Methodology and Tools for Next Generation Cyber-Physical Systems: The iCyPhy Approach

Pierluigi Nuzzo, Alberto L. Sangiovanni-Vincentelli, Richard M. Murray
Department of EECS, University of California at Berkeley, Berkeley, CA 94720, USA
Engineering and Applied Science, California Institute of Technology, Pasadena, CA 91125, USA

Abstract. The realization of complex, cyber-physical “systems of systems” can substantially benefit from model-based hierarchical and compositional methodologies to make their design possible let alone optimal. In this paper, we introduce the methodology being developed within the industrial Cyber-Physical (iCyPhy) research consortium, which addresses the complexity and heterogeneity of cyber-physical systems by formalizing the design process in a hierarchical and compositional way, and provides a unifying framework where different modeling, analysis and synthesis tools can seamlessly interconnect. We use assume-guarantee contracts and their algebra (e.g. composition, conjunction, refinement) to provide formal support to the entire design flow. The design is carried out as a sequence of refinement steps from a high-level specification (top-down phase) to an implementation built out of a library of components at the lower level (bottom-up phase). At each step, the design is refined by combining synthesis from requirements, optimization and simulation-based design space exploration methods. We illustrate our approach on design examples of embedded controllers for aircraft power distribution and air management systems.

1. Introduction

Emerging information technology scenarios feature large collections of sensors, actuators, and local computing units that are immersed in physical systems and connected to the cloud to provide societal-scale services. These complex, heterogeneous and distributed “systems of systems” will require model-based development and compositional methods to make their design possible (Sangiovanni-Vincentelli, et al. 2012). However, in cyber-physical systems (CPS), embedded controllers are so tightly intertwined with the underlying physical processes that modeling efforts often result in poor abstractions, thus making it difficult to validate their behavior and achieve both accurate and efficient system-level performance estimations (Nuzzo and Sangiovanni-Vincentelli 2014).

Several modeling languages and tools have been proposed over the years to enable checking system level properties, or explore alternative architectural solutions for the same set of requirements. Among others, we recall: generic modeling frameworks, such as Matlab/Simulink$^1$ or Ptolemy II$^2$; hardware description languages, such as Verilog$^3$ or VHDL$^4$, and transaction-level modeling tools, such as SystemC$^5$, together

---

$^1$http://www.mathworks.com/products/simulink
$^2$http://ptolemy.eecs.berkeley.edu
$^3$http://www.verilog.com/
$^4$http://www.vhdl.org
$^5$http://www.accellera.org/downloads/standards/systemc
with their respective analog-mixed-signal extensions; modeling languages specifically tailored for acausal multi-physics systems, such as Modelica; and languages for architecture modeling, such as SysML and AADL. Some of these languages and tools focus on simulation while others are geared towards performance modeling, analysis and verification. However, an all-encompassing framework is very difficult to assemble and most designers resort to patched flows. Some industrial domains, such as automotive and aerospace, use the “V” diagram that was proposed several years ago by the German defense companies (Sangiovanni-Vincentelli, et al. 2012). In this methodology, there is a top-down design phase that ends with system implementation (the left arm of the V) followed by an integration and verification process that ends with the verification of the entire system (the right arm of the V). This water-fall method has produced good results when the complexity of the designs was relatively small. When complexity scales up, we cannot simply wait to initiate the verification phase after the design is completed. Rather we should favor early verification and continuous monitoring of the design while the refinement steps are taken. In addition, we should favor “formality” in all aspects of the design flow to allow analysis and even synthesis with guaranteed properties of the final outcome of the process.

In this paper we present the methodology that has been supported by a mixed Industrial (IBM and UTC)-University (Berkeley and Caltech) consortium, called industrial Cyber Physical systems (iCyPhy), which has been formed in 2013 with the goal of developing methodologies, models and tools to implement this vision. The approach is based on the Platform-Based Design (PBD) methodology (Sangiovanni Vincentelli 2007) that builds reliable abstractions of physical behaviors, is based on early verification, and enables correct development of cyber-physical systems in a hierarchical and compositional manner. In this paper, we advocate the use of assume-guarantee (A/G) contracts to provide formal support to the PBD process, including synthesis and verification of architectures and control protocols for CPS (Nuzzo et al. 2014, Iannopollo et al. 2014). In particular, we adopt the A/G contract framework, as introduced by (Benveniste, et al. 2012) to reason about requirements and their refinement during the design process. Because of the explicit distinction between component and environment, A/G contracts are considered rigorous yet intuitive tools, whose goal is to give certain performance figures for the system under specific assumptions on its environment.

