern enforces the constraint on all nodes). |

| | |
|---|---|
| `max_energy_per_period(V,val,t,units)` | The energy consumed by the component `V` within every period of `t units` must not exceed `val` Joules. |
| **LOCALIZATION (wireless)** | |
| `min_reachable_devices(loc,N,metric,val)` | Every location in the set `loc` must be reachable by at least `N` devices, such that the LQ `metric` (e.g., “RSS” or “ETX”) of corresponding links does not go beyond (or below) `val`. |

Table 4.1 – List of requirement patterns supported by ARCHEX 2.0 grouped by type. Primed arguments (e.g., $S'$) are optional.

The patterns in Table 4.1 cover the categories of constraints in Section 3.2 and can be reused across domains of applications, which is an indication of their power to capture the essence of the problems of interest. Finally, we note that the presented list of patterns can be incrementally extended to create more expressive, domain-specific languages based on simpler primitives.

## 4.3 Structure of the Toolbox

### 4.3.1 Main Components and Data Structures

ARCHEX has a modular, object-oriented organization aiming both at increasing the usability of the framework for end users, and for lowering the complexity of extending it and integrating new patterns, exploration problems and domain-specific features into existing infrastructure. As stated before, all problems applicable for the proposed architecture exploration methodology are defined within the same basis of decision variables and constraints. Therefore, most ARCHEX classes and data structures that are responsible for problem formulation are implemented in a highly reusable fashion. This makes the framework more general with respect to possible tools that support the proposed methodology within a single selected domain of CPS.

A simplified class diagram of ARCHEX is shown on Figure 4.4. Some relations are labeled with the entities provided by one class to another, e.g., variables or constraints. As mentioned in Section 4.1, core functionalities are implemented in a set of helper classes (Helper and Patterns packages), while the user creates and controls the *Problem* class. It is an abstract class that defines a set of properties (e.g., structures for storing template information, composition rules, functional flows) and generic routines (e.g., reading the specification, managing inputs and outputs) that are common for any exploration problem. The class also includes a set of virtual methods (declarations) that must be implemented by every child class that corresponds to a particular type (or domain) of problems. These methods include specifying the library, the template and the association between them, handling the constraints and the cost function, solving the optimization problem, visualizing and saving the results.

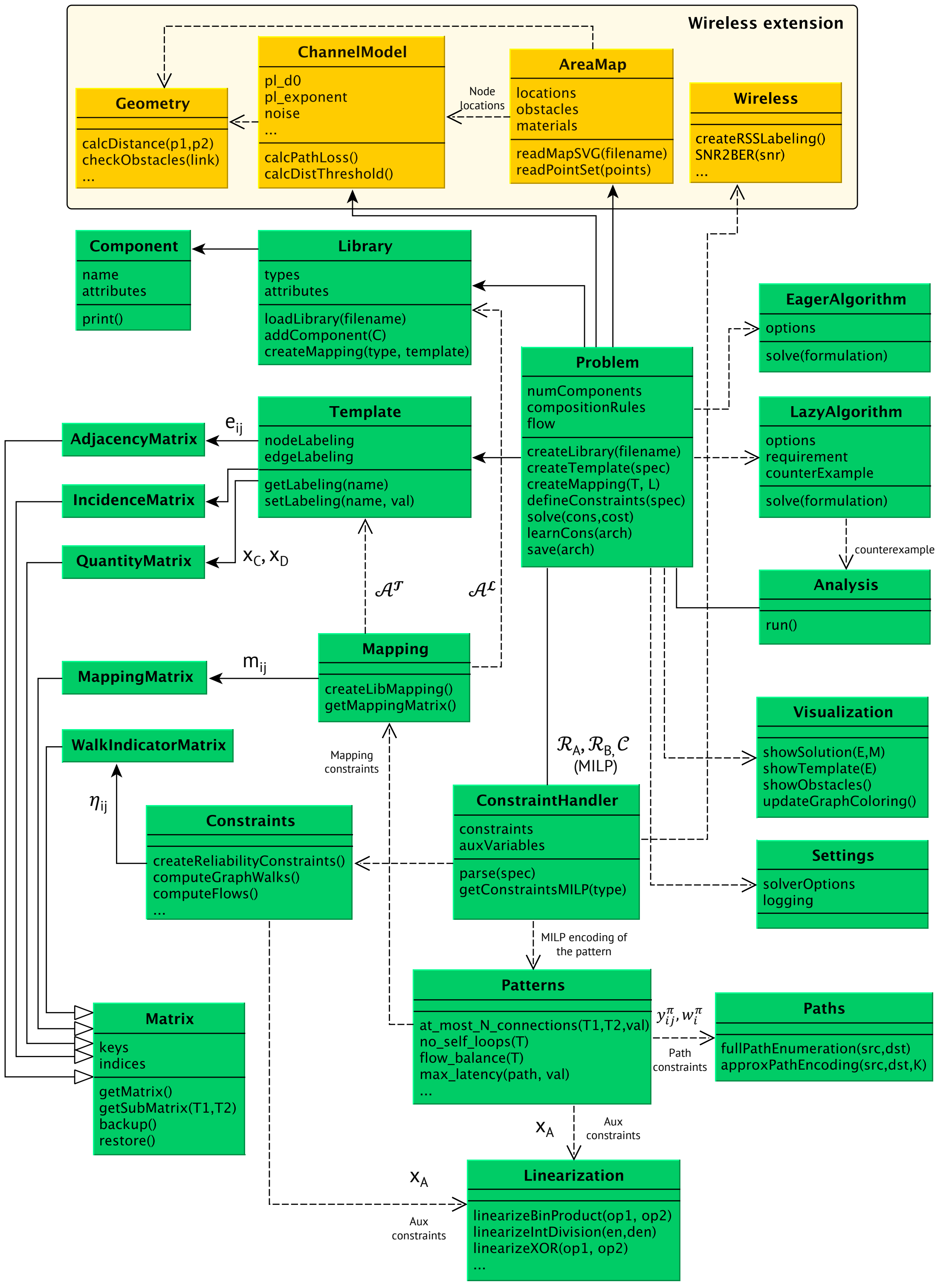


Figure 4.4 – Simplified class diagram of ARCHEX highlighting the relations between the main components of the framework. Classes shown in yellow have been implemented as an extension of the toolbox for wireless network design problems.

The *Component* class represents a generic system component with several attributes, e.g., type, subtype, cost. The class *Library* is a collection of Component objects. It provides methods for creating a library from a text file and querying it for different components and attributes. Every instance of the Problem class has an associated library. Both Component and Library are generic classes that can be extended with problem-specific atrributes, interfaces and parsing routines.

The information about the architecture template is stored in the *Template* class, which is also associated to Problem. It has methods for managing attributes $\mathcal{A}^{\mathcal{T}}$ of its elements, i.e., node and edge labelings, which are either numeric constants or decision variables or some expressions with these variables. Both Library and Template are used by the *Mapping* helper class that establishes an association between the two using the mapping variables and constraints discussed in Section 2.2.1. These constraints then become a part of the basis constraints $\mathcal{R}_B$. Function call `createMapping(T,L)` of the Problem class is a part of the formulation flow of ARCHEX (shown on Figure 4.2). Internally, this function refers to the Mapping class that provides the MILP encoding of the mapping.

As the system representation within our methodology and framework is a directed graph with most of decision variables being associated either to its nodes or to its edges, the main data structures of ARCHEX are matrices. Most of them inherit from the generic *Matrix* class and are reusable across different problems. In particular, *AdjacencyMatrix* stores the decision variables $e_{ij}$ of the topology selection problem, while the entries of *QuantityMatrix* can be related to different labelings of edges of the template, which could be Boolean, integer or real variables (e.g., flows $\lambda_{ij}$, path variables $y^{\pi}_{ij}$). *IncidenceMatrix* describes possible connections between nodes of the template and is used, for example, in path constraints (see Section 2.2.2). Mapping variables $m_{ij}$ have an associated *MappingMatrix* class as well. Auxiliary variables $\eta_{ij}$ used in balance and reliability constraints (Sections 3.2.3 and 3.2.4) are stored in the *WalkIndicatorMatrix* structure. Entries of these data structures, as prescribed by the parent Matrix class, can be addressed with numeric indices, while the groups of variables, i.e., submatrices, can also be queried according to certain keys (character strings). For example, the adjacency matrix object can be queried both for the connectivitity between a concrete pair of nodes $v_i$ and $v_j$ and between nodes of certain types (e.g., generators and electrical buses) using the same interface. This provides a lot of flexibility during the implementation.

Every exploration problem has an associated object of the *ConstraintHandler* class, which takes care of processing the requirements part of the specification, i.e., parsing and classifying them, calling the encoding functions for corresponding patterns and storing the obtained MILP formulations. The user is then able to query ConstraintHandler to add all constraints related to a particular concern (e.g., interconnection, timing, reliability) to their problem formulation. The querying can also be more precise, i.e., requirement-wise. This allows the user to quickly create optimization problems that capture certain design concerns from the specification. Such manipulation of constraints is useful for quickly refining the formulation (e.g., tightening the constraints), and for debugging.

Every requirement pattern has a function that generates the corresponding MILP encoding. These functions are stored in a collection of classes named in according to the category of the requirements they encode, e.g., Balance, Reliability or Timing (on Figure 4.4, for brevity, the *Patterns* class represents this collection, while in the actual implementation it is a package). In particular, mapping constraints that belong to the basis of the exploration problem $\mathcal{R}_B$ are also encoded using patterns. Every pattern has a corresponding class method that has the same name as the pattern itself (e.g., `flow_balance`, `in_conn_implies_out_conn`, and all others presented in Section 4.2). The methods are also accessible to the user, so he/she can manually add extra constraints to the created problem (this also works for learning functions).

Along with the Mapping helper class, there are also several other classes that compose the *Helpers* package. For example, the *Constraints* class is responsible for initializing auxiliary data structures, such as walk indicator matrices, and for encoding heavy or reusable constraints to make the pattern classes more compact. The *Paths* helper class provides the encoding of network paths (variables $y^{\pi}_{ij}$, $w^{\pi}_{i}$, and path constraints from the basis $\mathcal{R}_B$) using either a full enumeration of paths or an approximation introduced in Section 3.5.1. The *Linearization* class has a set of methods for converting the nonlinear operations (e.g., products of variables, logic operations) to a linear form so they could be added to the MILP formulation. Linearization methods are called from pattern methods whenever needed and return auxiliary variables $x_A$ and constraints that are further stored in ConstraintHandler together with the formulations. The *Wireless* class provides methods for encoding a set of properties and attributes of wireless networks (e.g., RSS, BER), which are then used to define the requirements. This and other classes implemented as a wireless extension of ARCHEX are further discussed in Section 4.3.2.

The two classes, *eagerAlgorithm* and *lazyAlgorithm*, implement the two optimization techniques. Depending on the selected algorithm, one of the routines is executed via the `solve()` method call in the Problem class. The abstract *Analysis* class provides a skeleton for different analysis techniques that must be implemented by the child classes (e.g., by reliability analysis), for example, the `run()` method. The latter can be called both from the Problem class and from the lazyAlgorithm to get the results and, if they do not meet the requirements, the counterexample (e.g., a functional link that does not have enough reliability, a network route that violates the timing constraints and so on).

Remaining classes are responsible for visualizing the generated architectures and for setting up the toolbox. The graph of the final architecture can be plotted by calling one of the methods of the *Visualization* class. The architecture template, i.e., a reconfigurable graph with all possible (allowed) connections, can also be drawn. This can help the user to visually verify the correctness of the specification (number of components, composition rules) as well as to set up the constraints. Some auxiliary information can also be shown on the graph, e.g., floor plan, coverage and others. Finally, the *Settings* class handles the logging of the toolbox as well as configurations of MILP solvers, i.e., presets of solver-specific settings (e.g., optimization gap, solution strategy, timeout and so on). Different presets can be used, for example, for different algorithms (eager, lazy) as well as for different problems. For instance, the optimization gap (distance between the current value of the objective and the minimal achievable one) can be increased, so the solution is generated more quickly without exploring the part of the search tree. More details are provided in Section 4.4.3.

### 4.3.2 Extension for Wireless Networks

ArchEx also includes an extension to a significantly different domain - wireless networks. These networks are becoming a locomotive of many networked cyber-physical systems being one of the primary communication channels for exchanging sensing data and control (actuation) commands. Design and calibration of the wireless infrastructure is, therefore, one of the major tasks in CPS design. As discussed in Section 1.4, a plethora of approaches for wireless network design have emerged over recent decades. Compared to those, our aim is to provide a more general framework that is capable to capture more types of heterogeneous requirements than most of existing tools. This thesis presents a generic basis for encoding CPS exploration problems as mixed integer linear programs, which allows designers to formulate optimization problems subject to connectivity, routing, energy consumption, location, timing, and other constraints. Another distinguishing feature of the methodology is the mapping of network components to implementations taken from libraries. Most of existing approaches assume fixed configurations of devices when solving optimization (and other) problems. Overall, our formulation significantly increases the breadth of the search space compared to related work, and is able to efficiently encode large problems and find solutions in reasonable time. Moreover, in our methodology and tool, wireless network topology can be seen as a subsystem and jointly synthesized with other (physical) components of the CPS.

The wireless extension of ArchEx is summarized in Figure 4.5. The goal of the exploration problem is the joint selection of the network components and the topology (node placement and routing) subject to a set of network requirements (e.g., link quality, routing, power consumption). Along with specification and library text files, the input is extended with a floor plan SVG (Scalable Vector Graphics) file that stores the information about space dimensions, obstacles (e.g., walls, doors, windows, furniture), locations of network devices as well as other auxiliary locations (e.g., for evaluating localization systems, as discussed in Section 3.2.9). Obstacles (represented by polygons) and node locations can be saved as different layers in the same SVG file, which is convenient, because several different network templates can be read from a single file, while adding new locations is easy with an SVG editor. In turn, the specification file is augmented with the parameters of the channel model (type, related parameters), the network protocol (e.g., bit rate, duty cycle, packet size) and the battery (number, type, capacity).

There are 4 additional classes (shown in yellow on Figure 4.4) that implement the new functionality. The *AreaMap* class includes a parser for the SVG floor plan that reads all the layers of the file and stores the information about space dimensions, existing obstacles and locations in different class properties. Node locations, in particular, are provided as $\lambda_i^{\mathcal{T}}$ attributes to the Template class as node labels. They can be seen as *candidate* locations of the network

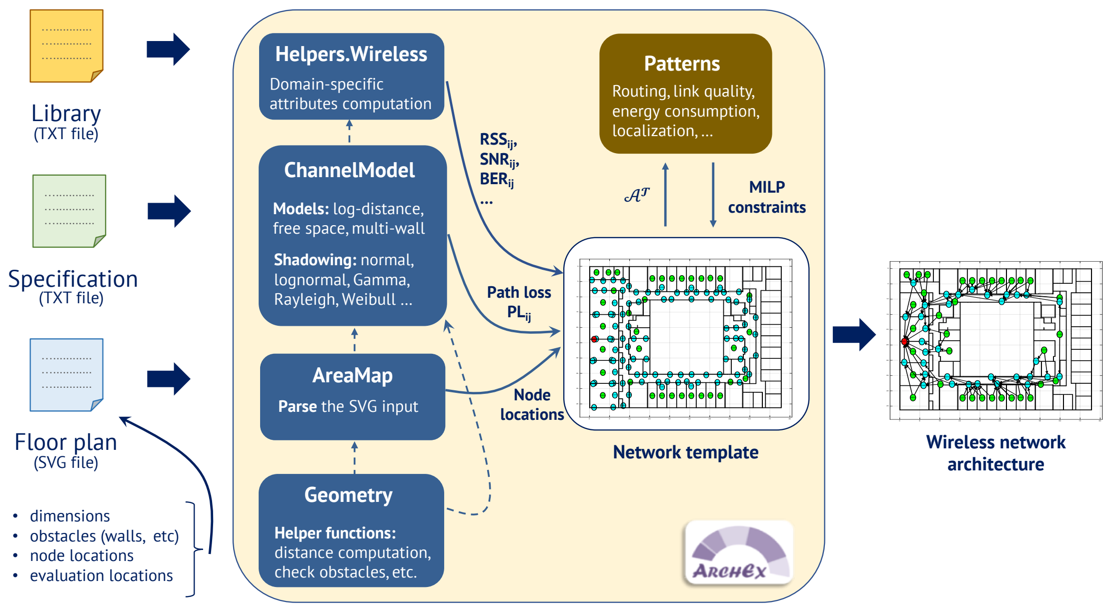

Library
(TXT file)
Specification
(TXT file)
Floor plan
(SVG file)
• dimensions
• obstacles (walls, etc)
• node locations
• evaluation locations
Helpers.Wireless
Domain-specific attributes computation
ChannelModel
Models: log-distance, free space, multi-wall
Shadowing: normal, lognormal, Gamma, Rayleigh, Weibull ...
AreaMap
Parse the SVG input
Geometry
Helper functions: distance computation, check obstacles, etc.
RSS_ij, SNR_ij, BER_ij ...
Path loss PL_ij
Node locations
Patterns
Routing, link quality, energy consumption, localization, ...
MILP constraints
Network template
Wireless network architecture
ArchEx

Figure 4.5 – Wireless extension of ArchEx.

deployment. Every layer in the SVG file has a name, so that a particular candidate deployment can be used in the specification. Also, certain constraints, such as localization, can be enforced for a particular set of evaluation locations from the SVG.

One of the important steps in defining the constraints for a wireless network topology is the calculation of the link path loss. The *ChannelModel* class provides a set of channel models that can be used for this calculation and have different levels of accuracy, complexity and flexibility. We briefly describe them later in this section. Links (edges) of the network template are labeled with the computed path loss values $PL_{ij}$. For lowering the size and the time of the computation, the latter can be skipped for the pairs of nodes that are located far away from each other according to some distance threshold, assuming that the corresponding link is not possible. The pruning can be also performed for the incidence matrix of the template by removing the "impossible" edges, which can significantly decrease the number of path variables $y^{\pi}_{ij}$ for every required path $\pi$ (see path constraint (2.2); smaller size of the matrix $C$ entails smaller number of edge variables $y^{\pi}$). On the whole, provided channel models, as well as the mechanisms for handling the area map (e.g., obstacles), support both indoor and outdoor scenarios.

Computed path loss values are also provided to the *Wireless* class from the Helpers package. This class provides a set of methods for defining the attributes related to wireless network components and links, such as the ones discussed in Section 3.2.7, e.g., received signal strength, signal-to-noise and others. These attributes are provided as labelings for nodes and edges of the template and, together with the path loss, they are used for defining the constraints, such as the ones presented in Sections 3.2.7-3.2.9. The Patterns package is extended with a set of classes containing corresponding requirements.

Finally, the *Geometry* class is used by AreaMap and ChannelModel for various geometric computations, including, but not limited to, computing the distance between the nodes and finding all the obstacles that cross the line of sight between the transmitter and the receiver.

**Channel Models**. ArchEx currently supports four different channel models. These models are included in our library in order to map virtual links of the architecture template to real (physical) ones. The *free space model* [110] assumes ideal propagation conditions with clear line-of-sight path between transmitter (TX) and receiver (RX). It is based on the Friis equation, which is defined as follows:

$$PL_{FS} = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2 L}, \tag{4.1}$$

where $PL(d)$ is the path loss at distance $d$, $P_t$ is the transmission power, $G_t$ and $G_r$ are transmitter and receiver antenna gains, $L$ is the system loss factor, and $\lambda$ is the wavelength.

An improvement to the free space model for long distances is called *two-ray ground reflection model* [110]. It also considers the signal reflected by the ground and is defined by the following formula:

$$PL_{RM} = \frac{P_t G_t G_r h_t^2 h_r^2}{d^4 L} \tag{4.2}$$

where $h_t$ and $h_r$ are TX and RX antenna heights, respectively, while the other terms correspond to the ones from the formula (4.1). In our formulation, TX power $P_t$ and antenna gains $G_t$ and $G_r$ are the attributes of the network template, therefore, they are expressed using the corresponding values from the platform library and the decision variables (mapping). Hence, the product $P_t G_t G_r$ is nonlinear, but linearization techniques can be applied.

One of the most used channel models, which does not require any nonlinear operations in our formulation, is the *log-distance path loss model* [110]. It is actively used, in particular, in wireless sensor network design tools (e.g., PASES [84], Castalia [19], MiXiM [65] or WSNet [25]). The model accounts both for propagation path loss, logarithmically decreasing, and for shadow fading effects using a probabilistic model:

$$PL_{LD} = PL(d_0) + 10\eta \log(d/d_0) + X_\sigma \tag{4.3}$$

where $d$ is the distance between TX and RX antennas, $PL(d_0)$ is the path loss at a reference distance $d_0$, which is typically one meter for indoor WSNs [42], $\eta$ is the path loss exponent (PLE), which is determined either empirically or from the literature, and $X_\sigma$ is a zero-mean Gaussian random variable with standard deviation $\sigma$. The Gaussian random variable accounts for shadow fading by adding some random error to the propagation path loss. The mean value of the underlying distribution is typically zero.

Finally, ArchEx also uses the multi-wall (MW) model [42], which is an extension of the log-distance model that also accounts for the attenuation in walls and other obstacles. Following expression is used:

$$PL_{MW} = PL_{LD} + \sum_{i \in Obst} \Big( c_i W_i^0 + l_i W_i \Big), \tag{4.4}$$

where $Obst$ is the set of indices of obstacles (in the list of all obstacles in the area) that cross the line of sight of the currently evaluated link, $c_i$ is the number of times that the signal crosses the obstacle facets, $l_i$ is the distance the signal travels inside the obstacle, $W_i^0$ and $W_i$ are the attenuations of the signal, respectively, per every cut of the obstacle's facets and per every meter that the signal propagates inside it. Attenuation values depend on the material of the obstacle. The values of $c_i$ and $l_i$ in our toolbox are computed using the methods of the Geometry class. The MW model, therefore, uses the path loss computed by the log-distance formula (4.3), which accounts for distance attenuation and for random shadowing effects, and augments it with the attenuation in walls and other obstacles along the signal path.

We further note that, despite the complexity of some of the aforementioned models, they still may give predictions with a large absolute error (e.g., 5-10 dB). Errors in the models may arise from various factors that are not captured, e.g., interference, multipath propagation (typical for indoor scenarios) or wave-guiding effects. These and other random factors are modeled using a probability distribution, such as the normal distribution, which may not be accurate in certain situations. Furthermore, ARCHEX deals with the architecture design stage and has to assume static steady-state values for system and environment parameters including the path loss. The situation is different for network simulators, which can more carefully investigate the link quality, because the channel model is typically used every time the new signal has to be transmitted during simulation. In contrast, path loss values in ARCHEX are computed only once during initialization and encoding of the exploration problem and are, therefore, constant. On such early design stage, one can consider, for example, worst case values of the path loss to make the system architecture provide certain guarantees, i.e., run several trials of the channel model for a given link and select the worst result according to a given criteria.

We use several approaches for improving the quality and the accuracy of the path loss computation. First, the ChannelModel class can return the worst, best or an average value out from an adjustable number of trials. Second, similarly to [108], we are able to integrate the site survey data, i.e., label the links with measured path loss values instead of the ones computed by the channel model. This can significantly increase the accuracy, but requires a measurement campaign in the deployment area, which may be hard to conduct especially for a large scale network. Third, we provide a set of different probability distributions to model the shadowing effects (e.g., Normal, Lognormal, Gamma, Rayleigh, Weibull, Nakagami-m), so it is possible to choose a more suitable one according to a given environment. Finally, we have performed a large set of measurements in indoor office spaces for a 2.4 GHz signal of wireless sensor networks. We have then statistically characterized the data and proposed a set of additional improvements for conventional channel models [64]. We provide the measurement and statistical results and discuss the proposed improvements in Section 6.3 of this thesis.

## 4.4 Using ARCHEX: Design Flow

In this section we outline the main steps for using the ARCHEX framework: creating the specification and the library, solving the exploration problem, tuning the solver parameters, visualizing and saving the solution. Additional details and demo problems are available in the public repository [1].

We employ an example from the avionics domain - Fuel Management System (FMS) of an aircraft [11]. A simplified illustration is given on Figure 4.6. An FMS performs pumping, managing, and delivering aviation- or jet fuel to the propulsion system and Auxiliary Power Unit (APU) of an aircraft. All these operations are automatically performed by a control algorithm and a set of sensors and actuators, which constitute the "cyber" part of the system. Fuel is piped from a set of tanks to control valves (also known as selector valves), which allow to choose, which tanks currently feed the engines. In particular, the valve can be shut off in case of emergency, e.g., fire in the engine. Each valve is connected by a set of pipes and low-pressure pumps to a fuel filter (or stainer) for water and small particles of sediment. After passing the filters, fuel is sent under pressure to the inlet side of the fuel injection metering unit. The metering unit for each engine provides the proper flow of fuel to the distribution manifold which feeds the injectors. The latter provide the fuel directly to the engines.

We note that, for the sake of demonstrating the functionality of the toolbox, we do not provide here the whole specification, library or generated architectures of realistic size and level of detail. Instead, we use a toy example and a very simplified system representation. We assume following component types: fuel tank (T), low pressure pump (P), filter (F), injector (I) and engine (E). The architecture template $\mathcal{T}$ consists of these five types, while pipes, selector valves and injection pumps are modeled with edges. Other components of the FMS are neglected in this example. We further state that pumps P can be connected together via additional pipes, which allows the fuel from one tank to be forwarded to another part of the system. Such connections are represented by "horizontal" edges of the architecture graph (between components of the same type).

The goal of the exploration is to minimize the dollar cost and weight of system components, i.e., the objective function is a weighted combination of the two concerns. Constraints on the architecture include number of components, interconnection and flow balance. Once an architecture that satisfies these constraints is generated, the next step is the synthesis of the control algorithm, which is out of scope of current work. However, the future work on ARCHEX involves the development of control-related patterns and integrartion with tools that perform controller synthesis for a given system architecture.

### 4.4.1 Creating the Specification

Having described the conceptual organization of the system, as a first step of using ARCHEX the user has to create the input files (library and specification). These are text files that follow a particular syntax described below.

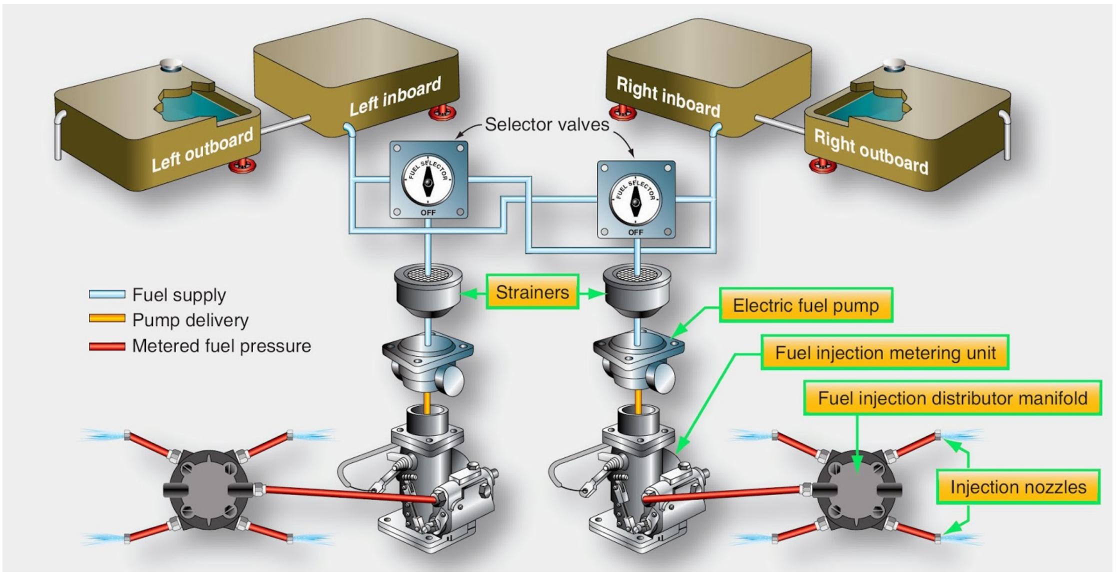


Figure 4.6 – A simplified FMS of a high-wing, twin-engine aircraft [11].