The notion of contracts originates in the context of compositional assume-guarantee reasoning (Clarke et al. 2008) that has been used mostly for software verification. In a contract framework, design and verification complexity is reduced by decomposing system-level tasks into more manageable sub-problems at the component level, under a set of assumptions. System properties can then be inferred or proved based on component properties. Rigorous contract theories have been developed over the years, including assume-guarantee contracts (Benveniste et al. 2008) and interface theories (de Alfaro and Henzinger 2001). However, their concrete adoption in system design is

http://www.accellera.org/downloads/standards/systemc/ams
7 https://www.modelica.org/
8 http://www.omg.org/spec/SysML
9 http://www.aadl.info/aadl/currentsite
10 http://v-modell.iabg.de/
still in its infancy, a major challenge being the absence of a comprehensive modeling formalism for cyber-physical systems, due to their complexity and heterogeneity (Sangiovanni-Vincentelli et al. 2012, Benveniste et al. 2012).

The paper is organized as follows. In Sec. 2, we provide background for PBD and contracts. In Sec. 3, we discuss the structure of our methodology, combining A/G contracts with PBD. Finally, in Sec. 4, we demonstrate the effectiveness of our approach on design examples of distributed controllers for avionic vehicle management systems, including power distribution and air management.

2. Background: Platform-Based Design and Contracts

At the highest level of abstraction, we model a CPS as a control system, composed of a physical plant, including sensors and actuators, and an embedded controller. The controller runs a control algorithm to restrict the behaviors of the plant so that all the closed-loop behaviors satisfy a set of system requirements. Specifically, we consider reactive controllers, i.e. controllers that maintain an ongoing relation with their environment by appropriately reacting to it. Our goal is to design the system architecture, i.e. the interconnection among system components, and the control algorithm, to satisfy the set of top-level requirements.

Following the PBD paradigm, our design flow progresses according to precisely defined abstraction levels. At each step, top-down refinements of high-level specifications are mapped onto bottom-up abstractions and characterizations of potential implementations. Each abstraction layer is defined by a design platform, which is a library (collection) of components and composition rules. In the top-down phase of each design step, we formalize the high-level requirements and associate them to different components or different viewpoints (aspects) of the system (e.g. reliability, safety, performance). In the bottom-up phase, we build the component library to model the system architecture (including both plant and controller) and the control algorithm.

If the design process is carried out as a sequence of refinement steps from the most abstract representation of the design platform (top-level requirements) to its most concrete representation (physical implementation), providing guarantees on the correctness of each step becomes essential. Specifically, we seek mechanisms to formally prove that: (i) a set of requirements are valid, i.e. there exists an implementation satisfying all of them; (ii) an aggregation of components are compatible, i.e. there exists a legal environment in which they can correctly operate; (iii) an aggregation of components refines a specification, i.e. it implements the specification and is able to operate in any environment admitted by it. Moreover, whenever possible, we require the above proofs to be performed automatically and efficiently, to tackle the complexity of today’s CPS. Therefore, to formalize the above design concepts, and enable the realization of system architectures and control algorithms in a hierarchical and compositional manner, we resort to the algebra of contracts.

A platform component $M$ can be seen as an abstraction representing an element of a design, characterized by a set of variables, ports, and models, both functional, assigning a history of “values” to variables and ports, and extra-functional, i.e. maps
corresponding to particular valuations of variables and ports\textsuperscript{11}. A contract $C$ for $M$ is a pair of assertions $(A,G)$, called the assumptions and the guarantees, each representing a specific set of behaviors over the component variables. An implementation satisfies a contract whenever it satisfies its guarantee, subject to the assumption. Formally, using set theoretic operations, we have $M \cap A \subseteq G$, where $M$ and $C$ have the same variables. A component $E$ is a legal environment for $C$ whenever $E \subseteq A$. A contract is consistent when the set of implementations satisfying it is not empty, i.e. it is feasible to develop implementations for it. This amounts to verify that $G$ is not empty\textsuperscript{12}. Let $M$ be any implementation for $C$, then $C$ is compatible if there exists a legal environment $E$ for $M$, i.e. if and only if $A$ is not empty. The intent is that a component satisfying contract $C$ can only be used in the context of a compatible environment.

Contracts associated to different components can be combined according to different rules. Similar to parallel composition of components, parallel composition ($\otimes$) of contracts can be used to construct composite contracts out of simpler ones. Let $M_1$ and $M_2$ be two composable components that satisfy, respectively, contracts $C_i$ and $C_2$. Then, $M_1 \times M_2$ is a valid composition if $M_1$ and $M_2$ are compatible. This can be checked by first computing the contract composition $C_{12} = C_1 \otimes C_2$ and then checking whether $C_{12}$ is compatible. A similar approach can be used to check consistency between $C_i$ and $C_2$. To compose multiple views of the same component that need to be satisfied simultaneously, the conjunction ($\land$) of contracts can also be defined so that if $M$ satisfies $C_i \land C_2$, then $M$ satisfies both $C_i$ and $C_2$. Contract conjunction can be computed by defining a preorder on contracts, which formalizes a notion of refinement, or substitutability. We say that $C_i$ refines $C_2$ if and only if $A_1 \supseteq A_2$ and $G_1 \subseteq G_2$. Refinement amounts to relaxing assumptions and reinforcing guarantees, therefore strengthening the contract. If refinement holds, we can replace $C_2$ with $C_i$; if $M$ satisfies $C_i$, then it will also satisfy $C_2$. On the other hand, if $E$ is a legal environment for $C_2$, then it will also be a legal environment for $C_i$. Mathematical expressions for computing contract composition and conjunction can be found in (Benveniste et al. 2012).

Since compatibility is assessed among components at the same abstraction layer, the contracts presented above can be denoted as horizontal contracts. On the other hand, vertical contracts are used to verify whether the system obtained by composing the library elements according to the horizontal contracts satisfies the requirements posed at the higher level of abstraction. If these sets of contracts are satisfied, the mapping mechanism of PBD can be used to produce design refinements that are correct by construction. Therefore, vertical contracts are tightly linked to the notions of mapping of an application onto an implementation platform, as further detailed in Sec. 3.

3. The iCyPhy Methodology

The structure of the proposed methodology is represented in Fig. 1 (a). Our focus is on top-level system design exploration, since it is at this level that both the highest

\textsuperscript{11} In what follows, we use the term variables to denote both component variables and ports. Moreover, we also use the symbol $M$ to denote the set of behaviors of component $M$.

\textsuperscript{12} For this definition and the following ones, we always make the technical assumption that contracts are in saturated (canonical) form. See (Benveniste et al. 2012) for further details on this assumption.
challenges and the highest benefits appear in today’s CPS design. The design articulates into two main steps, namely, system architecture design and control design. The system architecture design step instantiates system components and interconnections among them to generate an optimal architecture while guaranteeing the desired performance, safety and reliability. Typically, this design step includes the definition of both the embedded system and the plant architectures. The embedded system architecture consists of software, hardware, and communication components, while the plant architecture depends on the physical system under control, and can consist of mechanical, electrical, hydraulic or thermal components. Sensors and actuators reside at the boundary between the embedded system and the plant. Given an architecture, the control design step includes the exploration of the control algorithm and its deployment on the embedded platform.

The above two steps are however connected. The correctness of the controller needs to be enforced in conjunction with the assumptions on the plant. Similarly, performance and reliability of an architecture should be assessed for the plant in closed loop with the controller. In this context, the role of contracts is to incorporate the information on the environment conditions under which each entity is expected to operate, to achieve independent implementation of system architecture and control. At the highest level of abstraction, the starting point is a set of requirements, predominantly written in text-based languages that are not suitable for mathematical analysis and verification. The result is a lower level model of both the architecture and the control algorithm to be further refined in subsequent design stages. In what follows, we provide details on the three phases of our methodology, namely, the top-down requirement formalization phase, the bottom-up library generation phase, and the design exploration phase, where requirements are mapped into implementations out of the available component libraries.
3.1 Requirement Formalization

We use contracts to formalize top-level requirements, allocate them to lower-level components, and analyze them for early validation of design constraints. Requirement analysis can often be challenging, because of the lack of familiarity with formal languages among system engineers. Moreover, it is significantly different than traditional formal verification, where a system model is compared against a set of requirements. Since there is not yet a system at this stage, requirements themselves are the only entities to be analyzed.