**Library**. The library input file consists of two parts: description and list of components. The description part has two parameters: name of the library and list of comma-separated component types, which are present in this library (based on them the Library class knows which components to expect while parsing the file):

```
Name: FMS library;
Types: Engine,Filter,Injector,Pump,Tank;
```

This description section provides a library name ("FMS library") and five component type names. Components list has the following syntax:

```
Type_name:Name, Subtype, Cost, attr_1, attr_2, ..., attr_n;
```

Every component entry (row) has type name and a list of attributes of this component after a colon. Attribute values are separated by commas. The expected ordering of attributes is defined within a dedicated library class (e.g., FMSLibrary.m). Name, Subtype and Cost are mandatory attributes for every component. If they are not used they still must be present as first three attributes of a component, but their values can be omitted.

Other attributes are problem-specific (e.g., weight, fuel capacity for FMS). Their ordering in the input file is strict and predefined by a corresponding library class according to the implemented parsing routine. ARCHEX provides dummy library files for different application domains, which include comments on the ordering of component attributes that can be easily followed. A part of the library for our example FMS is shown in the snippet below:

```
%% COMPONENTS
%Type:Name,Cost($),Subtype,Weight(kg),Capacity(l),Throughput(l/sec);
Tank:FTank1,5000,no_intl_pump,500,460,3;
Tank:FTank2,5000,intl_pump,300,255,2.5;
Tank:FTank3,5000,intl_pump,250,239,3.2;
Pump:Pump1,1500,out_tank,23,,4.2;
Pump:Pump2,2100,in_tank,28,,2.8;
Pump:Pump3,3500,in_tank,20,,3.5;
Pump:Pump4,2600,emergency,15,,3;
Filter:Filter1,1000,active,8,,2.6;
Filter:Filter2,750,passive,5,,2.1;
```

The listing above illustrates a small collection of FMS components of 3 types: fuel tanks, pumps and filters. The mandatory attributes (name, cost and subtype) are followed by several problem-specific ones, e.g., component weight, capacity and throughput. Some attributes (e.g., capacities for pumps and filters) are omitted, because these components do not have such characteristics. Subtypes are used to signify component's additional features. For instance, `Tank2` and `Tank3` can have an electrical pump installed internally, which is marked by the `intl_pump` subtype, while this is not the case for `Tank1`. Consequently, some of the pumps from the library can be mounted inside the tanks and some cannot. Similarly, one can fill in the whole, possibly much larger, library, with realistic components of fuel systems.

**Specification**. As previously explained in Section 4.1 and Figure 4.2, the specification file consists of several parts. The first one is the problem description with several important inputs, namely, component types used in the problem (similar to the library), functional flow, name of the problem, tags (in our specification they are not used), and structure of the objective function. For our example, the problem description looks like following:

```
BEGIN (Description)
Name: FMS sample problem;
Components: Engine,Filter,Injector,Pump,Tank;
Functional flow: Tank,Pump,Filter,Injector,Engine;
Objective: $(0.5) + Weight(0.5);
END (Description)
```

Objective function is an equally weighted combination of dollar cost and total weight of components (ARCHEX generic parser of problem descriptions recognizes "`$`" as the dollar cost, while `Weight` is a problem-specific attribute from the library; weight coefficients for different members of the objective function are specified in brackets).

The description part is followed by the template structure and composition rules sections, which define, respectively, the maximum number of components of each type in the template, and the list of allowed connections between these types. The sections currently have the following syntax:

```
BEGIN (Template)
Tank 2;
Pump 4;
Filter 3;
Injector 3;
Engine 3;
END (TEMPLATE)

BEGIN (Composition rules)
Tank => Pump;
Pump => Pump;
Pump => Filter;
Filter => Injector;
Injector => Engine;
END (Composition rules)
```

The specified composition rules correspond to the functional flow, i.e., components of preceding type can only be connected to components of succeeding type (with an exception of pumps, as mentioned above).

Requirement patterns fill in the last part of the specification. In particular, component types and subtypes are actively used as arguments for the patterns. While the whole list of constraints can be larger, below we demonstrate a smaller illustrative part:

```
BEGIN (Constraints)
exactly_N_components(Tank,2);
exactly_N_components(Engine,3);
at_least_N_components(Pump,emergency,1);

at_least_N_connections(Tank,Pump,1);
at_most_N_connections(Pump,Tank,1);
exactly_N_connections(Engine,Injector,1);

in_conn_implies_out_conn(Pump,Tank,Pump|Filter);
in_conn_implies_out_conn(Filter,Pump,Injector);
in_conn_implies_out_conn(Injector,Filter,Engine);

cannot_connect(Tank,no_intl_pump,Pump,in_tank);
cannot_connect(Tank,intl_pump,Pump,out_tank);
cannot_connect(Pump,in_tank,Filter,active);
no_self_loops(Pump);

has_sufficient_input(Tank,Engine);
...
END (Constraints)
```

The above list includes patterns from several categories specified in Section 4.2, namely

Number of Components, Interconnection and Balance. The first group (3 patterns) presribes that all engines and tanks of the architecture template must be used. Moreover, one of the electrical fuel pumps installed between tanks and filters, must be an emergency pump (as marked by a corresponding subtype in the library), which is activated when one of the primary pumps fails. Following groups of constraints ensure the valid system interconnection (e.g., all engines must receive fuel from injectors, but no more than one for each engine; filter connected to a pump must be connected to an injector). The last pattern is a balance constraint, which ensures the sufficient fuel storage in the system.

While the provided specification is incomplete for the sake of demonstration, the rest of the requirements can be written down by the designer in the same intuitive way. On the whole, the number of rows (100-200) in the final specification file is very small compared to the input file generated for the CPLEX solver, which amounts to several thousands of rows, including constraints and variable declarations, even for such a small design example.

### 4.4.2 Solving the Exploration Problem

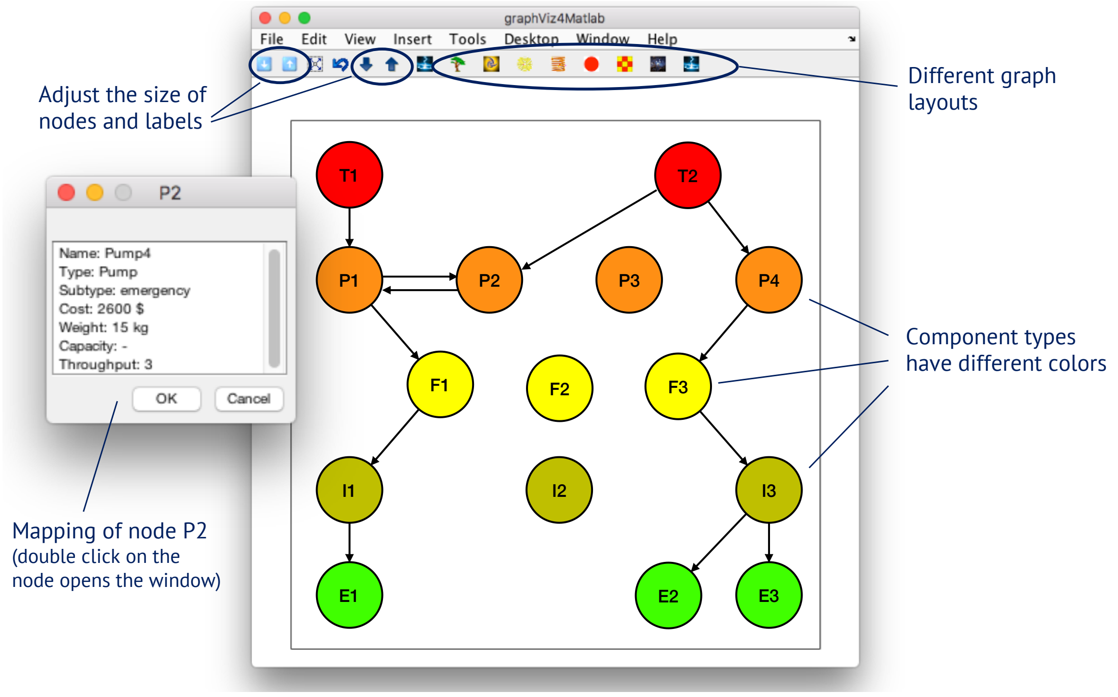
Having created the input files (library, specification and, possibly, the SVG floor plan for a wireless network problem), user can provide their filenames to ARCHEX as arguments:

```
ArchEx(problem_type, library_file, problem_file)
```

where the first argument is the problem type (string), so that the tool can create the instance of the specific problem (e.g., FMS). This script automatically includes all necessary folders to the MATLAB path, creates an instance of a corresponding problem and solves it, i.e., makes all steps of both the formulation and the solving execution flows shown on Figure 4.2. During the execution, ARCHEX does some console prints, which allows the user to see what it is currently doing. When the optimization is finished, the generated architecture graph is shown and solution is saved to disk. Some results are also printed to the MATLAB console, for example, formulation time, solving time, results of the analysis. If the formulation is not feasible, i.e., there are contradictions in the constraints or no satisfying architecture can be found, then `Unfeasible` will appear in the console and no graph will be shown.

An example of the visualization is shown on Figure 4.7: nodes of different types have different colors (more advanced coloring schemes highlighting different subtypes are also available). It is also possible to hide the nodes that are not instantiated, e.g., P3, F2 and I2 on Figure 4.7. User can check out the mapping of the components by double clicking on the corresponding nodes of the graph: information about a library element that implements the "virtual" component will be shown in a small window. Default graph layout sorts the nodes by component type, while other layouts are also available for specific problems, for example, for wireless networks.

**Debug Mode**. If the control script is called without arguments, then the tool is launched in debug mode, i.e., the problem is not automatically solved, but only formulated, so the user


graphViz4Matlab
File Edit View Insert Tools Desktop Window Help
P2
OK
Cancel
Adjust the size of nodes and labels
Different graph layouts
Component types have different colors
Mapping of node P2 (double click on the node opens the window)
T1
T2
P1
P2
P3
P4
F1
F2
F3
I1
I2
I3
E1
E2
E3

Name: Pump4
Type: Pump
Subtype: emergency
Cost: 2600 $
Weight: 15 kg
Capacity: -
Throughput: 3

Figure 4.7 – Visualization of the solution (architecture graph) in ARCHEX.

can check it before running the solver. The created Problem object (called `prob`) is stored in the MATLAB workspace, so it is possible to explore its contents (public properties) and call its methods to manually control the toolbox, e.g., redefine the constraints or the objective function, select the algorithm for solving the problem, adjust the settings of the solver, call the solver, run the analysis of the architecture, visualize or save the results.

One of the debugging methods available in ARCHEX is pushing constraints (patterns) one by one to the formulation. That is, if the problem is infeasible, the user can run ARCHEX in the debug mode and start with a simpler formulation, which has a feasible solution, and then incrementally add the constraints to find out which one brings the infeasibility. In such a way, errors in the specification, that cause the optimization to fail, can be quickly identified. Following method is provided by the Problem class:

```
prob.addConstraint(pattern);
```

The method automatically parses the pattern declaration, provided as argument, and creates the corresponding MILP constraints and the required infrastructure (e.g., auxiliary variables). Same method is used by learning functions in the "lazy" optimization approach to incrementally augment the specification with new proposed constraints. ARCHEX currently does not provide the `Pop()` interface, which would allow for taking out recently added constraints and backtracking, because, unlike SAT problems, adding new constraints typically affects the whole formulation, i.e., many existing references are redefined. Investigation of backtracking capabilities requires additional studies in future works.

User can also query the whole categories of constraints (e.g., balance or timing) to the formulation. As discussed in Section 4.3.1, this is possible by calling the ConstraintHandler class:

```
prob.ConstraintHandler.getConstraintsMILP(Timing);
```

As a result of the command above, the whole set of timing constraints (if any) from the specification will be added to the formulation. In such a way, it's easy to manipulate the viewpoints used in the current exploration problem. If the newly added set of constraints is independent from the current formulation (no dependencies or variable references have to be updated), then it is simply added to the list of problem constraints. Otherwise, existing formulation is fully redefined, i.e., existing constraints are also regenerated to obtain a valid monolithic formulation. This can also be done manually:

```
prob.defineConstraints();
```

We note that the template, library, mapping and existing basis constraints are not affected by this command. Therefore, redefining constraints can be much less time consuming compared to fully recreating the problem instance.

Users can also quickly switch between different objectives:

```
prob.setObjective(weight);
```

which allows users to run optimizations for different objectives within the scope of the same formulation.

The solving technique can be selected simply as

```
prob.setAlgorithm();
```

with "eager" or "lazy" as argument. Depending on a particular problem, iterative approach may or may not be implemented, while monolithic optimization is always the default one. For the problems that include path variables $y^\pi$ and $w^\pi$ in the basis, one can also choose between full path enumeration and approximate path encoding (we omit the code examples).

The solver can be invoked by the following command:

```
prob.solve(options);
```

where the `options` parameter specifies the settings for the solver, which are further explained in Section 4.4.3. In turn, analysis used in the given problem can be started by typing

```
prob.Analysis.run();
```

When the analysis is finished, results are printed to the console and stored in the workspace.

### 4.4.3 Additional Functionalities

In this section we briefly outline the additional important functionalities of ARCHEX, such as visualization, saving and configuring the solver.

User can draw the graph of the current solution with the following method:

```
graph = prob.showResult();
```

The graph will be drawn in a separate window as shown on Figure 4.7, while the graph object will be stored in the corresponding field of the MATLAB workspace. Finally, user can save this graph object as well as inputs and outputs of the exploration problem with

```
prob.Save();
```

This method of the Problem class saves the architecture graph as a .png image, while the specification data (types, flows, number of components, etc) and solution (assignments to decision variables) are saved to a .mat file. Furthermore, the input file with MILP formulation for the solver (currently, CPLEX) is saved to a separate text file.

ARCHEX also provides few additional functions for visualization:

```
graph = prob.showTemplate();

prob.showCoverage(nodes);

prob.showObstacles();

prob.showPoints(point_set);
```

The first method visualizes the architecture template, i.e., the input of the exploration problem. The graph shows all possible connections between nodes (an example can be seen on Figure 2.1), so that the user can verify if the composition rules have been correctly specified. Furthermore, he/she can get more understanding on the system structure, which may help in writing down the requirements. Next three methods are currently implemented for the wireless extension of ARCHEX. First one draws the coverage map of the network deployment, which is either estimated (if it is drawn for the template) or actual (if the problem is already solved and the topology has been generated). `showObstacles()` is used to visualize the floor plan on top of the topology graph, while the last method is used to draw the evaluation points used in localization constraints.

**Solver Options**. Certain parameters of the MILP solver can be configured by the user, for instance, optimality gap, timeout, verbose level, etc. Configuring them is useful for certain experiments and designs. For example, for debugging one might want to set the amount of displayed information to maximum. For testing it may be necessary to set a timeout to make the solver terminate at a certain moment, e.g., to see if it is able to find any feasible solution within a predefined time.

Furthermore, it is sometimes useful to adjust the *optimization gap*, i.e., the distance between the current value of the objective function and the best achievable one, as estimated by the branch and cut algorithm. This allows the solver to terminate once a near-optimal solution is found. MILP problems are NP-hard, so it is unknown, how long it takes to find the final solution, i.e., to explore the whole design space. However, a typical case happens when a very good solution (e.g., with a 10% gap from the optimum) is rapidly found, and then the structure of the problem does not allow the solver to prove the optimality for a long time (e.g., hours). That is, the rate of change of the optimization gap significantly decreases, and the solver progresses towards the optimal solution very slowly. In such situations, the optimization can be terminated with a near-optimal solution, which is still much better than the first found feasible configuration.

In the current version of ArchEx, interfacing with different MILP solvers is done by the Yalmip [76] toolbox. In particular, the `solve(options)` method of the Problem class, mentioned in previous section, calls one of the solving algorithms (eager or lazy). In turn, these algorithms internally call the `optimize(cons,cost,options)` function of Yalmip. Here, `options` is a dedicated structure of type `sdpsettings` provided by Yalmip, which stores the configurations of different solvers. It allows the user to flexibly tune the desired parameters. Following code snippet illustrates an example of useful settings of ‘the CPLEX solver:

```
% create solver options structure via YALMIP command
options = sdpsettings;

% select IBM CPLEX as the solver
options.solver = 'cplex';

options.cplex.threads = 1;
options.cplex.mip.tolerances.mipgap = 0.005;
options.cplex.timelimit = 43200; % 12h

options.savesolverinput  = 1;
options.savesolveroutput = 1;

% maximum verbose
options.debug        = 1;
options.verbose      = 2;
options.showprogress = 1;
options.saveduals = 0;
```

First, the `options` structure is initialized and CPLEX is chosen as a solver. CPLEX is then configured not to use parallelization by setting the maximum number of allowed threads to one. This is sometimes useful, because parallel exploration of different subtrees by the branch and cut algorithm may come with a very high time cost of synchronizing the threads. In our experience and in the scope ARCHEX, single-core optimizations typically completed much faster. Thread limitation is followed by specifying the optimization gap and the timeout, i.e., the solver will terminate once reaching the 0.5% gap or by the 12h timeout. Next group of parameters force CPLEX to save both its input and output and store them in the workspace of MATLAB. The last group deals with the verbose settings of CPLEX, i.e., how detailed is the information about the optimization progress printed to the console.

Once the `options` object is configured, it is passed to the solver. The `solve()` method can be also called without arguments for the default set of settings to be applied. ARCHEX stores some predefined setups for CPLEX in the Settings class (see the diagram on Figure 4.4). For quickly creating the options, the user can, for example, type

```
options = Settings.cplex_options('eager');
```

which will automatically create the `sdpsettings` object and initialize it with default solver parameters for the monolithic optimization. It is possible to add customized configurations for different solvers to the Settings class.

More details and instructions are available in the public repository [1], while we refer the reader to the manuals of particular solvers for more information on their calibration and usage. In the following two chapters we demonstrate the efficiency and the scalability of ARCHEX and the proposed exploration methodology by running numerical evaluations on several industrial case studies.

# 5 Reliability-Driven Design of Industrial Cyber-Physical Systems

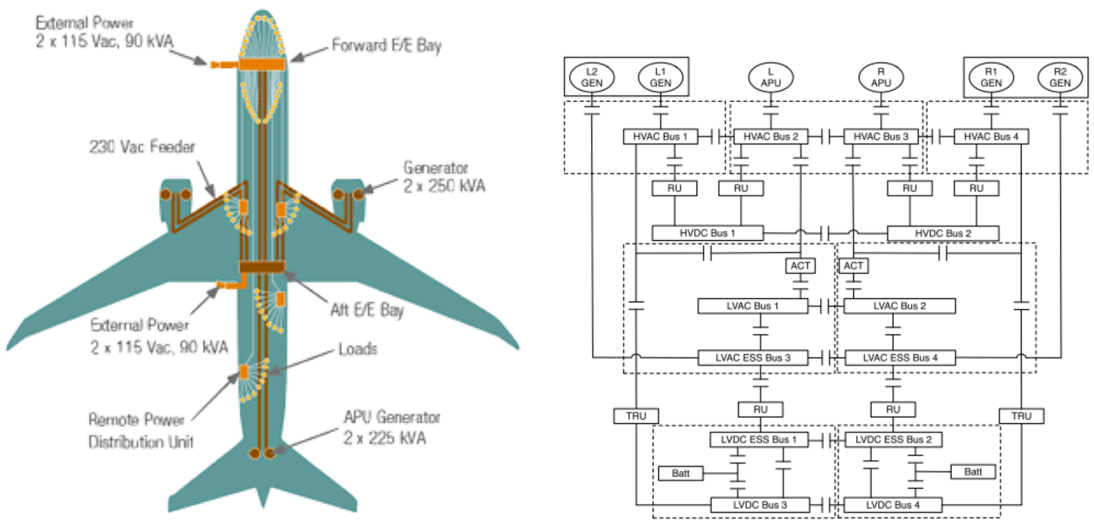
*This chapter deals with CPS applications from two different industrial domains, both of them having reliability as a major design concern. We use them to evaluate the efficiency and the scalability of the proposed architecture exploration methodology and the developed framework (*ArchEx*). The first application is the electrical power distribution network of a passenger aircraft, in which a set of electrical loads have to stay powered from a set of generators even if some of the components fail. That is, along with interconnection, balance and timing constraints, the functional links of the system are subject to tight reliability requirements. The second application, a reconfigurable manufacturing system, also has a bound on the failure probability of its production lines, while timing and workload constraints have to be satisfied, and different operation modes need to be supported.* ArchEx *is able to capture the heterogeneous requirements of these systems in short pattern-based specifications, which demonstrates both the expressiveness of the methodology and the usability of the framework. We further evaluate the two algorithms for solving exploration problems, monolithic and iterative, discussed earlier in this thesis. Despite having different complexity of the formulation, execution time and optimality guarantees, both of them are able to synthesize reliable cost-effective CPS architectures in reasonable time, proving the scalability of our approach.*

## 5.1 Aircraft Electrical Power Distribution Network

### 5.1.1 Overview and Problem Statement

We first apply our approach to the avionics case study from [12, 99]. The number of electronic components installed on modern aircrafts has increased over the past years, which makes the design of safety-critical subsystems challenging. Along with the fuel management system, introduced in Section 4.4, an aircraft Electrical Power distribution Network (EPN), such as the one in Figure 5.1a, is a notable example. As an aircraft becomes increasingly more electric (i.e., hydraulic and pneumatic systems are replaced with electric systems), the power system becomes increasingly critical for safe operation.

A sample EPN architecture is shown on Figure 5.1b in the form of a single-line diagram, which was previously introduced in [99], together with the overall system description, and



(a) (b)

Figure 5.1 – (a) Power distribution of Boeing 787 (source: www.boeing.com); (b) Simplified diagram of an EPN adapted from a Honeywell patent [99].

a qualitative discussion of the main design requirements. EPNs typically consist of power generation, primary distribution and secondary distribution subsystems. To demonstrate our methodology, we focus on the primary power distribution subsystem. Here, power is delivered from a set of sources (engines and batteries) to a set of sinks (electrical loads) via AC and DC buses. The system is divided into left and right parts, but the corresponding generators (L/R-GEN) and auxiliary power units (APUs) can power both parts. Components can be further classified as high voltage (HV) and low voltage (LV). Rectifier units (RU) are used to convert AC power to DC power, while HV levels can be converted into LV levels using a transformer-rectifier unit (TRU). Several buses are *essential* (ESS) meaning that they must be present in the architecture. Sensors monitor the health state of generators and buses and inform the controller, which actuates a set of switches (contactors) to keep critical loads powered even if the components fail.

Primary electric loads (not shown on Figure 5.1b) include communication and computing systems, electrically-driven actuation systems (including electro-hydraulic systems), anti-ice and/or de-ice systems, and lighting systems. DC loads can also be powered by batteries (Batt) in case of emergency or while on the ground, when the engines are shut down. We further note that some of the loads, e.g., avionics components, hydraulics, fuel boost pump and window heating, are *non-sheddable* (essential, critical), i.e., they must remain powered in any circumstances. In contrast, other loads, such as ice protection unit, heating and lighting, are *sheddable*, i.e., they can be dropped if power supplies are insufficient.

**Requirements**. Our methodology deals with the exploration of the architectural space. Therefore, we are interested in the part of EPN requirements that are related to the system architecture, i.e., to components and their interconnections, and neglect the ones related to the controller. In particular, *safety* requirements constrain the way each bus must be powered to avoid the loss of essential features. In this context, for example, AC power sources must not be parallelized, i.e., simultaneously connected to the same AC bus. Similarly, every TRU can serve a single bus. Furthermore, the power in nominal conditions must be balanced, i.e., total load must be within the capacity of the generators, while each bus must be capable of powering all the loads which are connected to it. We refer the reader to [99] for a more detailed list of safety requirements. Overall, in ArchEx all of them can be captured by interconnection and balance constraints as we demonstrate in the next section.

EPN *reliability* requirements prescribe that the system must be designed to be safe up to certain amount of operation time. Each component of the system has a failure probability, therefore, the system must be capable of tolerating any combination of component faults that has a joint probability more than a given threshold. We capture such requirements with reliability constraints discussed in Section 3.2.4. The latter use an encoding based on the approximate reliability algebra proposed by P. Nuzzo in [96], that replaces the full enumeration of system failure events with a set of linear constraints, which capture the system reliability with a small bounded error. Alternatively, the iterative optimization technique proposed in [12] and generalized in Algorithm 2 can be applied, as shown in Section 5.1.3.

*Performance* requirements impose certain priorities on the connections of the system, for example, requiring particular buses to be powered only from left/right generators or APUs. Furthermore, some components ought to provide certain guarantees, such as valid power levels for generators, bounds on opening/closing time of individual contactors and so on. Such requirements in our methodology are modeled by interconnection constraints in combination with valid mapping decisions.

Some safety and performance requirements are imposed on the controller synthesis problem, which is currently out of scope of our methodology. It is important to note, however, that, as shown in [99], design requirements can be partitioned in a way that an EPN can be designed in a *compositional* fashion, i.e., architecture exploration and controller synthesis methodologies can be deployed *independently*. In the framework of design contracts the authors prove that if the architecture and the controller satisfy their contracts, the controlled system will also be correct and will satisfy the system-level requirements [99].

**Exploration Problem Statement**. The goal of the exploration problem is to select the topology and the components of an EPN that satisfy a set of interconnection, balance (power flow), safety and reliability requirements, while minimizing the total component cost and the number of contactors. During optimization, a tradeoff between redundancy and system cost is explored, while the synthesized architecture serves as a specification (assumption) for the subsequent controller design step.

### 5.1.2 Specification

We apply ArchEx to solve the exploration problem for EPN. As a first step, we create a library $\mathcal{L}$ with components of the following types: generators (G), contactors (C), AC buses (A), rectifiers (R), DC buses (D), and loads (L). Each component is labeled with cost $c$, subtype $s$, and failure probability $p$. Common subtypes are $HV$ and $LV$, while generators and rectifiers have extra subtypes, $APU$ and $TRU$, respectively. Generators, buses, and loads are labeled with power ratings $g$, power capacities $b$, and power requirements $l$, respectively. Contactors are modeled with edges. We further assume that contactors and loads have no failures, the other components fail with probability $2 \times 10^{-4}$, and contactors have a fixed cost.

Next, we create a problem description file, starting with the definition of the template $\mathcal{T}$. As discussed in Chapter 2, $\mathcal{T}$ is a directed graph, where each node represents a component and each edge represents an interconnection (contactor). An edge is directed from node $v_i$ to node $v_j$ if $v_j$ receives power by (or through) $v_i$ when traversing the graph from a load to a generator. The following components are parts of the template: generators (LG/RG/MG), AC buses (LB/RB), rectifiers/TRUs (LR/RR), DC buses (LD/RD), loads (LL/RL). Here, prefixes L, R and M are *tags* that group components based on their location, i.e. left, right and middle. The middle part is used only for generators: MG components represent APUs that can be connected to both left and right AC buses. The template is characterized by the maximum number of components to be used on each side, while their exact amount is decided during optimization, except for the loads, which are fixed.

As the power is delivered from generators to loads via a network of buses, we set the functional flow to $\mathcal{F}$ = (G,B,R,D,L), i.e., every source-sink path within a functional link must have at least one component of each type $T \in \mathcal{F}$ and the order has to be preserved. Composition rules define legal interconnections between components according to $\mathcal{F}$ and to their location. For example, LG can be connected to LB, but not to LR and not to RB. Buses can be connected together (e.g., LD to LD, and also LD to RD, thus allowing us to connect left and right parts). The *objective function* of the EPN problem is the sum of the costs of all components (associated with nodes) and contactors (associated with edges) used in the electrical power network architecture, which can be encoded as in (2.7).

The summary of both $\mathcal{T}$ and $\mathcal{L}$ is given in Table 5.1. With a few simplifications (e.g., there are no components that represent batteries), $\mathcal{T}$ reflects the structure and the scale of a realistic EPN topology from Figure 5.1b based on a Honeywell patent [99], which is close to the architecture of a power system of a Boeing Dreamliner. The total number of contactors, $10^3$, is an approximate number based on the total possible amount of edges in the architecture graph, which is $n \times n$, $n$ being the total number of components in $\mathcal{T}$ ($n = 48$), and on the assumption that more than half of these edges are restricted by the composition rules.

We then specify the requirements using patterns. In particular, for *interconnection properties* we use the `at_most_N_connections(`$\mathtt{T_1}$`,`$\mathtt{T_2}$`,N)` pattern to limit the number of allowed direct connections from every component of type $\mathtt{T_1}$ to components of type $\mathtt{T_2}$ to `N`. Using two instances of this pattern, we can prescribe that any rectifier that is used cannot be directly connected to more than one AC bus and more than one DC bus by using (R,B,1) and (R,D,1), respectively, as arguments. Internally, the pattern creates the MILP constraints of the form (3.2a), which can be written as follows:

$$\sum_{i=1}^{n_{acb}} M_{i,j}^{br} \leq 1, \quad \sum_{i=1}^{n_{dcb}} M_{j,i}^{rd} \leq 1, \quad \forall\, j \in \mathbb{N}:\ 1 \leq j \leq n_{rec},$$

where $M^{br}$ and $M^{rd}$ are connectivity submatrices (subsets of the adjacency matrix of $\mathcal{T}$) that correspond to edges from, respectively, AC to rectifiers, and from rectifiers to DC buses, while $n_{acb}$, $n_{rec}$ and $n_{dcb}$ are the amounts of AC bus, rectifier and DC bus components in $\mathcal{T}$.

Furthermore, we can enforce "if-then" relationships for graph edges based on expressions (3.3a)-(3.3b) using patterns, such as `out_conn_implies_in_conn(T,T`$_{\texttt{out}}$`,T`$_{\texttt{in}}$`)`: if a component of type `T` has an outgoing connection to one of components of type `T`$_{\texttt{out}}$ then it must have an incoming connection from `T`$_{\texttt{in}}$. For example, we can require that all TRUs that are connected to a DC bus must be connected to at least one AC bus, i.e., $\forall\, j \in \mathbb{N}:\ 1 \leq j \leq n_{rec}$,

$$\bigvee_{i=1}^{n_{acb}} M_{i,j}^{br} \geq \bigvee_{i=1}^{n_{dcb}} M_{j,i}^{rd}.$$

Similarly we can enforce that all DC buses that are connected to a load or another DC bus must be connected to at least one rectifier to receive power from an AC bus and that all AC buses that are connected to a TRU or another AC bus must be connected to one generator.

The `bidirectional_connection(B,B)` pattern prescribes that contactors between two AC buses provide the power flow in both directions (the actual direction depends on the current system dynamic state) and, therefore, such "horizontal" connections must be represented by two oppositely directed edges. Similar constraint applies to DC buses. Finally, the pattern `cannot_connect(G,HV,B,LV)` is used to restrict direct connections between HV generators and LVAC buses (in our system the only legal connection between HV and LV is HVAC bus to LVDC bus via a TRU). Instances of aforementioned patterns are also applied to other component types (the full specification is omitted for brevity).

Furthermore, we use the `at_least_N_components` pattern to fix certain components in the architecture. For example, all 16 loads specified in $\mathcal{T}$ must be instantiated. Also, on the power source side, there must be HV and LV generators, i.e., aircraft engines, on both left and right parts, and there must be at least one APU. Moreover, each part must have at least one TRU.

With *balance constraints* we prescribe that the total power provided by the generators in each operating condition is greater than or equal to the total power required by the connected loads. For instance, in normal operating conditions, the power generated on each side should be greater than or equal to the total power required by the loads on that side. On the other hand, when only the APU is active, then it should be capable of powering at least the critical loads on both sides of the system. We have implemented and applied the pattern

Table 5.1 – Summary of template $\mathcal{T}$ and library $\mathcal{L}$ for the aircraft electrical power distribution network example. Notation $\{a \ldots b\}$ indicates that all values are within the interval between $a$ and $b$.

| **Type** | **Number in $\mathcal{T}$** | **Cost,** | **Fail prob.** | **g,b,l (kW)** | |
|---|---|---|---|---|---|
| | (Left,Right) | $\times 10^2$ | | HV | LV |
| Generator | 2,2 + 2 APU | g | $2 \times 10^{-4}$ | 60,70,80,150 | 15,20,30 |
| AC bus | 4,4 | 20,30 | $2 \times 10^{-4}$ | 100,150 | 30 |
| RU / TRU | 5,5 | 20,30 | $2 \times 10^{-4}$ | - | - |
| DC bus | 4,4 | 20 | $2 \times 10^{-4}$ | 30 | 5 |
| Load | 8,8 | 0 | 0 | $\{7 \ldots 20\}$ | $\{1 \ldots 5\}$ |
| Contactor | $\sim 10^3$ (total) | 10 | 0 | - | - |

`has_sufficient_power(T`$_1$`,T`$_2$`)`, which is a domain-specific version of the generic balance pattern from Table 4.1, to enforce aforementioned constraints by specifying power sources (G) and sinks (L) as, respectively, $T_1$ and $T_2$. Similarly, we express the requirement for DC buses: each of them must be capable powering all loads connected to it, i.e., have sufficient capacity.

Finally, a *reliability constraint* prescribes that the probability that a load gets unpowered because of failures should be less than a desired threshold. Some of the loads in our system are sheddable and the corresponding functional links must have a maximum failure probability of $10^{-5}$, while other, non-sheddable loads, have a tighter requirement and have to fail with a probability of at most $10^{-9}$. We use the pattern `max_failprob_of_connection(L,G,val)` to limit the maximum failure probability of functional links between loads (`L`) and generators (`G`) to `val`. It leverages the approximate encoding of reliability constraints, as discussed in Section 3.2.4.

Requirement patterns hide the details of the MILP formulation, which can be massive. Our specification for the EPN architecture exploration problem consists of only 46 instances of patterns and a total of 90 lines of code, including variable declarations and composition rules. The automatically generated MILP formulation for the monolithic optimization in standard form amounts to more than $100,000$ constraints and $20,000$ variables. The encoding for every iteration of the “lazy” optimization technique has around $5,000$ constraints and $1,500$ variables. Both resulting formulations are orders of magnitude larger than our specification. Moreover, these huge lists of linear inequalities are extremely complex to understand, refine and debug. This clearly shows the advantage of raising the level of abstraction of design capture using patterns.

### 5.1.3 Iterative Optimization: MILP Modulo Reliability

The MILP Modulo Reliability (MILP-MR) algorithm [12, 96] avoids the expensive generation of symbolic reliability constraints by adapting the MILP Modulo Theory approach [80] to reliability computations. This is done by applying Algorithm 2, i.e., generic iterative optimization scheme, to the reliability design viewpoint. A smaller MILP problem, without reliability constraints, is solved in a loop with an exact reliability analysis routine. The set of application constraints $\mathcal{R}_A$ includes only interconnection and balance constraints, for which SOLVEMILP generates minimum cost architectures. The analysis routine (calls to the RUNANALYSIS function in Algorithm 2) then computes the probability of composite failure events at critical nodes, starting from the failure probabilities of the components, a problem known as $K$-terminal reliability problem in the literature [77]. To do so, we apply a modified depth-first search algorithm from [99]. It traverses the graph $\mathcal{G}$ from the sink node $i$ (root) to the source nodes (leaves), by applying a path enumeration method, and by turning failure event relations into probability expressions. Although the $K$-terminal reliability problem is NP-hard, the key idea is to solve it only when needed, i.e., a small number of times, and possibly on smaller graph instances. The complexity of solving a MILP is also NP-hard. However, the actual runtime depends on the size of the MILP at each iteration, which is smaller than the one solved with the monolithic optimization scheme.

At each iteration of MILP-MR, if the obtained architecture satisfies the reliability constraints, it is returned as the final solution. The result is, in general, sub-optimal with respect to overall constraint set but the satisfaction of the reliability constraints is guaranteed by the exact reliability analysis. If the candidate architecture does not satisfy the reliability constraints, LEARNCONS (line 10 in Algorithm 2) estimates the number of redundant paths needed to achieve the desired reliability and suggests a set of strategies to implement the required paths by augmenting the original optimization problem with a set of interconnection constraints. Such strategies are subsequently deployed until the target failure probability is reached or no other strategies are available. This constraint learning function is, therefore, instrumental to efficiently converge towards a reliable architecture, while minimizing the number of calls to the analysis routine. We provide details about the implementation of LEARNCONS later in this section.

When no reliability constraints are enforced in the MILP, the solver attempts to use the minimum number of components and interconnections to perform a specific function at minimum cost. Typically, such a “minimal” architecture has also minimal redundancy, hence minimal reliability. Based on this intuition, we develop strategies that increase the reliability of the solution, albeit at a higher cost, by enforcing a larger number of redundant components and interconnections.

One of the strategies is summarized in Algorithm 3. It separately analyzes functional links for every sink (load) $s \in S$ and, if required, generates additional constraints that force the optimizer to add new components or connections in order to improve the reliability of the link (lines 4-16). Before doing this, FIXMAPPING (line 3) determines the components that are already used in the current version of the architecture (by looking at the mapping matrix $\mathbf{m}^*$) and constrains them to be instantiated with the same mapping on subsequent iterations. This is done for preserving the non-decreasing quality of subsequent solutions in terms of reliability. Next steps of Algorithm 3 enforce, in particular, new connections for components

**Algorithm 3:** LEARNCONS

```
Input: Current constraints $Cons$, reliability $r$, reliability requirement $r^*$, adjacency matrix $\mathbf{e}^*$,
       mapping matrix $\mathbf{m}^*$, functional flow $\mathcal{F}$
Output: Final constraints $Cons$
1  $NewCons \leftarrow [\,]$
2  $S \leftarrow$ GETSINKS($\mathbf{e}^*$)
3  FIXMAPPING($\mathbf{m}^*$)
4  forall $s \in S$ do
5      $k \leftarrow$ ESTPATH($r(s), r^*(s), \mathbf{e}^*$)
6      if $k \geq 1$ then
7          forall $T \in \mathcal{F} \setminus \{T_{sink}\}$ do
8              $AddedPaths \leftarrow 0$
9              $(AddedPaths, NewCons) \leftarrow$ CONNECTEXISTING($s$, $T$, $k$, $NewCons$, $\mathbf{e}^*$)
10             $\delta \leftarrow k - AddedPaths$
11             if $\delta > 0$ then
12                 $NewCons \leftarrow$ ADDCOMPONENT($T$, $\delta$, $NewCons$, $\mathbf{m}^*$)
13                 $(AddedPaths, NewCons) \leftarrow$ CONNECTNEW($s$, $T$, $\delta$, $NewCons$, $\mathbf{e}^*$)
14     else
15         $T_{min} \leftarrow$ FINDMINREDTYPE($s$, $\mathbf{e}^*$)
16         $NewCons \leftarrow$ FINETUNE($s$, $T_{min}$, $NewCons$, $\mathbf{e}^*$, $\mathbf{m}^*$)
17     if $NewCons = [\,]$ then
18         return Infeasible
19 $Cons \leftarrow Cons \cup NewCons$
20 return $Cons$
```

that are already instantiated. Without "freezing" the mapping, for the sake of minimizing the cost, the solver may decide to select a different subset of the components of the template $\mathcal{T}$ and, still being valid in the recently learned constraints, provide a less expensive, but less reliable topology. This contradicts with our goal of incrementally improving the solution.

The learning starts with the ESTPATH routine (line 5) that estimates the number of paths between the current sink (load) and the sources (generators) required to satisfy the reliability requirement, as shown in [12]. If one or more paths are needed, i.e., $k \geq 1$, then the strategy tries to make several improvements for every component type of the functional flow starting from the one closest to the sink (except the sink type itself). In particular, it first checks if the improvement can be made by adding additional contactors (edges), which has smaller cost with respect to adding new components. This can be done, for example, for DC buses, because they can be connected together. That is, function CONNECTEXISTING (line 9), by using the information from the current assignment on the variables of the adjacency matrix $\mathbf{e}^*$, determines the DC bus that is connected to the currently investigated sink. It then adds constraints that force additional connections between this DC bus and other instantiated buses. In our formulation, system organization and valid paths are ensured by defining interconnection constraints. That is, every instantiated (used) component of the architecture belongs to at least one path of at least one functional link. Then, by requiring to connect $n$ existing components together, we can ensure that all functional links related to components being connected will have at least $n-1$ additional paths. In other words, $n-1$ extra paths will be *implicitly* enforced.

The variable $\delta$ (line 10) is calculated as a difference between $k$ required extra paths and the number of paths added by connecting existing components, $\delta > 0$ meaning that the latter is not enough to meet the requirements. This can happen either if the requirement $r^*$ is very strict or if components of current type $T$ cannot be connected together by composition rules (e.g., rectifiers). In both cases, extra components of type $T$ (up to $\delta$) are added to the architecture (line 12). The assumption made in our learning strategies is that adding a new component will implicitly make it a member of some functional link, i.e., some path, as prescribed by the "if-then" interconnection constraints. Therefore, every new component will be automatically connected to some preceding and some succeeding component in $\mathcal{F}$. Moreover, the function also tries to connect newly added nodes to existing ones if this is allowed (line 13), which will ensure that the new path is accessible by the current functional link.

It may happen that EstPath returns $k = 0$, i.e., no more additional paths are needed or no more paths can be added due to architecture limitations to comply with the requirement $r^*$, as estimated by the function. However, $r^*$ is still not satisfied, because LearnCons has been called. In this case, Algorithm 3 attempts to fine tune the architecture. It determines the type $T_{min}$ of components, which has the minimal redundancy (line 15) and tries to increase the latter by either adding new components of $T_{min}$ or new connections between them (line 16). For example, if the maximum number of components in the EPN is already used and all the paths are connected together via contactors between DC buses, it is still possible to add extra contactors between AC buses to improve the reliability. If no new constraints are generated, i.e., no more paths can be added, Algorithm 3 returns `Infeasible`.

Let $\Pi$ be the partition of the template $\mathcal{T}$, $a$ and $b$ being the components of types $T_a$ and $T_b$ belonging to, respectively, sets $\Pi_a$ and $\Pi_b$ of $\Pi$. Formally, following constraints prescribe adding extra components and connections to the architecture:

$$\sum_{j=1}^{|\Pi_a|} \left( \bigvee_{i=1}^{|\mathcal{L}_a|} m_{ij}^{a} \right) \geq k + \sum_{j=1}^{|\Pi_a|} \left( \bigvee_{i=1}^{|\mathcal{L}_a|} m_{ij}^{a*} \right), \tag{5.1a}$$

$$\sum_{b \in \Pi_b} e_{a,b} \geq k + \sum_{b \in \Pi_b} e_{a,b}^{*}, \tag{5.1b}$$

where $\mathbf{e}$ and $\mathbf{e}^*$ are adjacency matrices for, respectively, template $\mathcal{T}$ and current architecture $\mathcal{G}^*$, same apllies for mapping matrices $\mathbf{m^a}$ and $\mathbf{m^a}^*$ for components of type $T_a$, and $\mathcal{L}_a$ is the subset of elements of the library $\mathcal{L}$ having type $T_a$. Constraint (5.1a) enforces $k$ additional components of type $T_a$ in the architecture by manipulating the mapping in a way that $k$ more components are required to be mapped, i.e., to be instantiated. Constraint (5.1b) adds $k$ extra connections from component $a \in \Pi_a$ to components of type $T_b$ from $\Pi_b$, which can be applied also for "horizontal" connections. One can use (5.1b) both for connecting existing components and enforcing additonal connections between new components, which implicitly forces the solver to add the components.

In a more general form, one can *explicitly* require $k$ additional paths between the sink $l$ and components of type $T_i$, belonging to the set $\Pi_i$, of the partition $\Pi$ of $\mathcal{T}$ with the following constraint:

$$\sum_{w \in \Pi_i} \eta_{w,l} \geq k + \sum_{w \in \Pi_i} \eta^*_{w,l}, \tag{5.2}$$

where $\eta$ and $\eta^*$ are the walk indicator matrices for, respectively, template $\mathcal{T}$ and current architecture $\mathcal{G}^*$. Moreover, the structure of the walk indicator matrix allows us to request new paths with a bound on the number of hops (edges) [96]. Despite the generality, using constraints of the form (5.2) may be expensive. Defining a walk indicator matrix and associating its variables with the basis variables requires a large number of auxiliary MILP constraints and variables, as discussed in [96]. Alternatively, if it is admitted by the composition rules, we can simply require adding new components and direct connections (edges), thus operating only on basis variables. With respect to the LEARNCONS algorithm presented in [12], we only replace the ADDPATH function that implements Constraint (5.2) with a set of actions for adding components and connections, as formalized by Constraints (5.1a)-(5.1b). These actions support our extended formulation (e.g., with respect to previous works we employ component subtypes in the architecture), and, same as in [12], ensure a nondecreasing sequence of system costs on subsequent iterations, because every time either nodes and edges are added or `Infeasible` is returned. Added components and connections will induce additional paths that help to satisfy the reliability requirement. Since the size of $\mathcal{T}$ is finite, eventually the iterative optimization (Algorithm 2 instantiated as MILP-MR) will terminate. Hence, our modifications do not violate the soundness and completeness of MILP-MR stated by the corresponding theorem in [12].

Finally, we note that at every iteration of Algorithm 3 some actions are taken for every component type in $\mathcal{F}$. That is, multiple components and connections can be added in the next generated architecture. On the one hand, this allows the algorithm to faster converge to a satisfying solution in terms of reliability requirement. However, resulting architecture may be over-designed, e.g., when too many new connections have been added, while $r^*$ can be satisfied by only using a part of them. Therefore, LEARNCONS can also implement a lazier strategy, which adds a single constraint at every iteration and makes it possible to more carefully tune the architecture and to avoid over-design. Such approach, however, might have much more iterations, which affects the execution time. It can be applied, for instance, when minimizing the objective is a high priority goal.

### 5.1.4 Numerical Evaluation

We evaluate both monolithic and iterative (MILP-MR) optimization techniques provided by ARCHEX by synthesizing complex EPN architectures based on the specification provided in Section 5.1.2. All the numerical experiments reported below have been performed on an Intel Core i7 3.4-GHz processor with 8-GB RAM running Ubuntu 16.04.

By using the monolithic optimization approach, ARCHEX generates the EPN topology shown on Figure 5.2a in about 2.5 h. In this architecture, only horizontal connections between DC

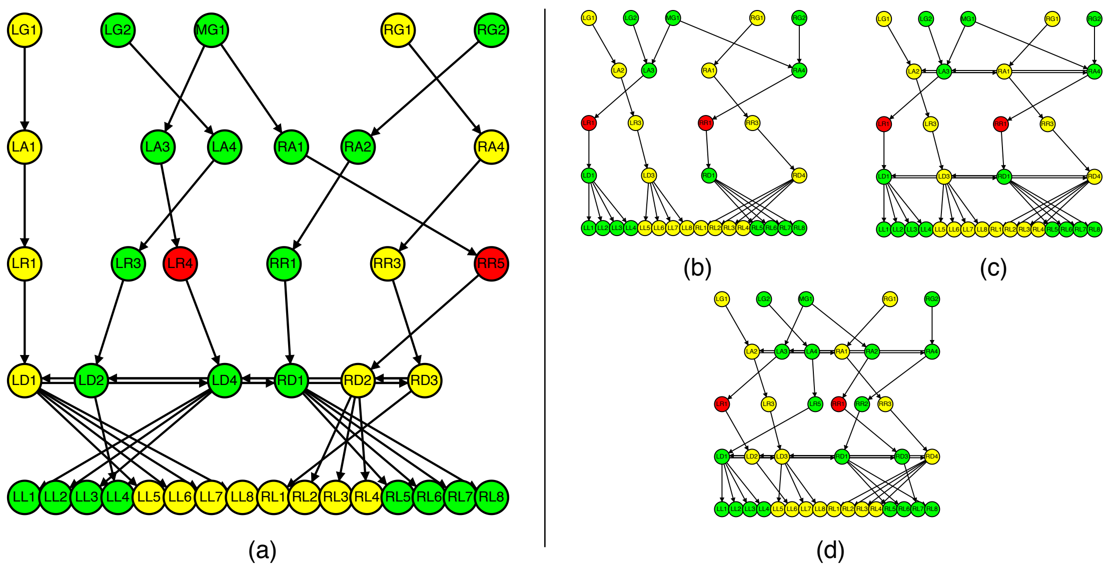


Figure 5.2 – EPN architectures generated by ArchEx (a) Monolithic optimization: $r = 0.5 \times 10^{-9}$, $\tilde{r} = 0.96 \times 10^{-10}$; (b) First iteration of MILP-MR: $r = (0.6, 0.8) \times 10^{-3}$ for (HV,LV) loads; (c) Second iteration of MILP-MR: $r = (0.2, 0.32) \times 10^{-6}$ for (HV,LV) loads; (d) Third (last) iteration of MILP-MR: $r = (0.38, 0.19) \times 10^{-9}$ for (HV,LV) loads.

buses are added to increase the system reliability, while AC buses are not connected. Green and yellow nodes of the graph represent HV and LV components, respectively. Red components are TRUs connecting the HV and LV portions of the system (can also be used to connect buses with the same voltage level). Unused nodes are not shown. Horizontal connections between DC buses increase the system reliability, by creating redundant paths from loads on one side of the system to sources on the other side or APUs. The resulting failure probability is $0.5 \times 10^{-9}$ for every functional link and the overall cost is 106,000. The approximate algebra provides an estimation $\tilde{r} = 0.96 \times 10^{-10}$ of the failure probability which is smaller than the value obtained by exact computation, but it is well within the error bound introduced in [96]. Furthermore, both exact and approximate values satisfy the reliablity requirement.

By using the iterative approach (MILP-MR), the problem is solved in 3 iterations as summarized in Fig. 5.2b-5.2d. By solving for just the interconnection and power flow constraints, we obtain the simplest possible architecture (Fig. 5.2b), which only provides a single path from every load to a generator (or APU), thus showing the highest failure probability. Because increasing the number of components is expensive, the algorithm first tries to increase the reliability by adding connections among existing components at the cost of additional contactors. As shown in Figure 5.2c, contactors are added between existing AC and DC buses (shown as horizontal connections) thus allowing the generators on the left side to be accessible by the loads on the right side (and vice versa). Rules of connecting different subtypes are preserved: HVDC bus is connected to another HVDC, same for the low voltage buses. Since the requirement is not yet satisfied, a third iteration is used, and two extra DC buses are added, one HV and one LV for each type, as well as two extra HVAC buses, and connected to existing buses (Figure 5.2d). The resulting failure probabilities are $(0.38, 0.19) \times 10^{-9}$ for the (HV, LV) functional links. The cost is 108,000, slightly higher than the one obtained with eager optimization, for a total execution time of 56 s, of which 98% are used for the problem formulation.

### 5.1.5 Scalability

We also tested the performance of ArchEx 2.0 on the benchmarks used in [12] by designing EPN architectures with an increasing number of components. In Table 5.2, we report on the execution time of the MILP-MR iterative technique based on Algorithm 2 and using the learning function LearnCons as in Algorithm 3. Results for the eager optimization are reported in Table 5.3. Clearly, the iterative approach outperforms the eager one for architectures with more than 20 nodes. On the other hand, once the optimization problem is generated for a given template, monolithic optimization is competitive for smaller architectures. Yet, problems with several tens of thousands of constraints, and including a realistic number of generators (normally less than 10), can still be formulated and solved in a few hours.

We can also point out that the complexity of the monolithic optimization (in terms of MILP consraints and decision variables) grows much faster with the size of $\mathcal{T}$, compared to the iterative one. This is because the latter only uses interconnection and balance constraints in the formulation, which are simple and only use the basis variables, possibly with few auxiliary ones for linearizing some terms in the expressions. Conversely, the approximate encoding of reliability constraints used in the eager approach additionally uses walk indicator matrices. The formulation of these auxiliary data structures involves computing powers of the adjacency matrix according to the definition from [12]. The latter consists of binary decision variables and every their product has to be replaced with a new variable and a set of linearization constraints. Their number grows rapidly when increasing the size of $\mathcal{T}$. Therefore, the biggest overhead in the complexity of the formulation comes with reliability constraints, while decoupling them allows obtaining much simpler formulations and solve them iteratively, as done by MILP-MR.

We also note that the longest execution time for MILP-MR in Table 5.2 (12 s) is several times smaller compared to the one reported in Section 5.1.4 (56 s). This is because interconnection constraints related to separation of HV and LV parts of the architecture were not used in the scalability tests. In the case studies in Section 5.1.4 they introduced additional timing overhead both for setting up the formulation and for the learning function.

We also ran a set of experiments with both solving techniques on a template of a fixed size while varying the size of the library $\mathcal{L}$. Results for both eager and lazy algorithms are shown in Table 5.4. They confirm our discussion in Section 2.2.1: the number of mapping variables (not shown in Table 5.4) grow linearly with $|\mathcal{L}|$. At the same time, the growth trend of the overall problem complexity (both variables and constraints) is logarithmic in $|\mathcal{L}|$. Same results have been observed for larger $|\mathcal{T}|$, which confirms the efficiency of the proposed separation of the mapping problem.