A framework for automatic requirement engineering has recently been developed by leveraging modal interfaces, an automata-based formalism, as the underlying specification theory (Benveniste et al. 2012). However, to retain a correspondence between informal requirements and formal statements, declarative, “property-based” approaches using some temporal logic are gaining increasing interest. They contrast imperative, “model-based” approaches, which tend to be impractical for requirement validation. In fact, constructing a hybrid model to capture all the behaviors allowed by the requirements often entails considering all possible combinations of system variables. Moreover, since hybrid models are usually hard to parameterize, small changes in the requirements become soon hard to map into changes in the corresponding models.

Our approach to requirement engineering uses instead A/G contracts, which allow specifying different kinds of requirements using different formalisms, to reflect the different viewpoints and domains in a heterogeneous system. When specifying the system architecture, steady-state (static) requirements, interconnection rules, component dimensions are captured by static contracts, expressed via arithmetic constraints on Boolean and real variables to, respectively, model discrete and continuous design choices. Then, compatibility, consistency and refinement checking translate into checking feasibility of conjunctions or disjunction of constraints, which can be solved via queries to Satisfiability Modulo Theory (SMT) solvers (Barret et al. 2009, Nuzzo et al. 2010) or mathematical optimization software, such as mixed integer-linear program, mixed integer-semidefinite-positive program, or mixed integer-nonlinear program solvers.

When specifying the control algorithm, safety and real-time requirements are instead better captured by contracts expressed using temporal logic constructs. In particular, linear temporal logic (LTL) (Pnueli 1977) allows reasoning about the temporal behaviors of systems characterized by Boolean, discrete-time signals or sequences of events (discrete event abstraction in Fig. 1a). Then, compatibility, consistency and refinement checking of LTL contracts translate into LTL satisfiability checking problems, which can be solved via queries to model checkers (Clarke et al. 2008). On the other hand, signal temporal logic (STL) (Maler and Nickovic 2004) deals with dense-time real signals and continuous dynamical models (continuous and hybrid abstractions in Fig. 1a). In this domain, compatibility, consistency and refinement checks are performed on hybrid signals obtained from system simulation (Guo et al. 2014).

As a demonstration of our approach, we have developed the CHASE (Contract-based Hierarchical Analysis and System Exploration) environment, which builds upon a set of backend verification and synthesis tools, such as the NuSMV model checker
Cimatti et al. (2002) and the TuLiP toolbox (Wongpiromsarn et al. 2011), to reason about LTL A/G contracts. To facilitate this task, CHASE provides a syntax-constrained editor for engineers to express requirements in a structured form, using a set of predefined high-level primitives, or patterns, from which formal specifications can be automatically generated. This approach is similar to the one advocated within the European projects SPEEDS and CESAR (Damm et al. 2011), albeit linked to automata-based formalisms. CHASE provides effective tests to check for requirement consistency, i.e. whether a set of contracts is realizable, or whether, in contrast, facets of these are inherently conflicting, and thus no implementation is feasible. Moreover, it is possible to interactively exclude undesired behaviors in a design, e.g. by adding more contracts, by strengthening assumptions, or by considering additional cases for guarantees. Finally, it can incorporate higher-level domain-specific languages (DSLs), as proposed in (Nuzzo et al. 2014) and further exemplified in Sec. 4.1.

3.2 Platform Library Generation

In the bottom-up phase of the design process, a library of components (and contracts) is generated to model (or specify) both the plant and embedded system. As shown in Fig. 1 (a), components and contracts are hierarchically organized to represent the system at different levels of abstraction, e.g. steady-state, discrete-event (DE), and hybrid levels. Typically, at the highest levels of abstraction, a signal flow approach is more appropriate to CPS modeling, as is the case in signal processing, feedback control based on sensor outputs and actuator inputs, and in systems composed of unilateral devices (Willems 2007). In these cases, relations between system variables are better viewed in terms of inputs and outputs, and interconnections in terms of output-to-input assignments. Inputs are used to capture the influence of the environment on the system, while outputs are used to capture the influence of the system on the environment. At the lowest levels of abstraction, acausal models, without a-priori distinction between inputs and outputs, become more suitable to model the majority of physical (e.g. mechanical, electrical, hydraulic or thermal) components, which are generally governed by laws that merely impose relations (rather than functions) among system variables, and where interconnections mean that variables are shared (rather than assigned) among subsystems.