Table 5.2 – Number of MILP constraints and variables, number of iterations, reliability analysis and solver time for different EPN architecture sizes ($r^* = 10^{-11}$) generated using integer linear programming modulo reliability (MILP-MR) with LEARNCONS (Algorithm 3).

| $\lvert\mathcal{T}\rvert$ (# Generators) | #Constraints / #Vars | #Iterations | Analysis time (s) | Solver time (s) |
|---|---|---|---|---|
| 20 (4) | 621 / 196 | 3 | 2 | 0.5 |
| 30 (6) | 1255 / 364 | 3 | 3 | 1 |
| 40 (8) | 2109 / 580 | 3 | 6 | 3 |
| 50 (10) | 3183 / 844 | 3 | 11 | 5 |
| 60 (12) | 4477 / 1156 | 3 | 12 | 12 |

Table 5.3 – Number of MILP constraints and variables, problem generation (setup) and solver times for different EPN architecture sizes ($r^* = 10^{-11}$) generated using monolithic optimization with approximate reliability constraints.

| $\lvert\mathcal{T}\rvert$ (# Generators) | #Constraints / #Vars | Setup time (s) | Solver time (s) |
|---|---|---|---|
| 20 (4) | 3677 / 1372 | 12 | 1 |
| 30 (6) | 14371 / 4576 | 22 | 12 |
| 40 (8) | 40701 / 11956 | 30 | 494 |
| 50 (10) | 93803 / 24864 | 108 | 1219 |
| 60 (12) | 187885 / 51340 | 253 | 19776 |

Table 5.4 – Problem complexity (number of constraints and variables) and solver time for exploration problems with different sizes of library $\mathcal{L}$ and a fixed-size template $\mathcal{T}$ (20 nodes) solved using monolithic and iterative optimization approaches.

| $\lvert\mathcal{L}\rvert$ | #Constraints / #Variables | | Solver time (s) | |
|---|---|---|---|---|
| | Monolithic | Iterative | Monolithic | Iterative |
| 10 | 3677 / 1372 | 621 / 196 | 1 | 0.5 |
| 25 | 3780 / 1451 | 724 / 275 | 1.2 | 0.5 |
| 50 | 3895 / 1566 | 839 / 390 | 1.3 | 0.5 |
| 75 | 3974 / 1651 | 922 / 482 | 1.6 | 0.6 |
| 100 | 4075 / 1746 | 1019 / 570 | 2 | 0.7 |

Overall, we infer that eager approach turns out to be preferable when we aim to a coarser estimation of the capability (and limitations) of an architecture template and a platform library in terms of reliability. Furthermore, its solutions are proven to be either optimal or within a known gap from the optimum (this can be configured for a given MILP solver as shown in Section 4.4.3), albeit with respect to an approximation of reliability constraints. On the other hand, MILP-MR makes it easier to incorporate domain-specific knowledge, since a designer can customize the techniques adopted to improve reliability at each iteration. Moreover, MILP-MR becomes the preferred choice, especially for larger problem instances, when we can estimate the number of redundant paths needed to satisfy the requirement as early as possible, or when we are willing to pay for a longer execution time to incrementally fine tune the reliability. On the whole, design projects that have tighter timescales may opt for the iterative technique, while the ones with a sufficient time budget and stricter need for minimizing the objective and guaranteeing the optimality may prefer the monolithic approach.

## 5.2 Reconfigurable Manufacturing System

### 5.2.1 Overview and Problem Statement

The recent concept of "Industry 4.0" advocates the usage of CPSs in factory automation as a major goal [88, 53, 73]. We demonstrate the effectiveness of our exploration methodology by applying it to a related industrial case study - optimized selection of reconfigurable manufacturing system (RMS) architectures. Such production systems are able to quickly adjust their functionality to respond to sudden market demand changes or unexpected machine failures [67, 39]. As exemplified on Figure 5.3, an RMS consists of a source that provides parts to be processed (assembled, packaged) on the line, a set of Computer Numerical Control (CNC) or Reconfigurable Machine Tools (RMT) connected by conveyors, and a sink that collects the final product. A typical RMS can have several sub-lines to process different product types or different parts (details) of the same family. These sub-lines are connected together with junction conveyors or gantries [66]. Reconfiguration is, in particular, related to having different *operation modes*. For instance, at some time, manufacturing of the product on line 1 is not required, while there is an increased demand for another product, processed by line 2. Instead of installing a fully parallel sub-line for the second product, which can be costly, it is possible to reuse (reconfigure) line 1 to increase the throughput. In this case, line 1 must use RMTs, which accept both product types and hence support both operation modes.

The possibility of an RMS to reconfigure is also useful for reliability purposes, so that in the event of component failure production units are rerouted to other active sub-lines in order to be processed without stalling the whole system, because the latter may lead to severe financial losses. This is similar to switching the contactors in the previous case study (EPN) in order to keep active the functional links between product sources and sinks. In other words, if there is a number of additional paths (sub-lines) for processing production units (for each product type) then the reliability of the RMS is increased.

**Requirements**. The design of RMS suggests a number of different concerns. From the *structural* viewpoint, a particular set of manufacturing devices (e.g., product sources, sinks/collection stations, certain types/subtypes of machinery) and interconnections between them must be present in the system architecture. Moving of production units (details, parts) along the manufacturing (assembling, packaging) lines of RMS can be seen as a *flow* of products. Therefore, components of the RMS are subject to *flow balance requirements*, so that flow rates are correctly split between different sub-lines. Moreover, the processing segment of the system, i.e., machines, also have *workload requirements* so that input flows rates of production units to a machine never exceeds its processing capabilities.

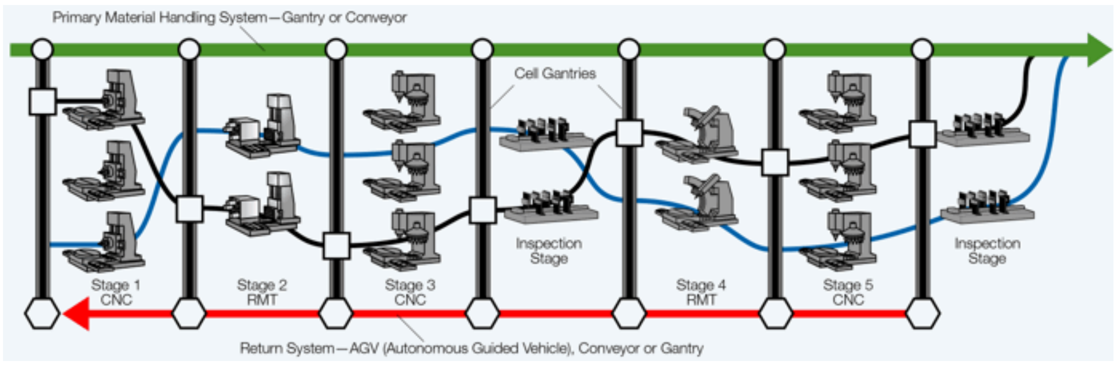

Primary Material Handling System—Gantry or Conveyor
Cell Gantries
Inspection Stage
Inspection Stage
Stage 1 CNC
Stage 2 RMT
Stage 3 CNC
Stage 4 RMT
Stage 5 CNC
Return System—AGV (Autonomous Guided Vehicle), Conveyor or Gantry

Figure 5.3 – Example of a reconfigurable manufacturing system (RMS) [1].

RMS also have *timing requirements*, which set up upper bounds on the product cycle time, i.e., the time it takes for a production unit to pass all manufacturing stages, i.e., to propagate through all the machines from the source to the sink. Furthermore, product flows and machines have to be chosen in a way that the idle rate of the latter is minimized or does not exceed a threshold. This is one of the major goals of load balancing in manufacturing since the stalled machinery has similar maintenance and service costs but does not produce any value. In many cases it is very unprofitable. For example, silicon foundries aim to configure their processes to maximally utilize the production lines, because this can cover the costs of the extremely expensive maintaining of the "clean" rooms.

*Reliability requirements* for manufacturing systems are similar to the ones from power systems. Functional links of RMS can be interpreted as sets of manufacturing sub-lines for each product type, or, similarly, as different manufacturing processes. If at least one of them for a given product is *active* (none of the components along the line is broken or malfunctioning) then the link is considered active as well. The requirement simply prescribes that the probability of a functional link to become *inactive* must not exceed a threshold. It can be separately enforced on every line (for certain product type) in every operation mode.

**Exploration Problem Statement**. The goal of the exploration problem is to select the proper amount and types of machinery and connect them together with conveyors and/or gantries. The system is subject to structural, interconnection, flow balance, workload, timing and reliability requirements, while the optimization objective is either to minimize the total cost of machines, or their total idle rate, or both (a weighted combination). The resulting architecture will serve as an input to the controller synthesis problem. RMS control algorithm is responsible for switching operation modes, dynamically adjusting and distributing the product flows between machines, enabling and disabling the equipment. These operations are performed based on the information provided by a set of sensors that monitor the state of the production lines, as well as on the external requests, e.g., product demands.

[1]**Figure courtesy of** Rod Hill, graphic designer at the University of Michigan.

### 5.2.2 Specification

Using ArchEx, we create a specification for an RMS as follows. The template $\mathcal{T}$ includes 4 component types: Source (SRC), Machine (M), Conveyor (C) and Sink (SNK). Sources and sinks, respectively, provide new production units and collect assembled products. Therefore, they can be also considered as machines, while we distinguish them for a more intuitive formulation. Nodes of $\mathcal{T}$ represent these components, and edges between components of different types are connectors that join conveyors and machines. Conveyors can be connected together to split the input flow of production units. These *horizontal* connections represent gantries or junction conveyors. The system is manufacturing two different products, A and B, each of them having two machines along the production path (M1 and M2). Components in $\mathcal{T}$ and library $\mathcal{L}$ are labeled with following attributes: cost $\mathbf{c}$, product flow rate $\lambda$, failure probability $\mathbf{p}$. Machines are also characterized by their throughput $\mu$. The maximum number of components of each type on each production line as well as the summary of $\mathcal{L}$ is given in Table 5.5. As there is only one component type in $\mathcal{L}$ to implement a machine, we use the subtypes A, B, and AB to, respectively, categorize the machines that can be used only for product A, B, or both. The RMS must support two operation modes. In mode $\Omega_1$, both products A and B must be simultaneously produced with rates $\lambda_A$ and $\lambda_B$. In mode $\Omega_2$, A is produced with a double rate, $2\lambda_A$, while line B is stalled. We assume that $\lambda_A$ and $\lambda_B$ are fixed and that conveyors can automatically adjust to any input rate. Finally, failure probability is $2 \times 10^{-4}$ for machines and conveyors and 0 for sources and sinks.

Similarly to the EPN example, we specify the interconnection constraints (e.g., "Every source must be connected to exactly one conveyor" or "If conveyor's input is connected to a product source or a machine then its output must be connected to another machine or a sink") using the same patterns. We further impose balance constraints using the `flow_balance` pattern for conveyors and machines. It requires the total input flow to be equal to the output flow for these components, i.e., it ensures the correct distribution of product parts on the line. The workload constraint `no_overloads(M)` is added for machines. We also use the mapping pattern `in_flow_implies_mapping_to(M1,A,B,AB)`: if machine $M1$ installed on line $A$ receives products of $B$ in some operation mode, then it must be an RMT, i.e., a corresponding subtype (AB) must be selected from the library to implement it. Similar constraints are written for other product types and machines (e.g., $M2$).

We set up the two operation modes $\Omega_1$ and $\Omega_2$ using `has_operation_mode(1A + 1B,0A + 2B)`, which is an *operation pattern* currently implemented only for using it in RMS specifications (not listed in Table 4.1). Its arguments represent the two modes by specifying the coefficients for each product type. These coefficients multiply the original flow rates for these types (given in Table 5.5). For example, `1A` means that the original rate has to be used, `2A` entails the double rate of A, `0B` marks the line B inactive in $\Omega_2$. The pattern labels the edges of the template $\mathcal{T}$ with flow rates in both operation modes. More precisely, it creates the *flow rate matrices* (objects of the QuantityMatrix class) $\Lambda^{k,x}$ with $k \in \{\Omega_1, \Omega_2\}$ and $x \in \{A, B\}$, where $\lambda_{ij}^{k,x}$ is a real decision variable representing the flow rate of product type $x$ in operating mode $k$ along the

Table 5.5 – Summary of template and library for the RMS example. Flow rates $\lambda$ and throughputs $\mu$ are shown for, respectively, sources and machines. Notation $\{a \dots b\}$ indicates that all values are within the interval between $a$ and $b$.

| **Type** | **Number in $\mathcal{T}$** | **Cost,** | **Fail prob.** | **Flow rate $\lambda$ or throughput $\mu$** (parts/min) | | |
|---|---|---|---|---|---|---|
| | (A,B) | $\times 10^3$ | | A | B | AB |
| Source | 1,1 | 0 | 0 | 12 | 10 | - |
| Machine | 3,2 | {2,3,...,15} | $2 \times 10^{-4}$ | 3,6,20 | 3,5,13 | 10 |
| Conveyor | 3,2 | 0.5,1 | $2 \times 10^{-4}$ | - | - | - |
| Sink | 1,1 | 0 | 0 | - | - | - |

edge $e_{ij}$. $\Lambda^{\Omega_1,A}$ and $\Lambda^{\Omega_1,B}$ set to zero all the flow rates between components associated with different product types, as no line can be borrowed for another product. This is not the case for $\Lambda^{\Omega_2,A}$, since the line associated with product B may be reused for product A in mode $\Omega_2$. Finally, $\Lambda^{\Omega_2,B}$ is a matrix of zeros. Flow rate matrices are used in computing idle rates of the machines, as well as in balance, workload and reliability constraints.

In one of the examples we also specify a bound of 10 parts/min on the sum of idle rates of all the machines (timing constraint) by using the `max_totat_idle_rate` pattern. Finally, we use `max_failprob_of_connection` to set up the maximum failure probability of production lines A and B in $\Omega_1$ and $\Omega_2$. For RMS, we have extended the signature of this reliability pattern, so that it also accepts the operation mode (e.g., “1A+1B”) as an argument. Also, we have modified the way of computing the approximate reliabilities of functional links. The same expression (3.9) (see Section 3.2.4) is used, however, for RMS we define degrees of redundancy $h_{ij}$ in a different fashion. Instead of being the number of components of type $j$ used in at least one path of a functional link $F_i$, as defined in [96], the redundancy is now calculated as the number of components of type $j$ that have incoming flow of products processed on the current line (link). More formally, if $\Pi_i$ is the set from partition $\Pi$ of $\mathcal{T}$ with all components of type $i$, then:

$$h_{ij} = \sum_{k \in \Pi_i} \delta_k^j, \tag{5.3}$$

where $\delta_k^j$ is a Boolean variable equal to one if component $k \in \Pi_i$ has an input flow $\lambda^j$ of product $j$ greater than zero, i.e., it is used on the corresponding production line (manufacturing process). Paths between such components and the sink are implicitly provided by the interconnection and flow balance constraints. The structure of RMS assumes that every product type is eventually delivered to a single collection point (sink). Therefore, according to our constraints, each instantiated component (machine, conveyor) that processes or transfers certain production units, will be connected to the corresponding sink, which allows us to use the number of such components as a degree of redundancy. Such approach does not require using walk indicator matrices that provide the information about the existence of paths, as we do for the EPN case study. Hence, we can avoid a large overhead, that comes with encoding of these matrices, and obtain a more compact formulation.

### 5.2.3 Optimization Results and Performance Analysis

Figure 5.4 shows four different RMS architectures, with slightly different requirements, generated by the monolithic optimization. Numerical results are summarized in Table 5.6. For the first one (Figure 5.4a; no reliability or timing constraints enforced), single source-sink path for each of the lines is enough, however, line A gets additional reliability by connecting to line B. This is done to satisfy the workload constraints for two given operation modes, in particular, to support $\Omega_2$. Part of the flow of product A in this mode is redirected to line B (C1A2 $\rightarrow$ C1B2), where reconfigurable machines (M1B1,M2B1), marked by a red color, are installed. Processed parts of product A are then sent back to Sink A (C3B2 $\rightarrow$ C3A1 $\rightarrow$ SnkA). For the given template $\mathcal{T}$, library $\mathcal{L}$ and requirements $\mathcal{R}_A$, reusing line B is more cost-effective than installing additional conveyors and machines on line A.

The second architecture, shown on Figure 5.4b, has a reliability requirement for the nominal operation mode $\Omega_1$ for both A and B: $r^*_{A,\Omega_1} = r^*_{B,\Omega_1} = 10^{-5}$. Therefore, additional components are added on lines A and B (in particular, for line A this is done because its reliability in $\Omega_1$ cannot be improved by reusing line B). As a result, the input flow rate of $\Omega_2$ can be split between machines on line A, i.e., line B is no longer reused by A in this mode, which would have larger cost due to the need of installing expensive reconfigurable machines. Enforcing a tighter requirement $r^*_{A,\Omega_2} = 10^{-7}$ for $\Omega_2$ leads to an architecture shown on Figure 5.4c. Here, lines A and B are again connected, and RMTs are installed on line B. This provides the third path from SrcA to SnkA in $\Omega_2$, which has smaller extra cost than adding extra components on line A. The cost of the system is higher with respect to the second example despite the same number of components, because reconfigurable machines (with subtype "AB") are more expensive than others in the library $\mathcal{L}$.

The last architecture (Figure 5.4d) is generated with a timing constraint: we set up a bound of 10 parts/min for the cumulative idle rate of machines in $\Omega_1$ and $\Omega_2$. Line B is still reused in $\Omega_2$, however, it is now more convenient to implement M1 and M2 on line A by inserting two additional machines in parallel. Both of them are cheaper and have slower processing speed, but as a result we achieve a total idle rate of 8 parts/min, which is a 3.5x reduction compared to the first architecture. We have not used reliability constraints for this architecture, however, line A gets a very high level of reliability as a side-effect of adding more machines. Clearly, it has the biggest number of components among the four examples and, therefore, it is the most expensive from the economic viewpoint. We further note that idle rate can be also added to the objective function either as a single cost or in combination with the component price. In such a way it is possible to explore the tradeoff between the two concerns. For the given setup, however, optimizing for idle rate (using it as an objective) results in the same architecture, i.e., the idle rate cannot be made lower than 8 parts/min in any possible configuration.

Complexity and solving time for provided examples are summarized in Table 5.7. Augmenting the formulation with reliability constraints (architectures 2 and 3) does not result in significant increase of the complexity because, as discussed in Section 5.2.2, walk indicator matrices are not created. Instead, degrees of redundancy of components as well as their approximate

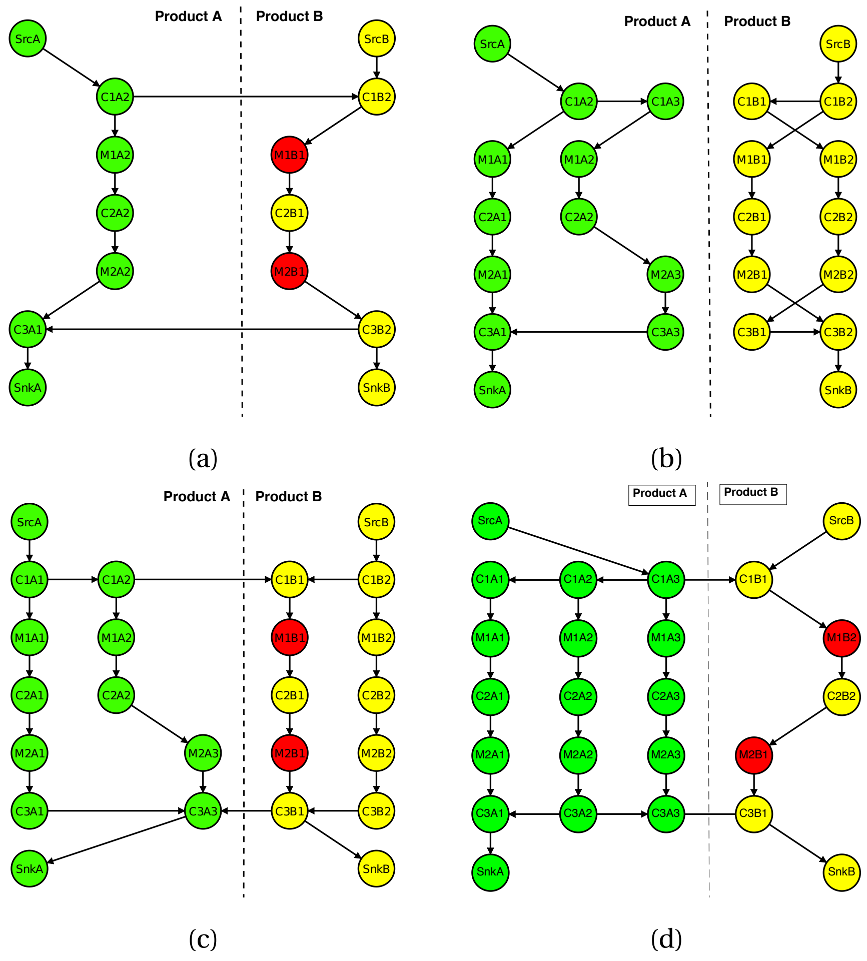


Figure 5.4 – RMS architectures generated by ArchEx: (a) No reliability or timing constraints; (b) With reliability constraints: $r^*_{A,\Omega_1} = r^*_{B,\Omega_1} = 10^{-5}$; (c) With tighter reliability constraints: $r^*_{A,\Omega_1} = r^*_{B,\Omega_1} = 10^{-5}$ AND $r^*_{A,\Omega_2} = 10^{-7}$; (d) Timing constraint - bound on the total idle rate of machines (3.5x reduction achieved).

reliabilities are computed using flow rate matrices, which are used in every RMS formulation. The total time of setting up and solving the problem is on the order of seconds. This demonstrates that using ArchEx for architecture selection of reconfigurable manufacturing systems is efficient. Moreover, our specification file consists of 67 instances of 13 patterns, while as evident from Table 5.7, the size of each MILP formulation is two orders of magnitude larger. This again confirms the advantage of using the pattern-based language, and the usability of ArchEx.

We have also evaluated the iterative approach for solving RMS exploration problems. The implementation of the learning function LearnCons has been modified with respect to Algorithm 3, so that the routine goes through every functional link in every operation mode. We also leverage the fact that each product line has a single sink. If some component accepts the input flow of some product type, interconnection and flow balance constraints ensure

Table 5.6 – Numerical results for generated RMS architectures: cost, reliabilities, idle rate.

| **Architecture** | **Cost** | **Idle rate,** | **Reliability** | | |
|---|---|---|---|---|---|
| | | parts/min | $r_{A,\Omega_1}$ | $r_{B,\Omega_1}$ | $r_{A,\Omega_2}$ |
| 1 | 45000 | 28 | $10^{-3}$ | $10^{-3}$ | $0.84 \times 10^{-6}$ |
| 2 | 58000 | 52 | $0.84\times10^{-6}$ | $0.84\times10^{-6}$ | $0.84 \times 10^{-6}$ |
| 3 | 64000 | 44 | $0.84\times10^{-6}$ | $0.84\times10^{-6}$ | $0.68 \times 10^{-9}$ |
| 4 | 69000 | 8 | $0.68\times10^{-9}$ | $10^{-3}$ | $0.58 \times 10^{-12}$ |

Table 5.7 – Problem complexity and solver time for RMS design examples.

| **Architecture** | **#Constraints / #Variables** | **Solver time (s)** |
|---|---|---|
| 1 | 4430 / 4673 | 0.5 |
| 2 | 5266 / 4851 | 3 |
| 3 | 5266 / 4851 | 25 |
| 4 | 4431 / 4673 | 2.5 |

that this component has a path to the sink. Therefore, instead of adding components and connections, the function proposes to increase the number of components that have a non-zero input flow of a product, for which the corresponding functional link does not yet meet the reliability requirement. This is flexible, because it does not explicitly require to add an additional sub-line or to reuse the line for another product. This choice is left for the solver. Intermediate conveyors (e.g., C2 in our template) can also be connected together to fine tune the architecture.

We used MILP-MR to synthesize an architecture with same reliability requirements as in the second example above ($r^*_{A,\Omega_1} = r^*_{B,\Omega_1} = 10^{-5}$). Since iterative problem modulo reliability does not include the latter as a constraint, on the first iteration we obtain the same topology as on Figure 5.4a. Reliability analysis then reveals that none of the two requirements is satisfied, therefore, LEARNCONS generates additional constraints that request to increase the number of components serving product A and product B. The second iteration results in an architecture similar to the one shown on Figure 5.4c. Both $r^*_{A,\Omega_1}$ and $r^*_{B,\Omega_1}$ are now SAT, however, the system cost is more expensive compared to the monolithic optimization. The reason is the impact of the FIXMAPPING function in Algorithm 3, which does not allow to modify the mapping of existing components. That is, reconfigurable machines selected on line B during the first iteration, cannot be replaced with the ones that handle only product B. Therefore, the solver can only add new components on both lines A and B, which results in over-design. The total number of components in the resulting architecture is equal to the one on Figure 5.4b generated with the same requirements by the eager approach, but system cost is higher because reconfigurable machines are used, and lines A and B are connected together.

If modifications of the mapping are not restricted, i.e., FIXMAPPING from Algorithm 3 is skipped, then the aforementioned problem is solved in two iterations and provides the same result as the monolithic optimization (Figure 5.4b). However, the performance of the MILP-

MR is slower, because at every iteration it solves a problem of almost the same size as the monolithic one. This is achieved by modifying the encoding of approximate reliabilities used in the eager approach as described in Section 5.2.2. With this modification, the overhead of reliability constraints is much less compared to the EPN case study, where the separation resulted in several orders of magnitude difference. We note, however, that our modified encoding was created under the assumption that production lines for each product type have a single sink, which is typical for RMS. If this is not the case, the original definition of degrees of redundancy [96] has to be applied. This will lead to heavy monolithic formulations, and the iterative optimization (MILP-MR) will become preferable in terms of complexity and speed. Moreover, integrating more domain-specific knowledge into the implementation of LEARNCONS can provide the possibility of more careful tuning of the architecture at every iteration, which is especially useful for large-scale designs.

## 5.3 Related Work

While the related work for CPS design methodologies and tools has been discussed in Section 1.4, below we put the presented applications of our architecture exploration methodology and the obtained results in the context of existing works in corresponding domains.

**Aircraft Electrical Power Networks**. Several recent works have tackled the problems of design space exploration, synthesis, optimization and performance analysis of aircraft electrical power networks. An optimization-oriented power system design methodology following the platform-based design paradigm was proposed in [103], where initial specifications are refined and mapped to the final implementation in four steps. In particular, the first two steps, generator selection and distribution network synthesis, build up the architecture exploration part of the design flow. Each of them formulates a binary optimization problem, with results of the first step being used in the second. In our approach, the generator selection problem is extended to a more general *mapping problem*, which is also decoupled from topology selection. However, in our formulation we *jointly* solve the two problems. We also provide an extensible toolbox for formulating and solving these problems, while all the constraints from [103] can be easily incorporated in our framework and pattern-based language.

The aforementioned design flow has been extended in [99] toward a more holistic approach for power system design, which enables systhesis of electrical power system topology and control, subject to hererogeneous sets of system requirements. The latter are not always approximated by binary or mixed integer linear constraints. However, the architecture exploration problem in [99] is regarded as a MILP optimization problem. It is further extended in [12] with the two optimization techniques, MILP with Approximate Reliability (MILP-AR) and MILP Modulo Reliability (MILP-MR), which have been implemented in the first prototype of ARCHEX. The former creates a monolithic optimization problem for aircraft EPN architecture selection, based on approximate reliability constraints proposed in [96]. The latter is the iterative optimization discussed in Sections 3.4 and 3.5.2 and used in both case studies in this chapter. Overall, our exploration methodology builds on the seminal works by Nuzzo et al. [99, 12].

In the context of this chapter, our approach is both more general and more efficient that the one in [99, 12], since we support a richer set of requirements (e.g., timing, workload), achieve problem formulations for power systems with up to one half of the constraints reported in these works, and 2-4x faster execution speeds. Moreover, with respect to its predecessor, ArchEx 2.0 [62] can efficiently generate more complex architectures, e.g., including HV and LV power distributions. Finally, the generality is achieved with our encoding of mapping constraints (its advantage was discussed in Section 2.2.1). In the broader scope, we are also able to apply our methodology and the ArchEx framework to other classes of CPS, such as manufacturing systems, fuel distribution systems and wireless networks.

A mixed discrete-continuous optimization scheme has been proposed in [40] with an application to selection and sizing of components of an aircraft environmental control system. This work involves a MILP-based topology selection executed in a loop with continuous sizing routine. The former provides candidate configurations, which are optimized by the latter over the space of continuous system propertes by running a set of simulations. Our approach is different in that it includes continuous parameters (e.g., flow rates) in the MILP formulation. Continuous properties of physical systems can be captured in our framework by static resolution and approximation of the dynamic behavior using the techniques, such as the one discussed in Section 2.4.2.

In [78], EPN control problems are formulated and solved as MILP optimization problems to yield load shedding, source allocation, contactor switching and battery charging policies, while optimizing a number of performance metrics, such as the number of used generators and shed loads. The authors model such system properties as instantaneous load power, battery dynamics and contactor switching as MILP constraints and generate policies for the load management system within a receding horizon approach. Control problems are out of scope of the architecture exploration methodology and are currently not supported by ArchEx. However, we consider them an interesting future work for extending the framework with new *control patterns*, which automatically translate the constraints as in [78] to a MILP formulation. Many control requirements can be generalized from electrical power systems to a more broader category of designs, which would allow ArchEx to contribute also to control design problems.

Finally, several works advocate the adoption of simulation for the analysis of aircraft power systems performance [13, 69]. Simulation models can capture various system properties, such as the ones of the power system, at different levels of complexity. These approaches, however, are more focused on verification and not on synthesis, i.e., an architecture has to be generated first. In contrast, the scope of our methodology and framework is correct-by-construction architecture selection. Moreover, design space exploration, optimization and analysis using simulation-based techniques can still become unaffordable from the computational standpoint unless proper levels of abstraction are devised.

**Reconfigurable Manufacturing Systems**. Our methodology is able to tackle several steps of high-level, conceptual design phase of manufacturing systems: equipment selection (number and type of machines), configuration selection (the way machines are arranged and interconnected) and process planning (distribution of product flows) [66]. Among the other related activities, they constitute the so-called *basic design* phase of RMS [8].

Several approaches have been proposed for optimizing the configuration of reconfigurable manufacturing systems of different complexity levels. In particular, an ILP-based design methodology for scalable machining system using a partial enumerative procedure has been proposed in [122, 121]. In these works, different system interconnections are explored in order to minimize the total life cycle cost of the system. However, the authors made an assumption that all the machines and other production modules are identical, which is unrealistic in modern RMS. Later on, various complex optimization problems have been proposed for optimized selection of RMS configurations, where availability of machines [131], reconfigurability [43] and different operation modes [32] are taken into account. Formulated problems are typically multiobjective nonlinear programming models, while several techniques are proposed for solving them, including genetic algorithms [43, 32] and tabu search [131].

In spite of the variety of existing approaches for the concept design of RMS, most of them are still either generic or very sophisticated to be used by system designers. The reason for the latter is the lack of supporting tools that allow to leverage the full power of these proposed techniques. For example, adaptation of one of the optimization problems [131, 43, 32] to a particular design by manually manipulating the mathematical formulation of constraints can be utterly complex. In this light, the advantage of ARCHEX is its *usability*, i.e., the possibility of writing design specifications for RMS configuration in a compact intuitive way using patterns. ARCHEX is in its infancy in the RMS domain, because it is a tool that rather supports a generic architecture exploration methodology. However, it is already capable of capturing a lot of concerns typically accounted for during the basic phase of RMS design, while the extensibility of the framework allows it to be customized to become a powerful tool in this domain.

## 5.4 Conclusions

In this chapter we have demonstrated the effectiveness of our methodology and the ARCHEX framework on *reliability-driven* designs in two different CPS domains, avionics and manufacturing. We were able to synthesize correct-by-construction cost-effective system architectures subject to a set of heterogeneous requirements (e.g., power flow, reliability, safety, workload, timing). We have evaluated the two techniques for solving exploration problems, numerically demonstrating their advantages and drawbacks. Overall, our experimental studies confirm that ARCHEX satisfies the following important criterias:

- *Usability for designers.* The pattern-based formal language allowed us to quickly write down the complex design specifications of the investigated systems with a set of short expressions. The resulting input files were easy to understand and update, while their size (less than hundred rows of code) was several orders of magnitude smaller than generated MILP formulations.

- *Reusability and extensibility.* We reused the same requirement patterns (e.g., interconnection, reliability) across two different domains. The interface of patterns was extended to support both applications. Furthermore, using the underlying generic basis of the exploration problem (Chapter 2) we implemented several domain-specific patterns that allowed us to more easily express some particular concerns (e.g., operation mode requirements).
- *Expressiveness.* It was possible to capture a variety of design concerns, such as interconnection, balance, workload, timing, reliability. Moreover, the interface of existing patterns provided enough flexibility to accurately capture the system structure at a chosen level of abstraction (for example, it was possible to distinguish between left and right parts of the aircraft, AC and DC buses, high and low voltage levels).
- *Performance and Scalability.* Architectures of realistic size and level of complexity have been generated in reasonable time, typically on the order of minutes. Monolithic optimization is preferable from the viewpoint of solution quality and optimality, while for large-scale designs with heavy formulations of reliability constraints its performance may deteriorate. In this case users may opt for the iterative optimization technique, which runs much faster. With several solving algorithms at hand, ARCHEX is scalable.

The aforementioned features of design methodologies and tools are all essential for the development of large systems. In ARCHEX, many important concepts, both methodology- and implementation-related, can be reused across different domains. For example, new application requirements of existing and new categories can be encoded within the same generic basis, while the same codebase can be used to quickly implement them.

Our results also suggest several directions for future work, which would improve the accuracy of high-level design capture. While some of these suggestions refer to the current case studies, others would also be useful in a more generic sense:

- *Multiple functional flows.* Even though the theoretical background of our exploration methodology allows several functional flows to be present in the same architecture, with ARCHEX we are currently able to specify only a single one. A set $\{\mathcal{F}_1 \ldots \mathcal{F}_n\}$ of functional flows would significantly increase the flexibility of existing design patterns as well as the overall expressiveness. These flows may represent several subsystems, that are jointly optimized within the same exploration problem. For example, AC and DC loads of an EPN may have different components in the paths that connect them to generators (e.g., for DC loads a rectifier has to be used to convert the AC from engines). Such difference is not allowed within a single $\mathcal{F}$, since every component type in $\mathcal{F}$ must occur at least once in every path. Similarly, in an RMS different product types, even from the same product family, can have different manufacturing processes, i.e., some of them can either bypass some machines in $\mathcal{F}$ or have additional ones. Therefore, having several functional flows would allow to formulate exploration problems for a broader class of RMS.

- *Reconfiguration requirements.* In case of component failures, an EPN has to switch contactors and reconfigure the power distribution network to keep the critical loads powered. Similarly, an RMS has to redistribute production flows both in case of machine failures and changes in the product demand (i.e., operation mode). The actions taken by the systems during reconfiguration are also subject to constraints, such as timing. For example, to switch a load from a faulty power source to a healthy one, a sequence of contactors have to be opened and/or closed. Contactors cannot be switched simultaneously, which may cause paralleling of AC sources. Also, each switching requires some time. A timing constraint for such reconfiguration of the power system should impose that the overall time taken for switching a sequence of contactors, i.e., the time when a critical load is unpowered, does not exceed a threshold. Intuitively, only two contactor switches are required: disconnecting from an unhealthy source or bus and connecting to a stable one. However, in general, the task is more complex, because other components of the EPN also may have to be rerouted, e.g., to keep the power balanced. Therefore, one not only needs to enumerate existing paths, but also their possible combinations, to identify all reconfigurability scenarios. Similarly to enumeration of fault events [12], this has exponential complexity, and efficient approximation techniques need to be devised. Another possible direction is to explore iterative optimization techniques and new learning functions specifically designed to support such reconfiguration aspects.

- *Enhanced theory solvers.* To minimize the gap of quality of solutions obtained with monolithic and iterative optimizations schemes, the latter has to be refined by integrating more domain-specific knowledge in the theory solvers, i.e., analysis routines and learning functions. This is especially important for RMS, where we obtained several cases of over-design by using the "lazy" approach. Furthermore, the previous suggestion (reconfiguration requirements) is encouraging for developing new theory solvers and investigating iterative schemes *modulo* other concerns (not only reliability).

- *Using simulation in the loop.* A more careful investigation of system behavior with generated architecture is possible via simulations. The latter can be used both for optimizing some continuous-valued parameters [40] and for verification. For example, system faults can be injected to an EPN to evaluate its resiliency. Simulation tool can be used as a theory-solver in the iterative optimization routine (Algorithm 2), which we further elaborate in Section 7.2.1.

- *Control patterns.* Finally, as already mentioned in Section 5.3, an interesting extension for ARCHEX, both in general and within the scope of the presented case studies, is related to capturing the constraints related to the controller. Then, by extending the solving part of the framework with additional algorithms (e.g., based on receding horizon), ARCHEX can be made useful also for the controller design step, which is succeeding the architecture selection.

# 6 Optimized Topology Selection and Component Sizing for Wireless Networks

*In this chapter we apply our methodology to architecture exploration problems in the wireless networks domain to jointly select topology and component sizing. Among the application requirements formalized in Chapter 3, there are several dedicated groups of constraints, which are implemented in* ArchEx *as a special extension of the framework. We evaluate this extension on two case studies from wireless sensor networks: data collection and localization.* ArchEx *translates rich sets of network requirements (e.g., routing, link quality, energy) into mixed integer linear constraints over path variables. These variables are a part of the basis of our exploration problems and denote the presence or absence of paths between network nodes. We then apply the proposed algorithm for compact, yet approximate, path encoding to reduce by orders of magnitude the problem complexity, and use MILP to solve large-scale and otherwise impractical problem formulations. By varying the degree of approximation provided by the algorithm, we explore the tradeoff between optimality and runtime for small and large problem instances. Finally, we also report on the results of an extensive signal strength measurement campaign in indoor office scenarios, which allowed us to propose several improvements to conventional wireless channel models in order to increase their accuracy. These improvements are currently implemented in the* ArchEx *framework as a part of its wireless extension and library, while their use in a network simulator is planned as future work.*

## 6.1 Overview

The ubiquitous deployment of devices in today's Internet of Things (IoT) relies on wireless networks to guarantee functionality and connectivity. The same applies to the rapidly increasing number of networked cyber-physical systems, where physical plants and controller units exchange sensor readings and control commands over the wireless channel. The system and network design, however, is heavily influenced by decisions made in the early stages of the design, when their impact is still hard to foresee. A major bottleneck is the lack of comprehensive frameworks for scalable, multi-dimensional design space exploration under heterogeneous network requirements (e.g., routing, latency, lifetime). Wireless network designers are expected to simultaneously reason about many alternatives and have to take risky decisions based solely on their heuristic evaluations and accrued knowledge. Exploration by simulations and prototype testbeds is often time consuming and limited in the number of evaluated configurations. Moreover, the lack of guarantees of meeting the requirements may lead to unexpected failures, unaffordable redesign cycles and, in certain cases, safety violations. Therefore, methodologies and tools that enable efficient co-design and provide correctness guarantees on a set of system-level concerns are highly desirable.

Wireless sensor networks (WSN) are one of the central wireless communication technologies both in CPS [109] and IoT [10]. Autonomous, self-powered and wire-free devices (nodes) can be distributed over a broad space providing large amount of real-time data while operating for a long period of time (up to several years). The key features and connectivity of WSNs enable users and researchers to easily access the data and experiment with various configurations of the sensing infrastructure. Typical application of WSNs include, but are not limited to, building automation (HVAC, adaptive lighting, gas detection), factory automation (environment control, monitoring of machines and robotic devices), security (survelliance, intrusion detection), localization, tracking and others. Sensor networks can be deployed both in indoor and outdoor scenarios.

Design of wireless sensor networks is a vast field of research [113, 46]. Our methodology focuses on a set of high-level decisions, such as selecting network components from a library of communication devices, and network topology. The latter includes the *physical topology*, i.e., node placement in the deployment area, and the *logical topology*, which consists of a set of routes in the network. Even at this level WSNs are subject to a variety of general and application-specific design concerns, so that design space exploration can have plenty of objectives [56]. Many of them can be already (or potentially) captured by ARCHEX. Hereafter, we discuss only those types of requirements that are used in our design examples, while the reader is referred to [129, 46] for a broader classification.

**Case Studies**. We have selected two examples of wireless sensor network deployments to evaluate our methodology and toolbox. Both of them are related to building automation and are deployed indoors, for example, in an office environment. First one, *data collection network*, consists of end devices (sensors), which measure or detect some physical environment phenomena, one or more base stations, which collect and process sensor data, and routing devices, or *relays*, that forward the messages towards the base station. Additionally, collected data (e.g., readings of light, humidity or gas sensors) may be provided to a control algorithm that manages a set of actuators (e.g., window blinds, room lights, heaters). Second application is a *localization network*, in which sensor nodes either determine their locations relative to each other or are used for positioning and, possibly, tracking of a device (mobile or static). A variety of localization WSN-based systems have been proposed [81] with different architectures and localization techniques involved. In this work, we use one of the most typical scenarios for evaluation of our methodology, as detailed in the next section.

Both our applications have been inspired by POVOMON [20], an open-data IoT sensor network designed in response to the growing demand for intelligent sensing solutions for buildings automation and power grids management. POVOMON has been deployed on one of the floors of the Department of Information Engineering and Computer Science at the University of Trento, Italy. The network[1] covers a set of typical indoor office spaces (e.g., corridors, work and study rooms, chill areas) within an overall area of 40 × 40 m. It is currently a data collection network of 25 nodes that track various indoor conditions including temperature, light, humidity and vibration. In our design examples, we synthesize WSN topologies of a larger scale (e.g., with a sensor placed in every office room) in the same indoor area. Our goal is to show that the proposed methodology can be effectively used for the exploration of high-level WSN architectures by selecting and calibrating network devices and topology. Generated architectures provide a set of guarantees to designers, thus increasing their confidence during the real deployment phase.

**Requirements**. One of important high-level design concerns in wireless networks, WSN in particular, is the *node placement*. Having a set of devices, the designer has to decide where to deploy them so that the network correctly performs its function as well as meets non-functional requirements (e.g., power consumption). Limitations and constraints for node placement are imposed by the number of available nodes as well as environment characteristics and certain features of the deployment area. In particular, it is common for a network designer to refer to a civil engineer, who can highlight potential spots, or *candidate locations*, for placing the nodes (e.g., walls, ceilings). Therefore, node placement is typically a discrete choice, which can be encoded as a set of decision variables in the network optimization problem.

Another paramount design aspect is *routing*. For example, in a data collection network, every sensor must have at least one route to the base station in order to deliver the readings. In some networks, routes are assigned dynamically by a routing protocol depending on current conditions of links and nodes (e.g., noise, battery level, location), while a lot of deployments use static route assignments. Moreover, routing protocols may also select routes out of a set of predefined candidates. Our design framework focuses on static assignments, therefore, in the scope of routing requirements our goal is to define a valid logical topology, i.e., routing, that meets other high-level design concerns. These requirements are detailed in Section 3.2.6.

Wireless networks are also subject to *link quality* (LQ) constraints. Several metrics, such as the ones exemplified in Section 3.2.7 (RSS, SNR, BER, ETX) can be used in these constraints. The ability of a node to reliably deliver a message to a sink (i.e., without errors) directly depends on the LQ level of wireless links and routes. In particular, network *reliability requirements* are highly related to the LQ. That is, certain level of network reliability is achieved by using links with an acceptable LQ level. Additionally, network fault resiliency is increased by having several replicas for every route, which we also express as a routing constraint (see Section 3.2.6). In this work, the LQ naturally depends on the selected routes as well as on some device parameters, such as TX power and antenna gain. Device configurations in our framework are selected via mapping the nodes of the topology to a library of network components.

Nodes that compose a WSN are typically autonomous battery-powered devices, while replacing the batteries is often an expensive task, since the network can be large-scale or deployed in

[1] POVOMON website: http://povomon.disi.unitn.it.

a hazardous area. Therefore, *energy consumption requirements* represent a significant aspect in sensor networks, as well as in other types of wireless communication. As a rule of thumb, WSN nodes must have a lower bound on their *lifetime*, i.e., the time until their battery is depleted so that they cannot perform a dedicated system function. Section 3.2.8 details the energy consumption constraints. In our formulation, the lifetime of nodes and the network depends on many design decisions, such as node placement, routing and the choice of the radio parameters, and can be jointly optimized with other objectives.

Finally, localization networks suggest a set of corresponding requirements, such as accuracy, precision, robustness and so on. We currently support a so-called *reachability constraint*, which prescribes that there must be a certain number of links between anchor nodes and a mobile target in a set of possible locations (evaluation points). These links must satisfy an LQ requirement. With these constraints, generated node placement ensures that localization is performed via a set of reliable links thus providing implicit guarantees on the quality, as further explained in Section 3.2.9.

## 6.2 Specification and Evaluation

All the numerical experiments reported in this chapter have been performed on an Intel Core i7 3.4-GHz processor with 8-GB RAM running Ubuntu 16.04.

### 6.2.1 Data Collection Network

As discussed in Section 4.3.2, the wireless extension of ARCHEX accepts a floor plan as an additional input for creating exploration problems. Therefore, as a first step we create an SVG file of the deployment area with dimensions 75 × 46 meters, which includes walls and doors, their thickness and material (e.g., glass, plasterboard, wood), and locations of WSN nodes. Other obstacles are neglected in this example. Node locations reflect the structure of the network template $\mathcal{T}$. The floor plan is shown on Figure 6.1a. There are 35 sensors (shown in green) located in rooms and corridors, and one base station (red). Their positions are fixed. The remaining nodes (cyan color) represent candidate locations for relays. The total number of nodes in the template $\mathcal{T}$ is 136.

We then create a library $\mathcal{L}$ with the following components: Sensor, Relay, and Sink. Each component is labeled with its cost **c**, processing delay $\tau$, TX power **tx**, antenna gain **g**, and current consumptions for the radio and other hardware components, as explained in Section 3.2.8. The characteristics correspond to those of real WSN hardware (e.g., transceivers and chips from [5]) and are summarized in Table 6.1. We select from only 3 distinct transceivers, while each of them can be configured to several different TX powers. Also, external antennas can be used. In the latter case, the cost of the device is increased.

In the specification, we declare the functional flow $\mathcal{F}$ = (Sensor, Relay, Sink), so that only the upstream traffic is allowed. The composition rules of $\mathcal{T}$ allow sensors and relays to commu-

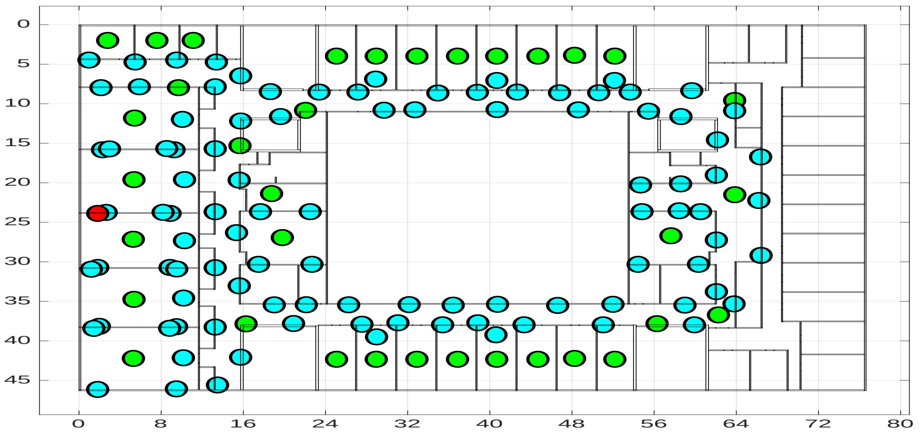
Table 6.1 – Summary of the WSN platform library $\mathcal{L}$: transceiver names, costs, processing delays, TX powers, antenna gains and current consumptions in TX, RX, idle (active) and sleep modes. Using an external antenna increases the cost for 5$.

| **Device** | **Cost** | **Delay** | **TX power** | **Gain** | **Currents** (mA) | | | |
|---|---|---|---|---|---|---|---|---|
| | ($) | (s) | (dBm) | (dBi) | $c^{TX}$ | $c^{RX}$ | $c^{active}$ | $c^{sleep}$ |
| CC2420 | 30, 35 | 0.8 | 0, -3, -7 | 0, 5 | 17.4, 15.2, 12.5 | 18.8 | 0.426 | 0.0021 |
| CC2520 | 40, 45 | 1.1 | 3, 0 ,-4, -7 | 0, 5 | 31.3, 25.8, 23.1, 20 | 18.5 | 1.6 | 0.0015 |
| CC2650 | 50, 55 | 0.7 | 0, 5 | 0, 5 | 6.1, 9.1 | 5.9 | 0.061 | 0.001 |

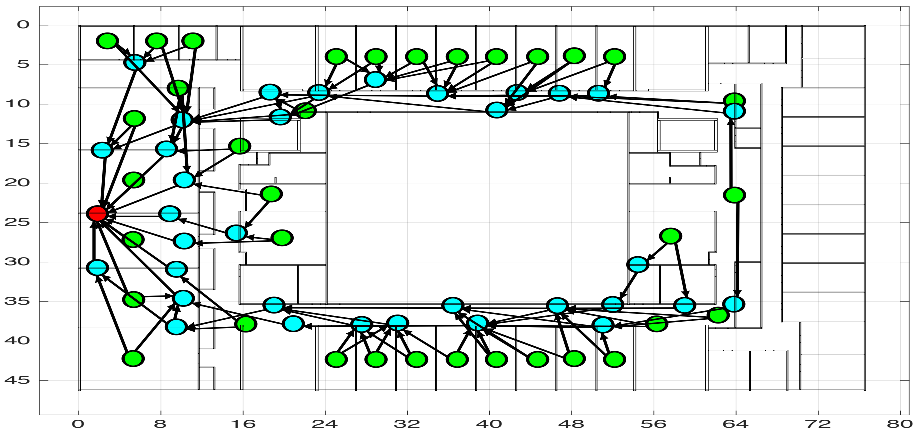
nicate to other relays or directly to sinks, with all other connections (e.g., between sensors) being restricted. The specification is further extended with the parameters of the environment, the protocol and the batteries. In particular, we assume the same noise level of -100 dBm for all links. Our network uses a TDMA protocol with a slot duration of 1 ms and 16 slots in a superframe, a packet length of 50 bytes. The bit rate for all links is 250 kbps. Sensors transmit a packet every 30 s and have zero cost. Finally, the power of WSN nodes is constrained by two 1.5 V AA batteries, each of 1500 mAh.

We then use routing, link quality, timing and energy consumption patterns to set up the requirements. Every sensor in a data collection network must have a route to the base station. We declare these routes with the pattern `p = has_path(A,B)`, where `p`, `A` and `B` are symbolic names for path, source and sink. To increase the network resiliency to faults, we add some redundancy. To do this, we require two disjoint routes for every sensor to the base station by using the pattern `disjoint_links(p1,p2)`. In total, we require 70 routes (2 for each of 35 sensors). For the link quality requirements, we consider the signal-to-noise (SNR) metric and the pattern `min_signal_to_noise` to set up a minimum SNR of 20 dB for every link. We instantiate a `max_latency_of_path(p,5,s)` pattern to set up a bound on the end-to-end delay of every path `p` to 5 seconds. Finally, with the `min_network_lifetime` pattern we require node batteries to last for at least 5 years. The whole specification created in ArchEx contains only 150 lines of code.

Table 6.2 shows the solver time and synthesis results obtained while optimizing for different objectives: dollar cost (final topology shown in Figure 6.1b), network energy consumption and an equally weighted combination. The selected components have different TX power, while some of them also have an external antenna to satisfy the LQ constraints. The result of minimizing for energy consumption is a network with a much higher dollar cost. Power consumption can be reduced by decreasing the TX power, while for certain links this may result in LQ constraints violation. This leads to the selection of more expensive low power components, for example, with smaller radio TX and RX currents or smaller MCU standby current, which highly affects the lifetime in the long run. The calibration of the software components (e.g., protocol parameters, such as slot duration, epoch, packet size) will be object of future work. The tradeoff between dollar cost and energy consumption can be explored when optimizing for a combination of objectives.



(a)



(b)

Figure 6.1 – (a) Template $\mathcal{T}$ of the data collection WSN (total of 136 nodes): sensors (green), sink (red) and candidate locations for relay nodes (cyan); (b) Generated topology of the data collection WSN optimized for dollar cost (only used nodes and links are shown).

For each experiment we obtain MILP formulations of around $1.5 \times 10^5$ constraints and $4.5 \times 10^4$ variables, which is orders of magnitude larger than the size of the specification written in ArchEx. The number of candidate paths $K^*$ generated by Algorithm 1 for every required connection was set to 20, which, based on the discussion in Section 6.2.3, provides a very cost-effective solution. We used link path loss values, precomputed by the multi-wall channel model, as weights for edges of the network graph. Exhaustive path enumeration led to problems with over $10^7$ constraints and $1.5 \times 10^6$ variables, which required several hours only for the encoding, to be contrasted with a few minutes in our approach; no solution was obtained before an 8 h timeout. Our path encoding algorithm reduces the problem size by two orders of magnitude. At the same time, it allows handling more complex problems than the ones in the literature [106, 108], in both architecture size and dimensionality of the design space.

Table 6.2 – Final number of nodes, dollar cost, average node lifetime and solver time for a data collection WSN optimized for different objectives.

| Objective | # Nodes | $ cost | Lifetime (y) | Time (s) |
|---|---|---|---|---|
| $ cost | 61 | 1022 | 7.33 | 45 |
| Energy | 63 | 1480 | 12.24 | 260 |
| $ + Energy | 61 | 1241 | 9.69 | 66 |

Table 6.3 – Final number of nodes, dollar cost, average number of reachable anchors by the mobile node, and solver time for a localization network optimized for different objectives.

| Objective | # Nodes | $ cost | Reachable | Time (s) |
|---|---|---|---|---|
| $ cost | 28 | 1050 | 3.1 | 115 |
| DSOD | 24 | 1310 | 3.6 | 121 |
| $ + DSOD | 24 | 1180 | 3.03 | 144 |

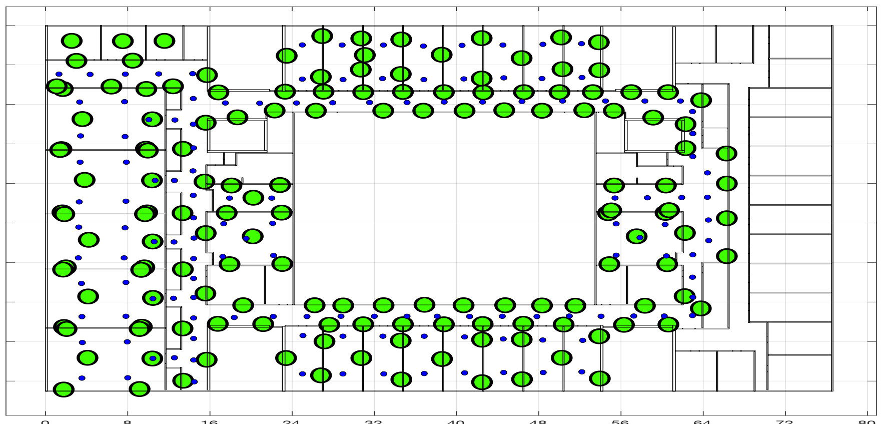
### 6.2.2 Localization Network

For this case study, we consider a range-based localization system that uses trilateration to calculate the 2D position of a mobile node (target) using distance measurements from fixed nodes with known locations (anchors). We assume that anchor nodes communicate only to the target and not to each other. The mobile node is responsible for computing its position based on the signals received from the anchors. Therefore, instead of requiring certain routes in the network topology, we need to ensure a set of local point-to-point connections between anchors and possible locations of the target, i.e., the network has a star topology. The goal of the exploration problem is to select anchor nodes from the library of components and to determine their best placement based on localization constraints specified below.

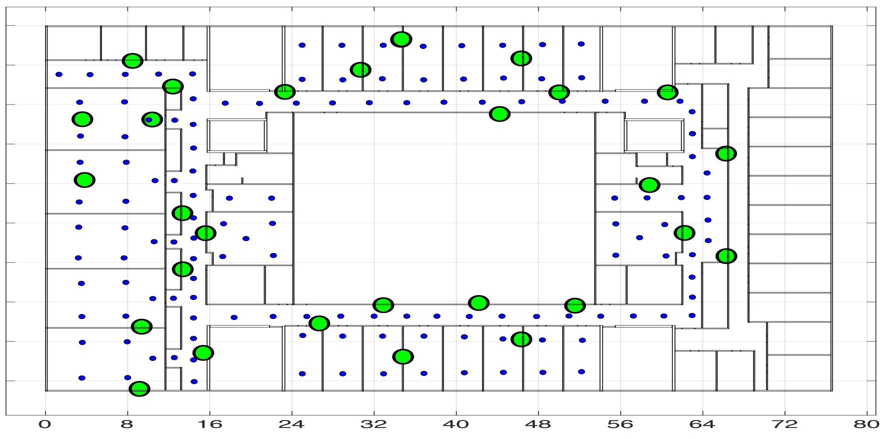
We specify 150 candidate node positions and 135 evaluation (mobile node) locations for the same building floor, as shown on Figure 6.2a. The `min_reachable_devices` pattern implements the localization constraints (4a)-(4b), and we apply it to require that, at every test point, the mobile node must be able to receive signals from at least 3 distinct anchors. Furthermore, with the same pattern, we request that only reliable links with a minimum RSS of -80dBm must be selected. This also contributes to decreasing the ranging error, which rapidly grows for larger path losses and unstable signals [64].

We solve the problem for two different cost functions: dollar cost (result in Figure 6.2b) and difference of sum of distances (DSOD) between network nodes and test points. The latter was proposed in [111] as a linear version of the Cramer Rao lower bound - a metric used in the accuracy evaluation of localization systems. To set up the reachability matrix **r** and localization constraints, we use Algorithm 1 with $K^* = 20$ candidate anchors for every test point.

Results in Table 6.3 show that optimizing for the DSOD objective produces a placement with smaller number of more expensive nodes equipped with antennas, i.e., their signal can reach more test locations. This system also has smaller power consumption. In all experiments, the number of variables and constraints counts up to, respectively, $3 \times 10^4$ and $3.5 \times 10^4$. A full enumeration of all test points reachable by all anchors would lead to several millions variables and constraints, thus making design exploration intractable. This again proves the effectiveness of the approximation.



(a)



(b)

Figure 6.2 – (a) Template $\mathcal{T}$ of the localization WSN: candidate locations of anchor nodes (green nodes, total of 150) and evaluation points (blue dots, total of 135); (b) Generated node placement for the localization WSN optimized for dollar cost (28 anchor nodes total).

## 6.2.3 Scalability and Optimality

We test the scalability of our techniques on data collection network architectures (same as in Section 6.2.1) with an increasing number of nodes (total) and end devices (sensors) in the template $\mathcal{T}$, and with $K^* = 10$, as reported in Table 6.4. The significant reduction in problem complexity and execution time shows the advantage of using the approximate encoding of network paths in Algorithm 1. We also provide the measured (for larger instances - estimated) number of constraints for the case of full enumeration of paths, which are several orders of magnitude larger. Execution times of the order of days are expected to solve these large problem instances, since only a few smaller networks were synthesized within an 8 h timeout.

The effect of selecting different values of $K^*$ is demonstrated in Table 6.5. Only optimizations for dollar cost are shown, while very similar trends have been also observed for other objectives (e.g., energy consumption). Comparison with the optimal solution obtained without approximation is only possible for the small WSN template, since exhaustive exploration becomes soon intractable. Increasing $K^*$ leads to higher quality results in terms of cost. In general, when $K^* \to \infty$ all possible paths are enumerated, leading to the global optimum. However, large values of $K^*$ result in a nonlinear growth of execution time, while the rate of improvement in the cost function decreases. In our experiments, a small decrement in cost for $K^* > 10$ comes at a very large price in terms of performance. On the other hand, when $K^* = 1$ only one candidate path is proposed for every required route, i.e., the routing is fixed. In this case, the performance is comparable to the one of heuristic algorithms in the literature [108]. Yet, even if a single candidate is generated for every required route, our approach additionally guarantees optimal component sizing for the selected topology. Moreover, slightly increasing $K^*$ significantly improves the cost function w.r.t. $K^* = 1$ without much timing overhead.

In general, the value of $K^*$ depends on the specific problem at hand (e.g., the types and number of constraints determining the feasible space of the optimization problem as well as the cost function). A reasonable method for deciding $K^*$ would be to setup a systematic search for a given problem by generating multiple topologies for different values of $K^*$ and stopping once the execution time becomes higher than a predefined threshold or there are no further cost improvements. As $K^*$ increases, we obtain solutions that are at least as good as, if not better than, the previous ones. The best generated topology can then be returned as a final result. Apart from incrementally increasing $K^*$, smart search strategies can also be applied, e.g., binary search.

Furthermore, we can provide a few guidelines that we empirically identified while validating our approach on the reported case studies. For example, for network sizes in the range of our examples, a reasonable recommendation would be to select a value of $K^*$ between 3 and 10, since values outside of this interval provided marginal advantages in terms of cost versus execution time. For smaller networks we can safely use a large K* to achieve better solutions without compromising the execution time. We can also use a larger K* for networks made up of well-identifiable clusters of nodes. In this case, increasing K* tends to generate new path candidates that have the same links (edges) but in a different order. Therefore, fewer new variables are introduced with each new path.

Table 6.4 – Number of constraints and solver time for different network architecture sizes generated by using the approximate path encoding algorithm ($K^* = 10$) compared to full enumeration of paths ("TO" means that no solution has been found within an 8 h timeout).

| **#Nodes** (total in $\mathcal{T}$) | **#End devices** (to be routed) | **#Constraints**, $\times 10^3$ (full / approximate) | **Time (s)** (full / approximate) |
|---|---|---|---|
| 50 | 20 | 862 / 24 | 8233 / 12 |
| 100 | 20 | 1743 / 54 | TO / 28 |
| 100 | 50 | ~ 3800 / 125 | TO / 55 |
| 100 | 75 | ~ 4800 / 150 | TO / 93 |
| 250 | 50 | ~ 3500 / 108 | TO / 340 |
| 250 | 100 | ~ 5700 / 175 | TO / 1175 |
| 250 | 200 | ~ 10000 / 310 | TO / 1708 |
| 500 | 50 | ~ 7400 / 230 | TO / 818 |
| 500 | 100 | ~ 11000 / 346 | TO / 5330 |
| 500 | 200 | ~ 21000 / 655 | TO / 8354 |

Table 6.5 – Costs and solver times for data collection networks with a small template $\mathcal{T}_1$ (20 end devices, 50 nodes total) and a larger template $\mathcal{T}_2$ (200 end devices, 250 nodes total) synthesized using different values of $K^*$, compared with the optimal solution (only for $\mathcal{T}_1$).

| **Template** | **Result** | $\mathbf{K^* = 1}$ | $\mathbf{K^* = 3}$ | $\mathbf{K^* = 5}$ | $\mathbf{K^* = 10}$ | $\mathbf{K^* = 20}$ | **optimal** |
|---|---|---|---|---|---|---|---|
| $\mathcal{T}_1$ | Cost (\$) | 920 | 861 | 805 | 642 | 619 | 579 |
| | Time (s) | 3 | 7 | 10 | 12 | 442 | 8233 |
| $\mathcal{T}_2$ | Cost (\$) | 2594 | 2280 | 2083 | 1909 | 1842 | - |
| | Time (s) | 8 | 85 | 358 | 1708 | 15334 | TO |

## 6.3 Improving the Wireless Channel Model

### 6.3.1 Motivation

One important consideration in wireless network design tools is their possibility of capturing the properties of the channel model. Estimated path loss of signals is used in many design decisions at different levels, including topology selection. Real signal path loss is very non-stationary and subject to a large number of random effects of the channel, such as reflections, shadow fading, multipath progagation and interference. Conventional channel models provide a reasonable approximation for applications in which a simplified physical layer representation does not severely compromise the overall outcome [125]. However, for several emerging systems, such as localization networks, these models become a bottleneck for achieving adequate results.

Undoubtfully, due to random factors affecting the signal, none of the channel models can capture all the properties of a particular area. Therefore, real-world prototyping and calibration of a system is an important and unavoidable step in the wireless network design flow. However, many decisions, e.g., hardware selection and routing, have to be made on earlier stages. In fact, high-level design spaces, in particular for WSN, can be huge. Their exploration can quickly become complex, time-consuming and overall unaffordable. Therefore, to estimate how the network would behave in a real environment, many designers rely on simulation-based design space exploration. The latter allows them to evaluate more configurations in shorter time, compared to prototyping.

Simulation models might not capture all possible aspects of a real scenario, however, they can still be very useful in obtaining best- or worst-case boundaries [85]. Same applies to our architecture exploration methodology. Many network requirements in our approach are expressed using path loss values, hence, their accurate estimation allows the tool to provide more precise guarantees on system properties (e.g., link quality, energy consumtpion). We note that, similarly to [108], ArchEx allows to integrate the site survey data in the formulation, i.e., to replace the channel model values with on-site signal strength measurements. However, in large-scale networks this may not be applicable due to a large number of required measurements (e.g., between every pair of candidate locations).

As mentioned in Section 4.3.2, ArchEx implements the multi-wall channel model, which is based on the classical log-distance model [110] and takes obstacles into account (see Expressions (4.3) and (4.4)). In the following sections, we summarize our efforts on improving the accuracy of the log-distance model, recapped below:

$$PL = PL(d_0) + 10\eta \log(d/d_0) + X_\sigma,$$

where $d$ is the distance between TX and RX antennas, $PL(d_0)$ is the path loss at a reference distance $d_0$, which is typically one meter for indoor WSNs [42], $\eta$ is the path loss exponent (PLE), which is determined either empirically or from the literature, and $X_\sigma$ is a zero-mean Gaussian random variable with standard deviation $\sigma$.

Our study is based on an extensive set of RSS measurements collected within different typical indoor spaces and node placement scenarios for wireless sensor networks [64]. We compared the results with corresponding values from the log-distance model, which allowed us to reveal interesting regularities within the measured data. Based on our observations, we propose several improvements for the existing model, which we have implemented in ArchEx, while their integration would be also beneficial for network simulators and other design and optimization tools for wireless networks.

### 6.3.2 Received Signal Strength Measurements in Indoor Environments

In all experiments we considered a WSN operating in the ISM band (2.4 GHz). The measurements were performed in several realistic indoor spaces, moreover, different node placement scenarios and antenna polarization were involved. We performed two groups of experiments: baseline measurements and sensor node measurements. The former was performed by transmitting an un-modulated carrier (sinusoid) and using a spectrum analyzer for measuring the

RSS with an intention to provide a baseline for WSN experiments. For brevity, we completely omit the corresponding results and refer the reader to [64]. The second and primary group of RSS measurements was collected using two off-the-shelf WSN nodes, TX and RX. We made several experiment datasets in each indoor space with different mutual placement of nodes. For each scenario we varied the distance between the nodes with a step of 1 [m] and collected the RSS data for 5-10 minutes. With a 5 [ms] average sampling period, this resulted in roughly 120000 values in each dataset.

**Equipment**.WSN measurements were taken with the Z1 off-the-shelf platform Zolertia with an MSP430 MCU and a CC2420 low-power radio. Both TX and RX nodes were enclosed in a plastic box, powered by two AA batteries and equipped with a 5 dBi external RP-SMA antenna. All nodes run TinyOS 2.1.2 with our testbed application. A gateway node is connected to a laptop and forwards commands to the TX/RX nodes. When the TX node receives the “start” command from the gateway, it starts sending small packets (the payload includes only a 1-byte sequence number) to the receiver. The RX node processes each packet and stores the RSS value in the log. After each experiment, logs are downloaded from the receiver via a micro-USB cable. For communication with the flash, we used the components from Trident, an open-source software for in-field connectivity assessment for WSN [57]. For all experiments, TX power was set to 0 dBm (1 mW). Each experiment duration was set to 10 minutes.

**Measurement Scenarios**. All measurements have been performed in the building “Polo Ferrari” of the Department of Information Engineering and Computer Science at the University of Trento, Italy (partly, in the same area considered in the case studies in this chapter). Following spaces were considered:

- A corridor (in the following, *corridor1*), 48 x 2.8 x 2.6 [$m^3$], first wall - glass, second wall - gypsum plasterboard with many adjoined offices.
- Another corridor (in the following, *corridor2*), 56 x 2.42 x 2.5 [$m^3$], both walls made from gypsum plasterboard, with adjoined offices on both sides.
- A *hall*, 19.4 x 9.8 x 2.4 [$m^3$], side brick walls, front and back glass walls.
- A big *office room* (11.5 x 7.5 x 3 [$m^3$]), one glass wall, 3 other walls - gypsum plasterboard. The room is furnished with a lot of working desks, chairs, PCs.

Our measurement scenarios aim to cover typical alternatives of WSN nodes mutual placement. We are interested in finding out regularities or anomalies in the RSS behavior across these scenarios (they are illustrated on Figure 6.3):

1. *“Single wall”*. In this scenario, both TX and RX nodes were placed on the same wall at the height of 2.25 [m] (we ran some calibration tests beforehand, placing the nodes on the bottom, middle and top of the wall and observed the strongest and most stable link at the top).
2. *“Middle”*. Both nodes were placed in the middle of the space under study at the height of 0.8 [m] (such height is very convenient for real case studies, like a bodyworn node, a PDA in a hand or a robotic device).

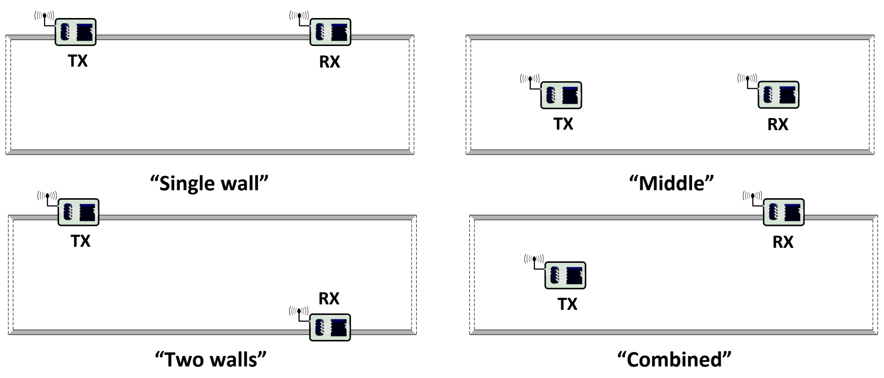


Figure 6.3 – WSN node mutual placement scenarios (exemplified in a corridor).

3. *"Two walls"*. This scenario was run only in the corridors. Nodes were located on reciprocal walls (height - 2.25 [m]). This placement scenario is typical of a WSN deployment in a corridor.
4. *"Combined"*. Was run only in the corridors. The RX node was placed on the wall (height - 2.25 [m]) and the TX node was located in the middle (height - 0.8 [m]).

Several factors are common for all scenarios. We varied the distance with a 1 [m] step to obtain distinct experimental datasets. This step has been chosen considering a good trade-off between the position updates of an object (e.g., a human) moving in a realistic scenario and the reasonable total time for collecting the measurements. The interval that we considered in our statistical investigation was [1 m, 15 m] in most cases (in some of them it was different due to space/scenario limitations). Second, we used the RSS from the minimum distance as the $PL(d_0)$ in calculations. Third, TX and RX antennas were in co-polarization condition. Finally, no mobile nodes were present in our scenarios and the LOS condition was assumed.

**Results**. Our results come from analyzing 16 distinct groups of data. Four are the baseline measurements, the rest are experimental datasets obtained with sensor nodes. Each group consists of separate measurement sets related to a particular distance between TX and RX. Each of the latter has roughly 60000 or 120000 values observed over a 5 min or 10 min period, respectively. TX and RX antennas were placed with the same polarization and their gain effects are removed from the data.

As part of the analysis, we compare the empirical results from different scenarios with corresponding analytical values, which could be provided by a log-distance model in a WSN simulator. The latter means that channel parameters in the tool, i.e., $PL(d_0)$ and $\eta$, are configured with limited knowledge of the real channel under study. That is, the designer selects them partly or totally relying on best practices, because doing a channel characterization for deriving them empirically could be complex and time-consuming. As we demonstrate with our measurements and analysis, these best-practice parameter values in fact can be very different from those estimated from the data. Even if $PL(d_0)$ is taken from measurements (it can be done easily), different values of the PLE result in considerable difference in the curves, which entails incorrect results provided by the channel model.

For each comparison we use two curves calculated with formula (4.3): analytical and empirical. For the former one we select the values of the path loss exponent $\eta$ from the literature. Typical value of $\eta$ for indoor free space is 2, while it can be smaller for corridors (down to 1.5) and higher for furnished rooms (up to 3) when the LOS condition is assumed [42]. In industrial environments $\eta$ could be bigger (up to 5-6) [110]. In this work we select analytical $\eta$ to be 1.6 for the corridors, 2 for the hall and 3 for the office room. For the empirical curve we estimate $\eta$ from our measurements. For doing so we calculate the linear regression for each dataset (RSS vs distance) and use its slope as an approximation for $\eta$. The value of $PL(d_0)$ for both curves is estimated from the data. For the "Combined" and "Two walls" scenarios we used 2 [m] and 3 [m], respectively, as the reference distance $d_0$. For all other cases 1 [m] was used. The mean RSS value from corresponding datasets was used as $PL(d_0)$. The random component $X_\sigma$ of the model was set to zero to verify later on, which distributions describe the deviations of real RSS values from the log-distance curve in a best way.

For the sake of brevity, we omit the plots that show the measured RSS vs distance. Instead, we show the empirical log-distance curves with parameters estimated from the collected data, and remark that they represent the decay trend well. The data itself is quite random, as expected. Our primary goal is to explore the differences between the curve drawn from the data and the one typically provided by a channel model in a simulator within a particular scenario. Random factors will be represented by a distribution, which is studied in the following section. However, if the random component is calculated around the wrong curve, the simulation outcome might be far from reality.

By comparing empirical and analytical curves for WSN a difference of 5-10 [dBm] can be observed (Figures 6.4 a-f). In most cases, the empirical curve is higher (i.e., the path loss is smaller). One exception is the "Two walls" scenario (Figure 6.4f), where measured path loss, conversely, is higher than the one predicted by the analytical model. Comparison of scenarios reveals very high similarity in empirical PLE values for "Single wall", "Combined" and "Two walls" scenarios in different spaces, i.e., the curves have similar slopes (Figures 6.5 a-b), while for the "Middle" scenario path loss from the office room has behavior different from other spaces (Figure 6.5c). The office room is furnished and also WSN nodes in this scenario were placed at a lower height compared with others. Hence, the occurrence of reflections and scattering has a significant impact on the path loss.

Despite being highly similar within the same node placement scenario, empirical values of the PLE are, nevertheless, considerably dissimilar across different scenarios. This is a very important observation because it suggests that a single value of the PLE cannot accurately describe the path loss between every TX and RX within the same space if their mutual placement is different. Currently only one PLE value can be set for the whole simulation in the tools that implement the log-distance model. This might cause significant inaccuracies.

Also, one can observe from Figures 6.5 (a-c) the difference in the path loss within the same

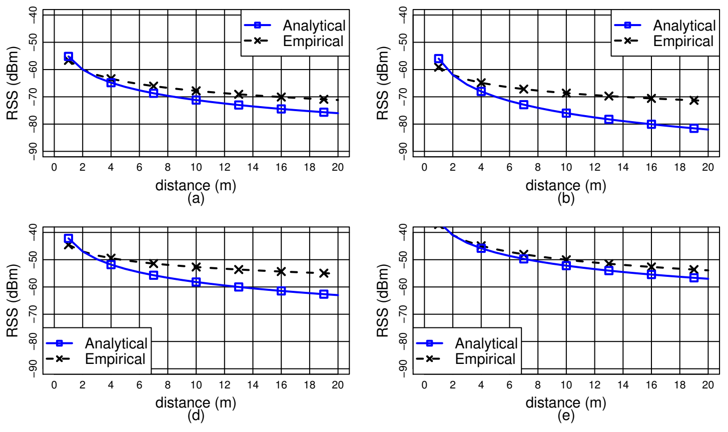


Figure 6.4 – Comparing analytical and empirical log-distance curves for WSN data. (a) Corridor1, "Single wall" scenario (b) Hall, "Single wall" scenario (c) Corridor1, "Combined" scenario (d) Corridor2, "Middle" scenario (e) Corridor1, "Middle" scenario (f) Corridor2, "Two walls" scenario.

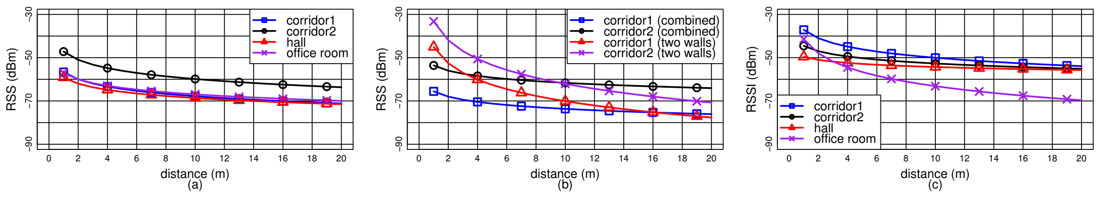


Figure 6.5 – Comparison of log-distance empirical curves for WSN data. (a) "Single wall" scenario (b) "Combined" and "Two walls" (c) "Middle".

placement scenario in different spaces. As the PLE values are similar, this is due to the varying $PL(d_0)$ parameter. For instance, on Figure 6.5a corridor2 has higher values, probably, due to the wave-guiding. On Figure 6.5b the path loss in corridor2 is smaller than in corridor1. This is likely related to different materials of these spaces and, therefore, different electromagnetic behavior of the signal.

### 6.3.3 Statistical Characterization of the 2.4 GHz Radio Channel

We also study the deviations of measured RSS from the root mean square (RMS) values computed for corresponding measurement groups, at different TX-RX distances. In reality, these deviations occur due to random fading effects such as shadowing. This would allow us to verify if all deviations of the RSS within different scenarios, spaces and distances can be adequately modeled by the same distribution. This is the way of implementing the log-distance model in WSN simulators: all random effects are considered by adding a random variable $X_\sigma$, which is a standard normal distribution. The value of $\sigma$ is the same for all signal evaluations within a simulation.

Fitting distributions is performed using the maximum likelihood (ML) method. We selected

Table 6.6 – MLE for data agglomerates across all measurement scenarios

| Distance | Distribution | $\Delta AIC$ |
|---|---|---|
| 1m | Normal ($\mu = 1.011$, $\sigma = 0.142$) | 7450 |
| 2m | Lognormal ($\mu = 0.0025$, $\sigma = 0.259$) | 14757 |
| 3m | Gamma ($a = 27.960$, $b = 27.425$) | 5337 |
| 4m | Lognormal ($\mu = 0.0012$, $\sigma = 0.187$) | 24025 |
| 5m | Lognormal ($\mu = 0.0020$, $\sigma = 0.238$) | 53088 |
| 6m | Lognormal ($\mu = 0.0027$, $\sigma = 0.284$) | 12483 |
| 7m | Lognormal ($\mu = 0.0018$, $\sigma = 0.234$) | 80391 |
| 8m | Lognormal ($\mu = 0.0040$, $\sigma = 0.349$) | 647 |
| 9m | Lognormal ($\mu = 0.0056$, $\sigma = 0.416$) | 40594 |
| 10m | Gamma ($a = 11.802$, $b = 11.273$) | 13355 |
| 11m | Gamma ($a = 16.601$, $b = 16.070$) | 19764 |
| 12m | Lognormal ($\mu = 0.0044$, $\sigma = 0.370$) | 58543 |
| 13m | Lognormal ($\mu = 0.0019$, $\sigma = 0.243$) | 16629 |
| 14m | Nakagami-m ($m = 7.474$, $\omega = 0.187$) | 1864 |
| 15m | Lognormal ($\mu = 0.0019$, $\sigma = 0.255$) | 64069 |

Normal and Lognormal: $\mu$ - mean, $\sigma$ - standard deviation. Gamma: $a$ - shape, $b$ - rate. Nakagami-m: $m$ - shape, $\omega$ - scale.

six distributions for our analysis, following the similar processing flow presented by Smith et al. [120] for body area networks. They are Normal, Log- normal, Gamma, Weibull, Nakagami-m and Rayleigh. We use the Akaike information criterion (AIC) [6] to compare the fitted distributions. This criterion allows finding a model with the minimum information loss among those that are considered. We consider AIC a relevant metric for our study, because we are interested not only in accurate modeling of distribution tails (i.e., high attenuation region), but also in the values around the mean.

For comparision, we created agglomerated datasets by joining the RMS-normalized values for each scenario (for example, 4 datasets for the “single wall” scenario were joined into one for all TX-RX distances) and, similarly, for each indoor space (e.g., joined the data from all placement scenarios for “corridor1”). For brevity, we omit most numerical results and report only the total agglomerate of all data across all studied indoor spaces and scenarios in Table 6.6. From the results we observe that RSS deviations at different TX-RX distances are best described by several distributions. While the Lognormal distribution is the dominating one, it has different parameters for different distances. Moreover, other distributions (e.g., Gamma, Normal) are also present. On the whole, our results support the hypothesis that a single distribution is not able to describe random RSS deviations well, and therefore, this might be a potential loss of accuracy for the log-distance path loss model. We refer the reader to [64] for an extended discussion.

### 6.3.4 Proposed Improvements for the Channel Model

On the basis of our results we propose the following improvements to the log-distance channel model in WSN simulators. First, different values of the path loss exponent can be used for different placement scenarios. During a simulation one could determine the mutual placement of nodes, for which the path loss is evaluated, and use the corresponding PLE. In particular, we noticed that the PLE for the "Two walls" scenario can be 2-3 times higher than for others. Therefore, it requires a separate PLE value to provide accurate results. Values for other placement scenarios (in this work we tried only 4 most typical) can also be configurable.

Second proposed improvement is related to modeling random factors using a certain distribution. We have shown that RSS deviations are best described by different distributions and parameters at different distances (at the same time having notable similarities within different spaces and placement scenarios). During the evaluation of a particular path loss, one could check the distance between TX and RX and use a distribution from a corresponding distance interval instead of using the same standard normal distribution in every case. Another approach is to keep the existing $X_\sigma$ random variable but allow the distribution parameters (mean $\mu$ and standard deviation $\sigma$) to be generated every time with their corresponding distributions. For example, for scenarios studied in this work $\mu$ and $\sigma$ can be generated with Lognormal and Rayleigh distributions, respectively.

Although in this work we considered only accessible office environments without mobility, the proposed analysis methodology and model improvements could be applicable to other environments, such as industrial, characterized by high noise level and presence of various mechanical obstacles. This would require additional measurements, but would highly contribute to achieving more accurate results in using a WSN simulator during the design. Overall, this study can be beneficial for improving the channel models (and using them to simulate the designed system in different environments and conditions) as well as for configuring a particular system within a particular environment.

We have currently implemented the proposed improvements in ArchEx for the log-distance and multi-wall models, which are both parts of our wireless library. In particular, while reading the SVG file with the information about the deployment area and the candidate locations of nodes, the tool is able to analyze each pair of nodes and determine their mutual placement. It then classifies these placements according to a set of scenarios, such as the ones presented here (e.g., single wall, reciprocal walls) and assign a corresponding PLE value to the link, which is then used for computing the path loss of the latter. Also, the distance between the nodes is computed and used to select a probability distribution and its parameters for each particular case in order to more accurately capture the random part of the log-distance model. The distributions are currently selected according to the numerical results from our measurement campaign [64], but can also be extended with datasets obtained from other environments, e.g., industrial. One of our future work directions is related to using the latter in a loop with simulation for a more precise evaluation of network behavior, similarly to [87]. Corresponding simulator could benefit from the improved channel model as well.

## 6.4 Related Work

The problem of optimizing device placement and connectivity in wireless networks is well studied [130]. State-of-the-art approaches include simulated annealing [24], genetic algorithms [61, 52], tabu search [111], nonlinear optimization [26, 111] and MILP-based techniques [106, 108, 7, 38]. With respect to these approaches, our mapping constraints can capture a richer set of requirements and allow for *component sizing* in addition to topology selection. Many parameters that were fixed in previous formulations (e.g., transmit power, antenna gain, current consumption) can now be selected based on a library of components. The user can also specify the types of components and their communication rules, thus handling a broader category of designs.

The goal of the MILP optimization problem presented by Amaldi et al. [7] is to jointly select the node placement and the channel assignment, while minimizing the installaton cost and satisfying the connectivity requirements. Multiple radio interfaces, multiple frequency channels and interference are taken into account. The authors also propose two polynomial heuristic algorithms, which allow them to tackle larger problem instances (over 50 nodes), while achieving up to 5x faster execution times compared to solving monolithic MILP problems. In particular, one of the algorithms suggests to solve a smaller MILP problem, without link capacity constraints, in a loop with feasibility checking routine and learning new routing constraints. In other words, their approach is similar to our iterative optimization technique. Differently from [7] we account for link quality metrics (e.g., RSS, BER) and energy consumption of the nodes. At the same time we neglect, in particular, channel assignment. Yet, these constraints can be easily incorporated in our formulation.

Pinto et al. [106] proposed a generic MILP formulation for synthesizing wireless network architectures for building automation and control. The goal is the minimization of the cost of network nodes under a set of topological, link utilization and link quality (end-to-end delay, packet error rate) constraints. More recently, Puggelli et al. [108] extended the formulation from [106] with different routing patterns (e.g., unicast, multicast) and power consumption constraints for indoor wireless sensor networks. They explore a broader design space and solve a more complex optimization problem. With respect to [106, 108], our approach is more general, i.e., it not only supports the formulations these previous works, but also other concerns (e.g., lifetime constraints, localization constraints). Overall, our approach is superior both in network size and the dimensionality of the design space, which now includes the sizing of network components (e.g., choosing the transmission power of devices, selecting external antennas and so on).

This work differs from efforts aiming at polynomial-time approximate algorithms to solve the NP-hard exploration problem [108, 49, 58], since it rather focuses on more compact approximate encodings that can still leverage the empirical advances of state-of-the-art MILP solvers. For example, according to the results provided by Pugelli et al. [108], their MILP formulation becomes impractical, i.e., no solution is found within a timeout, for networks of more than 50 end devices. Therefore, they propose a polynomial-time heuristic algorithm in place of the NP-hard MILP problem that applies Dijkstra's shortest path routine for synthesis of large networks that cannot be handled by exact MILP formulations. Instead, in our work we use Yen's algorithm [128], which is a generalization of Dijkstra's algorithm, to *symbolically* generate compact MILP formulations of network path constraints that can scale to hundreds of nodes.

Some works solve the synthesis problem using techniques different from MILP. For example, the Wi-Design tool [83] uses agent-based optimization algorithm for finding optimal positions of WSN routers and sinks. Candidate locations for these devices are not specified by the designer as in most of other works, but estimated using a variation of a self-growing natural gas algorithm. In [52] a genetic algorithm is used for WSN placement and topology planning. In contrast to other problems, the objective is to find optimal locations for sensors and routers, while only the sink position is fixed. The optimization is in a loop with the WSNet simulator [25], which verifies the network characteristics, e.g., latency, packet drop rate and network lifetime. With respect to our approach, both [83, 52] are able to conduct a more careful and precise search of the best node placement (any location can be chosen as opposed to a discrete selection among candidate locations). Such exhaustive search, however, can become extremely time-consuming for large networks, thus making corresponding design problems intractable.

A recent MILP-based approach has been proposed for generating cost-effective hardware configurations of IoT devices [45]. The authors propose an optimization problem formulation for selecting components of the hardware platform that are compatible, e.g., use the same interfaces (UART, analog) and have sufficient number of pins. Hardware selection is one step in the proposed holistic system for rapid development of IoT applications, which also includes automatic software generation. Overall, this system operates on a level of single IoT devices, but not on the network level. Our methodology can be applied for solving hardware configuration problems, yet it also captures a higher, network-wide level of abstraction and allows to express non-functional properties (e.g., timing, reliability).

Satisfiability modulo theory (SMT) based encodings have also been recently proposed to find feasible solutions for wireless network scheduling problems [33]. Our work is different since it targets optimal solutions. While SMT-based techniques have been recently investigated for solving optimization problems [74], their scalability is, however, often limited for problems with a large number of real constraints.

Finally, our approach is complementary and can be combined with simulation-based design exploration [30, 124], as it provides system-level correctness guarantees that can be used to reduce the number of simulations needed for the exploration. A similar idea is leveraged by Moin et al. [87] for the optimized design of a human intranet network. In their approach, a mixed integer linear program generates candidate network architectures under coarse energy constraints, which are then checked by a discrete-event network simulator under reliability constraints. Simulation results generate lower bounds on the power consumption, which are added to the MILP to prune the search space and achieve faster convergence. Our methodology can be applied on the optimization step of [87] and provide more expressiveness and generality to the existing formulation with additional types of constraints and sizing of network components.

## 6.5 Conclusions

Our results demonstrate the applicability of our optimized mapping approach to large-scale wireless network designs, which are a crucial component of today's IoT applications. Enhancing the ArchEx framework with additional functionalities, such as channel models and floor plan support, allowed us to extend the list of application requirements with the ones typical for the wireless networking domain, and to use the methodology for corresponding design problems. In particular, we were able to synthesize cost-effective topologies and select the components of a data collection wireless sensor network under link quality, timing and lifetime constraints. Moreover, the reliability of the topology was also considered by adding redundancy to every network path using routing constraints. In the second case study, we generated node placements and chose the components of a localization network, while ensuring that the mobile target is reachable by a minimum of 3 anchors in a set of its possible locations.

Most of the requirements were captured using the path variables that belong to the generic basis of our exploration problem formulations. Indeed, the presence or absence of a node or an edge in a particular network route is an important information for computing the end-to-end delay of a route, energy consumption of a node and other properties. However, general encoding of network paths requires their full enumeration. For the network sizes in the range of our examples, such enumeration leads to heavy and impractical problem formulations with millions of MILP constraints.

Our methodology provides a *scalable* solution for topology synthesis problems. To decrease their complexity, we applied the approximate encoding of network paths, as proposed in Algorithm 1. As a result, we obtained more compact (several orders of magnitude smaller) MILP formulations and were able to solve large problems in reasonable time. A small number of "best" path candidates proposed by the algorithm efficiently guides the solver to the most promising portion of the design space. Moreover, the size of this space and the tradeoff between optimality and execution speed can be controlled by tuning the value of the $K^*$ parameter. In our experiments, even a small number of proposed candidate paths (e.g., $K^* = 3$) led to solutions that are more cost-effective than greedy heuristic solutions for topology synthesis. Also, the solver guarantees the optimality of the proposed solution in the reduced design space. Larger values of $K^*$ give more improvements in the quality of the solution, at a higher price in terms of execution time, while the latter also depends on the size of the network template. As discussed in Section 6.2.3, a systematic strategy can be applied to find the best $K^*$ for a given network considering the available timeframe of the project.

Being significantly smaller with respect to full enumeration of network paths, the size of the exploration problems reported in this chapter is still very large. For example, different formulations for the data collection network amount to tens or hundreds of thousands of constraints and variables. Manual handling of such formulations is extremely difficult, if not impossible. In this situation, the advantage of using the pattern-based language, provided by ArchEx, is again evident.

The assesment of properties and the statistical characterization of the 2.4 GHz radio channel, conducted based on an extensive measurement campaign in indoor office spaces, allowed us to identify interesting regularities in the behavior of wireless signals in these environments. We then proposed two improvements for the conventionally used log-distance channel model: adaptive assignment of the path loss exponent depending on the mutual placement of TX and RX nodes, and using different probability distributions for estimating the random attenuation effects at different distances between the nodes. This is a first step towards the integration of our methodology with an accurate network simulator, which will also incorporate the proposed improvements in its channel model.

In the scope of cyber-physical systems, case studies presented in this chapter tackle the design of the communication infrastructure, i.e., the "cyber" part. However, we observe that the graph-based representation of the network topology is generic, i.e., other types of components, such as the ones composing the physical part of the system, can be easily integrated in the same formulation. For example, a set of *actuators* can be added to a WSN design problem by introducing additional component types and extending the library. An actuator can represent a physical component (e.g., hydraulic, electric) or an interface to the latter. In turn, some edges of the architecture graph can be mapped to wireless links, while others may represent physical connections (e.g., wires, valves). Our methodology allows users to co-design different subsystems within the same high-level architecture by using multiple functional flows. For instance, one could use a WSN as a communication and control infrastructure of a manufacturing system, presented in Chapter 5. The possibility of simultaneously selecting and interconnecting both physical and electronic components of a CPS at a conceptual level of design, while meeting the system-level requirements of the two subsystems and optimizing the overall cost, brings a large number of potential applications for the proposed methodology.

The work presented in this chapter suggests several promising future work directions:

- *Iterative optimization with simulation in the loop.* Proven to be efficient in several recent works [40, 87], lazy coordination of a MILP solver with a simulation model can enable accurate verification of the generated architecture. Furthermore, simulation results can be used by a learning function to propose new constraints for the MILP solver in order to guide it towards a solution that meets certain requirements. In particular, in wireless sensor networks, careful estimation of power consumption of nodes is of crucial importance. Therefore, network topologies generated by ArchEx can be verified with a power-aware simulator, such as PASES [84]. If the simulation outcome is not satisfying, then the lifetime constraints of the optimization problem have to be automatically tightened. Moreover, similarly to [40], our topology synthesis methodology can be augmented with simulation-based design space exploration and sizing of continuous network properties, such as protocol parameters.

- *Investigation of tradeoffs across HW/SW boundaries.* Several system parameters in current WSN specifications can be calibrated in future work within the same exploration problem. For example, the parameters of a network protocol, such as duty cycle, epoch (period for sending packets), number and length of slots in a superframe of a TDMA protocol, packet size, and others are currently fixed in the specification. Instead, they can be added to the formulation as real or integer decision variables. Current expressions of network requirements may need additional linearization, but in general this would enable the sizing of software parameters. Alternatively, if some expressions become impractical for being added to the MILP problem, the sizing can be performed with other tools, including simulators.

- *Joint optimization of wireless infrastructure and physical plant.* More realistic and complex CPS designs can be supported by combining the features of the proposed methodology, presented in Chapters 5 and 6. For example, same architecture template can include both wireless network components and a physical system, such as a production line. In turn, edges of the same template can be mapped to different types of connections (e.g., wireless links for the network, conveyors for the production line, some physical connectors or actuators for interfacing the two). Synthesized architecture can then include both the interconnections and the machinery of the physical part, and the topology of the communication network. Novel encoding algorithms may be proposed to keep the complexity of such versatile formulations at a reasonable level.

- *Additional improvements of the channel model.* One of our goals is the integration of the proposed channel model improvements to the PASES simulator. Overall, the presented study of the wireless channel is a good starting point, from which we can move our investigation to real industrial environments to draw more conclusions aboth the WSN radio channel and further improve the models. Such environments are typically characterized by high electromagnetic pollution, usually at low frequencies. Also, no-line-of-sight (NLOS) conditions are to be studied due to the presence of strong multipath effects, which may play a dominant role in signal path loss. Furthermore, movement scenarios are to be considered in both LOS and NLOS. From the latter we expect lower temporal stability and coherence time of the channel, in particular, due to Doppler effects related to moving object or environment.

- *Localization accuracy constraints.* Our current localization constraints (reachability of every test point by a minimum number of anchor nodes) affect the quality of localization only in an implicit way. There are some research efforts that analytically characterize the quality of localization [127] and the ranging error from RSS measurements [34]. The possibility of using such expressions as constraints in our formulation would allow us to derive *explicit* bounds on the accuracy of localization and, therefore, has to be investigated.

# 7 Conclusions and Future Work

*This chapter summarizes the contributions of the research made in this dissertation. We also suggest promising research directions for future work.*

## 7.1 Conclusions

By joining the cyber and the physical, and by forcing them to cooperate, we attempt to establish a more controllable, reliable, cost-effective, safe and connected surroundings in different spheres of our lives. Embedded electronics becomes pervasive, and its tight integration with modern mechanical and other physical systems shows promise of making the resulting holistic cyber-physical system more capable and efficient. Indeed, CPS are expected to outperform their predecessors, in which electronic and physical elements are kept separated. However, the complexity and the heterogeneity of CPS is increasing their design and verification challenges. Complex designs, for example, in the automotive and avionic domains, are typically carried out by multiple companies, teams and suppliers. Being well-known and addressed for years by different industries, nowadays design complexity is exhacerbated with the new challenges intrinsic to CPS. Design decisions made for different subsystems directly affect each other, while an error introduced early in the design may stay undetected for a long time and may lead to severe vulnerabilities in the system realization. A confident and efficient design process naturally suggests raising the level of abstraction, so that the efforts of different design teams can be synchronized, coordinated and constrained by a set of system-wide requirements.

In this dissertation we seek to advance the state of the art in cyber-physical system design by addressing the challenges of one of its earliest steps, i.e., *concept design.* We introduced a methodology for the exploration of high-level CPS architectures to efficiently investigate and prune the broad search space of interconnections of system components, and select the one that better fulfills the requirements. We apply optimization techniques, such as mixed integer linear programming, to create exploration problem formulations that can leverage the empirical advances of state-of-the-art information theoretic solvers to generate cost-effective system architectures that are correct by construction. Our formulation is capable of capturing a variety of high-level properties, such as reliability, workload, timing and energy, while abstracting complex dynamic characteristics by replacing them with steady-state or worst-case approximations. Following the platform-based design paradigm, our novel formulation keeps the implementation and application spaces separated. That is, topology selection problem (number of components and their interconnections) is decoupled from the mapping problem (choosing components from a domain-specific library). In addition, we developed a set of algorithms for *scalable* encoding and solving of exploration problems, and an extensible framework, ARCHEX 2.0, to facilitate the use of the proposed methodology by CPS designers.

In Chapter 2, based on a graph-based representation of the architecture, we introduced a common semantic domain as a set of variables and generic MILP constraints that capture the main design decisions on high-level architectures. These decisions include selecting the number of components and their direct connections (topology configuration), the paths between source and destination components (routing), and the association of all components and connections to library elements (mapping). The resulting *basis of the exploration problem* can be used to derive expressions that capture different system-level properties and requirements. All of them are built on top of the basis, sometimes with an addition of domain-specific and auxiliary variables.

Our architecture exploration methodology, presented in Chapter 3, consists of design specification, encoding, solving and analysis steps. The former one is fortified by the language of *patterns*, i.e., short expressions that reflect the requirements expressed in natural language and can be leveraged to write compact and comprehensive specifications. The encoding part uses the generic basis, proposed in Chapter 2, for instantiating application requirements. It also includes a set of algorithms for generation of approximate encoding of some entities and properties of the architecture, such as network paths and reliability constraints. Two different techniques are used to solve exploration problems: a monolithic optimization with all possible constraints, and a lazy coordination of a MILP solver and a theory solver that verifies a certain property (e.g., reliability) and guides the solver towards a feasible solution. Finally, a set of analysis techniques, e.g., reliability, timing and workload, are used to verify the corresponding aspects of the architecture.

In Chapter 4 we presented ARCHEX 2.0, an extensible framework for the exploration of CPS architectures that supports the proposed methodology. ARCHEX leverages an extensible set of patterns to facilitate the problem formulation and debugging. Pattern-based specifications are orders of magnitude smaller than automatically generated MILP formulations. The software structure of the framework relies on a set of abstract classes and reusable data structures, so that it can be customized to different CPS domains. In particular, we presented a large extension of the toolbox, which allows it to support topology synthesis problems for wireless networks.

The disseration also provides an extensive numerical evaluation of the proposed methodology on a set of case studies from different CPS domains. In Chapter 5, we demonstrated the effectiveness of our approach on two *reliability-driven* industrial design examples: aircraft electrical power distribution network and reconfigurable manufacturing system. Both of them require minimum-cost architectures while meeting the interconnection, balance, workload, timing, safety and reliability requirements. The performance and the quality of both monolithic and iterative solving algorithms was investigated. Our results confirmed that both the formulation and the algorithms can be applied to complex industrial designs, with realistic system architectures being synthesized in reasonable time (from several seconds to several minutes). Our tests also provided an empirical evidence of the efficiency of our mapping mechanism with respect to previous works [99, 12]. Increasing the size of components library only led to a logarithmic growth of problem complexity, compared to a quadratic growth when using the approach from [99, 12].

In Chapter 6, we focused on a significantly different domain, wireless networks, for synthesizing reliable and energy-efficient topologies under routing, link quality and lifetime constraints. Two relevant case studies for wireless sensor networks have been presented: data collection and localization. In particular, we demonstrated the capabilities of the proposed path encoding algorithm to generate compact, yet approximate, MILP formulations that allow the large-scale network topologies to be synthesized in reasonable time and with high cost-effectiveness. The algorithm can be calibrated to explore the tradeoff between cost and complexity. Last but not least, we proposed a set of improvements for the conventional channel models (log-distance, multi-wall) used in network design tools. These improvements help to capture the operating conditions of network designs with greater accuracy and, apart from ArchEx, can be also integrated to network simulation tools, which is planned in future work.

Our experiments confirmed the following important characteristics of the proposed architecture exploration approach:

- *Usability*, facilitated by the pattern-based language for requirement specification.
- *Expressiveness*, allowed on the theoretical side by the generic basis of the exploration problem, and on the practical side by a set of patterns. As a result, a variety of heterogeneous requirements can be captured.
- *Extensibility and reusability*, due to the modular structure of the ArchEx framework. It is possible to reuse and/or customize same requirement encodings in different domains.
- *Scalability*, which is provided by efficient algorithms for encoding and solving exploration problems.

With respect to existing satisfiability- and optimization-based approaches for concept design of CPS architectures, such as [101, 12, 108, 103], our approach has demonstrated superior expressiveness by being able to capture more design concerns. In particular, we address more general problems by allowing the sizing of components. That is, many parameters, fixed in previous formulations, can now be selected from a library of components. Similarly, by being more expressive, our techniques can replace the discrete optimization stages in hybrid optimization/simulation approaches, e.g., [40, 87]. The methodology can also be complementary to existing simulation-based solutions [23, 17, 47] and tools [107, 30].

## 7.2 Future Work

Inspired by the extensibility of the proposed architecture exploration methodology and by design examples shown in this thesis, we have outlined several promising and interesting directions of future work and research. We believe that our approach can be more tightly integrated into the cross-layer design flow of CPS by interfacing with preceding and succeeding steps via other existing tools. Also, we outline a set of potential applications to apply the methodology, which can also be motivating for improving and extending the existing formulation.

### 7.2.1 Theory and Algorithms

On the theoretical side, one could concentrate on enriching the set of existing supported requirements which would extend the methodology to a broader class of applications. For example, scheduling, reconfigurability and network coverage constraints can be studied. Also, HW/SW boundaries can be investigated at a high level, e.g., selecting a network protocol or a localization function, as well as sizing of software parameters, e.g., duty cycle, packet size or number of retransmissions. Finally, certain properties of the architecture, such as power flow, battery discharge, switching of contactors/valves or reconfiguration of a manufacturing line, can be captured in their dynamics. Encoding of such properties as mixed integer linear expressions is possible by discretizing the time scale, selecting the time horizon and sampling time, and capturing the property values at certain moments in time. The resulting problem will then include the optimization and sizing of dynamic properties and can be solved using the receding horizon approach [78]. The possibility of applying the proposed techniques for redesign/reconfiguration in runtime (on the fly) can be investigated as well.

One more theoretical direction is related to devising efficient *approximations* for heavy nonlinear constraints. An approximate algebra has been proposed in [96] for encoding of reliability constraints. Similarly, approximations with mathematically proven theoretical bounds would be advantageous, for example, for certain physical properties, which can be expressed only as piecewise linear functions. The latter is a universal technique able to replace any expression, however, the accuracy of such approximation highly depends on the number of selected linear intervals. The complexity of the formulation can also be very sensitive to the latter.

On the side of the algorithms, the development of new theory solvers and analysis techniques to be used within iterative optimization schemes would be interesting. In contrast with aforementioned approximate encodings, which facilitate monolithic problem formulations, here the goal is to decouple complex constraints and leverage conflict-driven learning routines for guiding the solver towards feasible solutions. For example, procedures for timing, scheduling and energy consumption constraints can be investigated.

Similarly, iterative optimization techniques incorporating *simulation* are a promising direction of future research. Simulation models can be used for verification of the final architecture [108], learning new constraints for the MILP formulation [87], as well as for design space exploration [40]. Using simulation in a loop with the proposed architecture exploration approach would further increase the breadth of the design space, since more system properties can be captured by the models. This also has the potential of increasing the efficiency and the scalability of existing techniques, since simulation models can evaluate "heavy" system properties, otherwise encoded as a part of a complicated MILP problem. Finally, the methodology that effectively combines optimization-based and simulation-based methods would cover more steps in the holistic CPS design flow.

In our approach, as well as in other existing ones [12, 40, 87], iterative optimization, both with external theory solvers and simulators, leverages *greedy* strategies for manipulating the MILP formulation. That is, if the generated architecture does not meet the requirements, new constraints are added to the formulation. However, if the next result is worse than previous (or infeasible), recently added constraints are not removed in an attempt to backtrack and try out another strategy. Similar to SAT/SMT solvers, a *push-pop* mechanism can be applied in learning strategies to cancel the effect of wrong actions that compromise the quality and the feasibility of the architecture. Combined with the domain-specific knowledge, backtracking mechanisms will enable more powerful and reliable iterative optimization schemes.

Another algorithmic improvement would be to devise a set of strategies for choosing $K^*$, i.e., the number of proposed candidate path implementations for every required network route in Algorithm 1, for a given system/network. Apart from the one discussed in Section 6.2.3, which is related to conducting a systematic search for different values of $K^*$, an alternative and more general approach would be to integrate our path pruning techniques in a variation of a branch-and-bound algorithm. The possibility to branch on a set of selected paths would allow to estimate and compare the best bounds of these selections and efficiently prune the search space. In contrast with the current version of the pruning method, which can be seen as an approximation, this would provide optimality guarantees and allow to calculate the optimization gap.

Finally, it would be very interesting to compare the capabilities currently provided by the methodology with the *Optimization Modulo Theories* approach [116, 28] and related tools [117, 74, 18]. OMT is a novel and very promising technology, from which we expect an advantage of better expressiveness and a drawback of limited scalability for handling a large number of real constraints. However, no applications to concept design of CPS, as well as for CPS architecture exploration in general, have been presented so far. Overall, symbolic optimization with SMT solvers has a great potential, and one has to investigate its capabilities of solving large design problems.

### 7.2.2 Tools

One potential improvement of the ArchEx framework is related to debugging optimization problems. At the moment, if the formulation is infeasible, the MILP solver only provides the information about a single constraint that caused the inconsistency. Users are able to debug the problem at a higher level of abstraction, i.e., by adding or removing certain patterns from the specification. In general, it allows them to determine the requirement that led to infeasibility of the whole problem. However, many patterns internally translate the requirements to a large number of MILP constraints, e.g., hundreds or thousands. Moreover, multiple properties can be captured within the same pattern. Therefore, the possibility of providing a minimal set of inconsistent constraints, i.e., the *unsatisfiability core*, is highly desirable. More importantly, it should be possible to *trace back* these constraints by associating them to particular requirements, so that the designer could understand the reason of inconsistency. Such information would also be helpful to locate potential errors introduced during implementation of new constraints in ARCHEX.

Interfacing ARCHEX with other design exploration tools and problem solvers, such as simulators or solvers for nonlinear (e.g., convex) optimization problems, is another important direction. This would allow the proposed methodology to be integrated to different design flows. Being capable of generating the output (system architecture) in a format that is accepted by other tools is an important step towards *interoperability*, which is still lacking between CPS design tools.

Finally, extending the proposed pattern-based language with new supported requirements would increase the number of applications, where ARCHEX can be directly applied. Also, comparing the performance and the capabilities of different MILP toolboxes for solving architecture exploration problems could be helpful for providing usage guidelines for designers. Overall, the improvements listed above can facilitate the dissemination of the ARCHEX framework in different communities and industries.

### 7.2.3 Applications

Our generic formulation allows us to represent both the embedded system (or network) and the physical plant as a single graph. In this graph, nodes can be associated with components of both types, while edges can be mapped to different communication means, e.g., wireless links, contactors, wires or valves. For instance, a sensor can be connected to a base station via a wireless channel, and to a robotic device with a physical connector. In turn, the robot can be connected to different mechanisms (e.g., machines, conveyors) with some other interface. Therefore, the architecture template in our methodology can include different infrastructures and subsystems (e.g., mechanical, communication). The functionality and the capabilities of ARCHEX, demonstrated in Chapters 5 and 6 can be jointly applied to handle more complex and realistic designs. For example, the placement of the machinery of a reconfigurable manufacturing system, studied in Chapter 5, can be optimized to occupy the minimum surface of the shop floor. The template of the RMS architecture can also include a subgraph that represents a wireless networking infrastructure that monitors the state of the machines and issues control commands (e.g., changing operation modes, redistributing product flows). Novel encoding algorithms and synthesis techniques may be proposed to keep the complexity of such versatile formulations at a reasonable level.

Existing case studies also suggest several directions of future work. They include advanced timing and reconfigurability requirements for electrical power networks and manufacturing systems, control patterns, support of multiple functional flows (which is also necessary for complex architectures discussed above), localization accuracy requirements, calibration of software parameters (e.g., of a network protocol) and iterative optimization schemes that incorporate theory solvers and simulation in the loop. These improvements are discussed in detail in Sections 5.4 and 6.5.

Finally, several new applications can be suggested. In particular, smart grid [132] is a representative class of complex cyber-physical systems. Interconnection and power balance requirements from the aircraft power system design example can be similarly applied to a smart grid case study. Routing and reliability constraints can also be enforced. The large scale of a power grid network prompts to use our algorithm for approximate encoding of network paths, i.e., proposing several candidate paths for every electrical line, as well as iterative optimization schemes for efficiently solving exploration problems. Additionally, existing case studies, such as aircraft environmental control system [40], fuel management system (Chapter 4) and body area network [87], can be elaborated.

# Bibliography

[1] (2018, Apr.) *ArchEx 2.0: CPS Architecture Exploration Framework.* [Online]. Available: https://bitbucket.org/regkirov/archex.

[2] (2018, Apr.) *Das V-Modell.* [Online]. Available: http://v-modell.iabg.de.

[3] (2018, Apr.) *Python module for Simulated Annealing Optimization.* [Online]. Available: https://github.com/perrygeo/simanneal.

[4] (2018, Apr.) *SeDuMi: a linear/quadratic/semidefinite solver for Matlab and Octave.* [Online]. Available: https://github.com/sqlp/sedumi.

[5] (2018, Apr.) *Texas Instruments: Zigbee products.* [Online]. Available: http://www.ti.com/lsds/ti/wireless-connectivity/zigbee/products.page.

[6] H Akaike. “Information theory as an extension of the maximum likelihood principle”. In: *Second International Symposium on Information Theory.* 1973, p. 267.

[7] Edoardo Amaldi, Antonio Capone, Matteo Cesana, Ilario Filippini, and Federico Malucelli. “Optimization models and methods for planning wireless mesh networks”. In: *Computer Networks* 52.11 (2008), pp. 2159–2171.

[8] Ann-Louise Andersen, Thomas Ditlev Brunoe, Kjeld Nielsen, and Carin Rösiö. “Towards a generic design method for reconfigurable manufacturing systems: Analysis and synthesis of current design methods and evaluation of supportive tools”. In: *Journal of Manufacturing Systems* 42 (2017), pp. 179–195.

[9] *Architecture Analysis and Design Language.* [Online]. Available: http://www.aadl.info/aadl/currentsite.

[10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. “The Internet of Things: A Survey”. In: *Computer networks* 54.15 (2010), pp. 2787–2805.

[11] *Aviation Maintenance Technician Handbook - Airframe (FAA-H-8083-31). Aircraft Fuel System.* Vol. 2. US Federal Aviation Administration, 2012. Chap. 14.

[12] Nikunj Bajaj, Pierluigi Nuzzo, Michael Masin, and Alberto Sangiovanni-Vincentelli. “Optimized selection of reliable and cost-effective cyber-physical system architectures”. In: *Proceedings of Design, Automation and Test in Europe (DATE).* 2015, pp. 561–566.

[13] Johann Bals, Gerhard Hofer, Andreas Pfeiffer, and Christian Schallert. “Virtual Iron Bird - a multidisciplinary modelling and simulation platform for new aircraft system architectures”. In: *Proc. of German Aerospace Conference.* 2005, pp. 1–9.

[14] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. *The Satisfiability Modulo Theories Library (SMT-LIB).* [Online]. Available: www.SMT-LIB.org. 2016.

[15] Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. "Satisfiability Modulo Theories". In: *Handbook of Satisfiability.* Ed. by Armin Biere, Hans van Maaren, and Toby Walsh. Vol. 4. IOS Press, 2009. Chap. 8.

[16] Albert Benveniste, Benoit Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone, and Christos Sofronis. "Multiple viewpoint contract-based specification and design". In: *International Symposium on Formal Methods for Components and Objects.* 2007, pp. 200–225.

[17] Ajinkya Bhave, Bruce H Krogh, David Garlan, and Bradley Schmerl. "View consistency in architectures for cyber-physical systems". In: *Proceedings of the 2011 IEEE/ACM Second International Conference on Cyber-Physical Systems (ICCPS).* 2011, pp. 151–160.

[18] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. "$\nu$Z-An Optimizing SMT Solver." In: *Proceedings of TACAS.* Vol. 15. 2015, pp. 194–199.

[19] Athanassios Boulis. "Castalia: Revealing Pitfalls in Designing Distributed Algorithms in WSN". In: *SenSys'07.* Sydney, Australia, Nov. 2007.

[20] Davide Brunelli, Ivan Minakov, Roberto Passerone, and Maurizio Rossi. "POVOMON: An Ad-hoc Wireless Sensor Network for indoor environmental monitoring". In: *IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems Proceedings* (Sept. 2014), pp. 1–6.

[21] Bruttomesso, Roberto and Pek, Edgar and Sharygina, Natasha and Tsitovich, Aliaksei. "The OpenSMT Solver." In: *Proceedings of TACAS.* Vol. 6015. 2010, pp. 150–153.

[22] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. "The MathSAT 4 SMT Solver". In: *International Conference on Computer Aided Verification (CAV).* 2008, pp. 299–303.

[23] Arquimedes Canedo and Jan H Richter. "Architectural design space exploration of cyber-physical systems using the functional modeling compiler". In: *Procedia CIRP* 21 (2014), pp. 46–51.

[24] Jiun-Jian Chang, Pi-Cheng Hsiu, and Tei-Wei Kuo. "Search-oriented deployment strategies for wireless sensor networks". In: *Proc. of ISORC.* 2007, pp. 164–171.

[25] Guillaume Chelius, Antoine Fraboulet, and Eric Fleury. "Worldsens: a fast and accurate development framework for sensor network applications". In: *Proceedings of the 2007 ACM symposium on Applied computing.* ACM. 2007, pp. 222–226.

[26] Peng Cheng, Chen-Nee Chuah, and Xin Liu. "Energy-aware node placement in wireless sensor networks". In: *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM'04).* Vol. 5. 2004, pp. 3210–3214.

[27] A. Cimatti, L. Palopoli, and Y. Ramadian. "Symbolic Computation of Schedulability Regions Using Parametric Timed Automata". In: *Real-Time Systems Symposium, 2008.* Nov. 2008, pp. 80–89.

[28] Alessandro Cimatti, Anders Franzén, Alberto Griggio, Roberto Sebastiani, and Cristian Stenico. "Satisfiability Modulo the Theory of Costs: Foundations and Applications." In: *Proceedings of TACAS*. Vol. 6015. 2010, pp. 99–113.

[29] Davide Dardari, Pau Closas, and Petar M Djuric. "Indoor tracking: Theory, methods, and technologies". In: *IEEE Transactions on Vehicular Technology* 64.4 (2015), pp. 1263–1278.

[30] Abhijit Davare, Douglas Densmore, Liangpeng Guo, Roberto Passerone, Alberto L Sangiovanni-Vincentelli, Alena Simalatsar, and Qi Zhu. "metro II: A Design Environment for Cyber-Physical Systems". In: *ACM Transactions on Embedded Computing Systems* 12.1s (2013).

[31] Leonardo De Moura and Nikolaj Bjørner. "Z3: An efficient SMT solver". In: *Tools and Algorithms for the Construction and Analysis of Systems* (2008), pp. 337–340.

[32] Jianping Dou, Xianzhong Dai, and Zhengda Meng. "Optimisation for multi-part flow-line configuration of reconfigurable manufacturing system using GA". In: *International Journal of Production Research* 48.14 (2010), pp. 4071–4100.

[33] Qi Duan, Saeed Al-Haj, and Ehab Al-Shaer. "Provable configuration planning for wireless sensor networks". In: *Proc. of the 8th International Conference on Network and Service Management*. 2012.

[34] Jared Dulmage, Robert Cioffi, Michael P Fitz, and Danijela Cabric. "Characterization of distance error with received signal strength ranging". In: *Proceedings of Wireless Communications and Networking Conference (WCNC)*. IEEE. 2010, pp. 1–6.

[35] Bruno Dutertre. "Yices 2.2". In: *Proceedings of Computer-Aided Verification (CAV'2014)*. Vol. 8559. July 2014, pp. 737–744.

[36] *DYMOLA Systems Engineering*. [Online]. Available: https://www.3ds.com/products-services/catia/products/dymola.

[37] Edward A. Lee and Sanjit A. Seshia. *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*. Second. MIT Press, 2017.

[38] Andreas Eisenblätter and H-F Geerdes. "Wireless network design: solution-oriented modeling and mathematical optimization". In: *Wireless Communications, IEEE* 13.6 (2006), pp. 8–14.

[39] Hoda ElMaraghy, ed. *Changeable and Reconfigurable Manufacturing Systems*. 1st ed. Springer Series in Advanced Manufacturing. Springer-Verlag London, 2009.

[40] John Finn, Pierluigi Nuzzo, and Alberto Sangiovanni-Vincentelli. "A mixed discrete-continuous optimization scheme for cyber-physical system architecture exploration". In: *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*. 2015, pp. 216–223.

[41] Sicun Gao, Soonho Kong, and Edmund M Clarke. "dReal: An SMT solver for nonlinear theories over the reals". In: *International Conference on Automated Deduction*. 2013, pp. 208–214.

[42] Jerry D Gibson, ed. *Mobile Communications Handbook*. 3rd ed. CRC Press, 2012.

[43] Kapil Kumar Goyal, PK Jain, and Madhu Jain. “Optimal configuration selection for reconfigurable manufacturing system using NSGA II and TOPSIS”. In: *International Journal of Production Research* 50.15 (2012), pp. 4175–4191.

[44] Yanying Gu, Anthony Lo, and Ignas Niemegeers. “A survey of indoor positioning systems for wireless personal networks”. In: *Communications Surveys & Tutorials, IEEE* 11.1 (2009), pp. 13–32.

[45] Gaoyang Guan, Wei Dong, Yi Gao, Kaibo Fu, and Zhihao Cheng. “TinyLink: A Holistic System for Rapid Development of IoT Applications”. In: *Proceedings of the 23rd ACM Annual International Conference on Mobile Computing and Networking*. 2017, pp. 383–395.

[46] Vehbi C Gungor and Gerhard P Hancke. “Industrial wireless sensor networks: Challenges, design principles, and technical approaches”. In: *IEEE Transactions on industrial electronics* 56.10 (2009), pp. 4258–4265.

[47] Liangpeng Guo, Qi Zhu, Pierluigi Nuzzo, Roberto Passerone, Alberto Sangiovanni-Vincentelli, and Edward A Lee. “Metronomy: a function-architecture co-simulation framework for timing verification of cyber-physical systems”. In: *Proc. of CODES+ ISSS*. 2014, pp. 1–10.

[48] *Gurobi Optimization - The State-of-the-Art Mathematical Programming Solver.* [Online]. Available: http://http://www.gurobi.com/.

[49] Xiaofeng Han, Xiang Cao, Errol L Lloyd, and Chien-Chung Shen. “Fault-tolerant relay node placement in heterogeneous wireless sensor networks”. In: *IEEE Transactions on Mobile Computing* 9.5 (2010), pp. 643–656.

[50] C. Hang, P. Manolios, and V. Papavasileiou. “Synthesizing Cyber-Physical Architectural Models with Real-Time Constraints”. In: *Proceedings of the International Conference on Computer Aided Verification (CAV)*. Dec. 2011.

[51] Tom Haute et al. “Performance analysis of multiple Indoor Positioning Systems in a healthcare environment”. In: *Intern. Jour. of Health Geographics* 15.1 (2016).

[52] Danping He, Gabriel Mujica, Jorge Portilla, and Teresa Riesgo. “Modelling and planning reliable wireless sensor networks based on multi-objective optimization genetic algorithm with changeable length”. In: *Journal of Heuristics* 21.2 (2015), pp. 257–300.

[53] Mario Hermann, Tobias Pentek, and Boris Otto. “Design Principles for Industrie 4.0 Scenarios”. In: *Proc. 49th Hawaii International Conference on System Sciences*. 2016, pp. 3928–3937.

[54] *IBM ILOG CPLEX Optimizer.* [Online]. Available: http://www.ibm.com/software/commerce/optimization/cplex-optimizer/.

[55] *IPOPT - Interior Point OPTimizer.* [Online]. Available: https://projects.coin-or.org/Ipopt.

[56] Muhammad Iqbal, Muhammad Naeem, Alagan Anpalagan, Ashfaq Ahmed, and Muhammad Azam. “Wireless sensor network optimization: multi-objective paradigm”. In: *Sensors* 15.7 (2015), pp. 17572–17620.

[57] Timofei Istomin, Ramona Marfievici, Amy L Murphy, and Gian Pietro Picco. “TRIDENT: In-field Connectivity Assessment for Wireless Sensor Networks”. In: *Proceedings of the 6th Extreme Conference on Communication and Computing (ExtremeCom)*. 2014.

[58] J. Tang et al. “Relay node placement in large scale wireless sensor networks”. In: *Computer communications* 29.4 (2006), pp. 490–501.

[59] Jeff C Jensen, Danica H Chang, and Edward A Lee. “A model-based design methodology for cyber-physical systems”. In: *7th IEEE International Wireless Communications and Mobile Computing Conference (IWCMC)*. 2011, pp. 1666–1671.

[60] *JModelica.org: Open Source Modelica Platform for Modeling, Simulation and Optimization.* [Online]. Available: http://www.jmodelica.org.

[61] Damien B Jourdan and Olivier L de Weck. “Multi-objective genetic algorithm for the automated planning of a wireless sensor network to monitor a critical facility”. In: *Proc. of SPIE Defense and Security Symposium*. 2004, pp. 565–575.

[62] Dmitrii Kirov, Pierluigi Nuzzo, Roberto Passerone, and Alberto Sangiovanni-Vincentelli. “ArchEx: An Extensible Framework for the Exploration of Cyber-Physical System Architectures”. In: *Proceedings of the 54th Design Automation Conference (DAC)*. 2017.

[63] Dmitrii Kirov, Pierluigi Nuzzo, Roberto Passerone, and Alberto Sangiovanni-Vincentelli. “Optimized Selection of Wireless Network Topologies and Components via Efficient Pruning of Feasible Paths”. In: *Proceedings of the 55th Design Automation Conference (DAC)*. 2018.

[64] Dmitrii Kirov, Roberto Passerone, and Massimo Donelli. “Statistical characterization of the 2.4 GHz radio channel for WSN in indoor office environments”. In: *21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2016, pp. 1–9.

[65] Andreas Köpke, Michael Swigulski, Daniel Willkomm, and Karl Wessel. “Simulating Wireless and Mobile Networks in OMNeT++ The MiXiM Vision”. In: *OMNeT++ Workshop ’08 Marseille, France*. ICST, 2008.

[66] Yoram Koren, Xi Gu, and Weihong Guo. “Reconfigurable manufacturing systems: Principles, design, and future trends”. In: *Frontiers of Mechanical Engineering* (2017), pp. 1–16.

[67] Yoram Koren and Moshe Shpitalni. “Design of Reconfigurable Manufacturing Systems”. In: *Journal of manufacturing systems* 29.4 (2010), pp. 130–141.

[68] Pratyush Kumar, Devesh B Chokshi, and Lothar Thiele. “A satisfiability approach to speed assignment for distributed real-time systems”. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*. IEEE. 2013, pp. 749–754.

[69] Tolga Kurtoglu, Peter Bunus, and Johan De Kleer. “Simulation-based design of aircraft electrical power systems”. In: *Proceedings of the 8th International Modelica Conference, Dresden, Germany*. 063. Mar. 2011, pp. 704–712.

[70] *LabVIEW - National Instruments*. [Online]. Available: http://www.ni.com/en-us/shop/labview.

[71] Edward A. Lee. “Cyber Physical Systems: Design Challenges”. In: *11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. 2008, pp. 363–369.

[72] Edward A Lee, Stephen Neuendorffer, and Michael J Wirthlin. “Actor-oriented design of embedded hardware and software systems”. In: *Journal of circuits, systems, and computers* 12.3 (2003), pp. 231–260.

[73] Jay Lee, Behrad Bagheri, and Hung-An Kao. “A cyber-physical systems architecture for industry 4.0-based manufacturing systems”. In: *Manufacturing Letters* 3 (2015), pp. 18–23.

[74] Yi Li, Aws Albarghouthi, Zachary Kincaid, Arie Gurfinkel, and Marsha Chechik. “Symbolic optimization with SMT solvers”. In: *ACM SIGPLAN Notices*. Vol. 49. 1. 2014.

[75] Hui Liu, Houshang Darabi, Pat Banerjee, and Jing Liu. “Survey of Wireless Indoor Positioning Techniques and Systems”. In: *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)* 37.6 (Nov. 2007), pp. 1067–1080.

[76] J. Löfberg. “YALMIP : A Toolbox for Modeling and Optimization in MATLAB”. In: *Proceedings of the CACSD Conference*. Taiwan, 2004.

[77] Corinne Lucet and Jean-François Manouvrier. “Exact methods to compute network reliability”. In: *Statistical and Probabilistic Models in Reliability*. Springer, 1999, pp. 279–294.

[78] Mehdi Maasoumy, Pierluigi Nuzzo, Forrest Iandola, Maryam Kamgarpour, Alberto Sangiovanni-Vincentelli, and Claire Tomlin. “Optimal load management system for aircraft electric power distribution”. In: *52 IEEE Annual Conference on Decision and Control (CDC)*. 2013, pp. 2939–2945.

[79] Manolios, Panagiotis and Pais, Jorge and Papavasileiou, Vasilis. “The Inez Mathematical Programming Modulo Theories Framework”. In: *International Conference on Computer Aided Verification (CAV)*. 2015, pp. 53–69.

[80] Panagiotis Manolios and Vasilis Papavasileiou. “ILP modulo theories”. In: *International Conference on Computer Aided Verification*. Springer. 2013, pp. 662–677.

[81] Guoqiang Mao, Barış Fidan, and Brian DO Anderson. “Wireless sensor network localization techniques”. In: *Computer networks* 51.10 (2007), pp. 2529–2553.

[82] *MATLAB - Mathworks*. [Online]. Available: https://www.mathworks.com/products/matlab.html.

[83] Alan McGibney, Antony Guinard, and Dirk Pesch. “Wi-Design: A modelling and optimization tool for wireless embedded systems in buildings”. In: *Local Computer Networks (LCN), 2011 IEEE 36th Conference on*. IEEE. 2011, pp. 640–648.

[84] Ivan Minakov and Roberto Passerone. “PASES: An energy-aware design space exploration framework for wireless sensor networks”. In: *Journal of Systems Architecture* 59.8 (Sept. 2013), pp. 626–642. ISSN: 13837621.

[85] Ivan Minakov, Roberto Passerone, Alessandra Rizzardi, and Sabrina Sicari. “A comparative study of recent wireless sensor network simulators”. In: *ACM Transactions on Sensor Networks (TOSN)* 12.3 (2016).

[86] *Modelica Language.* [Online]. Available: http://www.modelica.org.

[87] Ali Moin, Pierluigi Nuzzo, Alberto L Sangiovanni-Vincentelli, and Jan M Rabaey. “Optimized Design of a Human Intranet Network”. In: *Proceedings of the 54th Design Automation Conference (DAC)*. 2017.

[88] László Monostori et al. “Cyber-physical systems in manufacturing”. In: *CIRP Annals-Manufacturing Technology* 65.2 (2016), pp. 621–641.

[89] *MOSEK optimization software.* [Online]. Available: https://www.mosek.com/products/mosek/.

[90] H. Neema, Z. Lattmann, et al. “Design space exploration and manipulation for cyber physical systems”. In: *Proc. Workshop Design Space Exploration of Cyber-Physical Systems*. 2014.

[91] Sandeep Neema, Janos Sztipanovits, Gabor Karsai, and Ken Butts. “Constraint-based design-space exploration and model synthesis”. In: *International Workshop on Embedded Software (EmSoft)*. 2003, pp. 290–305.

[92] John A Nelder and Roger Mead. “A simplex method for function minimization”. In: *The Computer Journal* 7.4 (1965), pp. 308–313.

[93] Robert Nieuwenhuis and Albert Oliveras. “On SAT modulo theories and optimization problems”. In: *International conference on theory and applications of satisfiability testing*. 2006, pp. 156–169.

[94] *NPSOL - software for solving constrained optimization problems (nonlinear programs).* [Online]. Available: http://www.sbsi-sol-optimize.com/asp/sol_product_npsol.htm.

[95] P. Nuzzo, A. Puggelli, S.A. Seshia, and A. Sangiovanni-Vincentelli. “CalCS: SMT solving for non-linear convex constraints”. In: *Proceedings of the International Conference on Formal Methods in Computer Aided Design (FMCAD)*. Oct. 2010, pp. 71–79.

[96] Pierluigi Nuzzo. “Compositional Design of Cyber-Physical Systems Using Contracts”. PhD thesis. Electrical Engineering and Computer Sciences, University of California at Berkeley, 2015.

[97] Pierluigi Nuzzo and Alberto Sangiovanni-Vincentelli. “Let’s get physical: Computer science meets systems”. In: *From Programs to Systems. The Systems Perspective in Computing*. Springer, 2014, pp. 193–208.

[98] Pierluigi Nuzzo, Alberto L Sangiovanni-Vincentelli, Davide Bresolin, Luca Geretti, and Tiziano Villa. “A platform-based design methodology with contracts and related tools for the design of cyber-physical systems”. In: *Proceedings of the IEEE* 103.11 (2015), pp. 2104–2132.

[99] Pierluigi Nuzzo, Huan Xu, Necmiye Ozay, John B Finn, Alberto L Sangiovanni-Vincentelli, Richard M Murray, Alexandre Donzé, and Sanjit A Seshia. “A contract-based methodology for aircraft electric power system design”. In: *IEEE Access* 2 (2014), pp. 1–25.

[100] *OMG System Modeling Language.* [Online]. Available: http://www.omg.org/spec/SysML.

[101] Steffen Peter and Tony Givargis. “Component-Based Synthesis of Embedded Systems Using Satisfiability Modulo Theories”. In: *ACM Trans. on Des. Automation of Electr. Systems* 20.4 (2015).

[102] Quang-Dung Pham and Yves Deville. “Solving the Longest Simple Path Problem with Constraint-Based Techniques.” In: *Proceedings of the International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR)*. Springer. 2012, pp. 292–306.

[103] Alessandro Pinto, Sandor Becz, and Hayden M Reeve. “Correct-by-construction design of aircraft electric power systems”. In: *AIAA ATIO/ISSMO Conference*. 2010, pp. 1–11.

[104] Alessandro Pinto, Luca P Carloni, and Alberto Sangiovanni-Vincentelli. “COSI: A framework for the design of interconnection networks”. In: *IEEE Design & Test of Computers* 25.5 (2008).

[105] Alessandro Pinto, Luca P Carloni, and Alberto L Sangiovanni-Vincentelli. “A communication synthesis infrastructure for heterogeneous networked control systems and its application to building automation and control”. In: *Proc. of EMSOFT*. 2007, pp. 21–29.

[106] Alessandro Pinto, Massimiliano D’Angelo, Carlo Fischione, Eelco Scholte, and Alberto Sangiovanni-Vincentelli. “Synthesis of embedded networks for building automation and control”. In: *Proceedings of the ACC*. 2008, pp. 920–925.

[107] Claudius Ptolemaeus, ed. *System Design, Modeling, and Simulation Using Ptolemy II*. Ptolemy.org, 2014. ISBN: 9781304421067. URL: http://ptolemy.org/books/Systems.

[108] Alberto Puggelli, Mohammad Mostafizur Rahman Mozumdar, Luciano Lavagno, and Alberto L Sangiovanni-Vincentelli. “Routing-aware design of indoor wireless sensor networks using an interactive tool”. In: *IEEE Systems Journal* 9.3 (2015), pp. 714–727.

[109] Ragunathan Raj Rajkumar, Insup Lee, Lui Sha, and John Stankovic. “Cyber-physical systems: the next computing revolution”. In: *Proceedings of the 47th Design Automation Conference (DAC)*. ACM. 2010, pp. 731–736.

[110] Theodore S Rappaport. *Wireless communications: principles and practice*. Vol. 2. Prentice hall PTR New Jersey, 1996.

[111] Alessandro EC Redondi and Edoardo Amaldi. “Optimizing the placement of anchor nodes in RSS-based indoor localization systems”. In: *Ad Hoc Networking Workshop (MED-HOC-NET)*. IEEE. 2013, pp. 8–13.

[112] Felix Reimann, Martin Lukasiewycz, Michael Glass, Christian Haubelt, and Jürgen Teich. “Symbolic system synthesis in the presence of stringent real-time constraints”. In: *Proceedings of the 48th Design Automation Conference*. ACM. 2011, pp. 393–398.

[113] Kay Romer and Friedemann Mattern. “The design space of wireless sensor networks”. In: *IEEE Wireless Communications Journal* 11.6 (2004), pp. 54–61.

[114] Alberto Sangiovanni-Vincentelli. “Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design”. In: *Proceedings of the IEEE* 95.3 (Mar. 2007), pp. 467–506.

[115] Alberto Sangiovanni-Vincentelli, Werner Damm, and Roberto Passerone. “Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems”. In: *European Journal of Control* 18.3 (2012), pp. 217–238.

[116] Roberto Sebastiani and Silvia Tomasi. “Optimization modulo theories with linear rational costs”. In: *ACM Transactions on Computational Logic (TOCL)* 16.2 (2015).

[117] Roberto Sebastiani and Patrick Trentin. “OptiMathSAT: a tool for optimization modulo theories”. In: *International Conference on Computer Aided Verification*. 2015, pp. 447–454.

[118] Yasser Shoukry, Pierluigi Nuzzo, Alberto L Sangiovanni-Vincentelli, Sanjit A Seshia, George J Pappas, and Paulo Tabuada. “SMC: Satisfiability modulo convex optimization”. In: *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*. 2017, pp. 19–28.

[119] *Simulink - Simulation and Model-Based Design*. [Online]. Available: https://www.mathworks.com/products/simulink.html.

[120] David B Smith, Leif W Hanlen, Jian Andrew Zhang, Dino Miniutti, David Rodda, and Ben Gilbert. “First-and second-order statistical characterizations of the dynamic body area propagation channel of various bandwidths”. In: *Annals of telecommunications* 66.3-4 (2011), pp. 187–203.

[121] John Patrick Spicer. “A design methodology for scalable machining systems”. PhD thesis. University of Michigan, 2002.

[122] P Spicer, Yoram Koren, Moshe Shpitalni, and D Yip-Hoi. “Design principles for machining system configurations”. In: *CIRP Annals-Manufacturing Technology* 51.1 (2002), pp. 275–280.

[123] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. “Real-time calculus for scheduling hard real-time systems”. In: *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*. Vol. 4. 2000, pp. 101–104.

[124] András Varga and Rudolf Hornig. “An overview of the OMNeT++ simulation environment”. In: *Proc. of the SIMUTools conference*. 2008, p. 60.

[125] Federico Viani, Leonardo Lizzi, Paolo Rocca, Manuel Benedetti, Massimo Donelli, and Andrea Massa. “Object tracking through RSSI measurements in wireless sensor networks”. In: *Electronics Letters* 44.10 (2008).

[126] Wayne L Winston. *Operations research: applications and algorithms*. Duxbury press Belmont, CA, 2004.

[127] Zheng Yang and Yunhao Liu. “Quality of trilateration: Confidence-based iterative localization”. In: *IEEE Transactions on Parallel and Distributed Systems* 21.5 (2010), pp. 631–640.

[128] Jin Y Yen. “Finding the K Shortest Loopless Paths in a Network”. In: *Management Science* 17.11 (1971), pp. 712–716.

[129] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. “Wireless sensor network survey”. In: *Computer networks* 52.12 (2008), pp. 2292–2330.

[130] Mohamed Younis and Kemal Akkaya. “Strategies and techniques for node placement in wireless sensor networks: A survey”. In: *Ad Hoc Networks* 6.4 (2008), pp. 621–655.

[131] Youssef, AMA and ElMaraghy, HA. “Availability consideration in the optimal selection of multiple-aspect RMS configurations”. In: *International journal of production research* 46.21 (2008), pp. 5849–5882.

[132] Xinghuo Yu and Yusheng Xue. “Smart grids: A cyber–physical systems perspective”. In: *Proceedings of the IEEE* 104.5 (2016), pp. 1058–1070.