By reflecting the nature of the requirements, the component library is also viewpoint and domain dependent. At each level of abstraction, components are capable of exposing multiple, complementary viewpoints, associated with different design concerns and different formalisms (e.g. graphs, linear temporal logic, algebraic differential equations). Models include extra-functional (performance) metrics, such as timing, energy and cost, in addition to the description of their behaviors. A major challenge in multi-view and hierarchical modeling is to maintain consistency among different models and views, which are often developed using domain-specific frameworks, as the library evolves (Nuzzo and Sangiovanni-Vincentelli 2014).

To facilitate the integration of different domains within a unifying framework, (Shah et al. 2009) propose the customization of SysML by using profiles and DSLs to support multiple representations (or architectures) of the system, and graph transformations to describe the relations between them. We leverage the algebra of contract to incrementally check coherency or refinement among models as the library evolves. This information can then be stored in the library to speed up verification
tasks at design time (Iannopollo et al. 2014). Moreover, we use vertical contracts to establish conditions for an abstract, approximate model, to be a sound representation of a concrete model, i.e. to define when a model still retains enough precision to address specific design concerns, in spite of the vagueness required to make it manageable by analysis tools (Nuzzo et al. 2012).

### 3.3 Mapping Specifications to Implementations

In the absence of a unified framework for automated synthesis of cyber-physical systems simultaneously subject to a heterogeneous set of requirements, we propose to reason about different aspects or representations of the design by using specialized analysis and synthesis frameworks that can operate with different formalisms. During design space exploration, contracts can be used to define both the specification and the implementation platforms, thus playing a key role in checking or enforcing that an aggregation of components is compatible, and that the implementation is a correct refinement of the specification.

The mapping problem can be formulated and solved by either leveraging a pre-existing synthesis tool, or by casting an optimization problem that uses constraints from both the specification and the implementation levels to evaluate global tradeoffs among components. Accordingly, we denote as $C_{syn}$ a contract that can be used as input of a specialized synthesis tool, and as $C_{opt}$ a contract that serves as a conjunction of constraints in a more generic optimization problem. $C_{opt}$ can be further characterized as $C_{ver} \land C_{sim}$, where $C_{ver}$ denotes a contract whose satisfaction can be formally verified, while $C_{sim}$ refers to a contract that can only be checked by simulation.

For instance, in the design of the system architecture, $C_{A,syn}$ in Fig. 1 (a) includes the specification contract, expressed in terms of linear (or quadratic) arithmetic constraints on Boolean and real variables, as well as the steady-state models of the architecture, e.g. represented as constraints on a graph. Then, an implementation can be directly synthesized by solving a mixed integer-linear (or quadratic) program to minimize a cost function (e.g. component number, weight, cost, energy) while satisfying the constraints above. We have shown that such a formulation encompasses a variety of requirements, such as connectivity, safety, reliability, and energy balance.

To handle reliability requirements, we have proposed two algorithms to decrease the complexity of exhaustively enumerating all failure cases on all possible graph configurations, namely, Integer-Linear Programming Modulo Reliability (ILP-MR) and Integer-Linear Programming with Approximate Reliability (ILP-AR). ILP-MR lazily combines an ILP solver with a background exact reliability analysis routine. The solver iteratively provides candidate configurations that are analyzed and accordingly modified, only when needed, to satisfy the reliability requirements. Although exact reliability analysis is an NP-hard problem, the key idea is to perform it only when needed, i.e. a small number of times, and possibly on smaller graph instances. Conversely, ILP-AR eagerly generates a monolithic problem instance, albeit of a larger size, in polynomial time, using approximate reliability computations that can still provide estimations to the correct order of magnitude, and with an explicit theoretical bound on the approximation error (Nuzzo et al. 2014, Bajaj et al. 2015). We have implemented both the algorithms in the ARCHEX framework,
leveraging CPLEX (IBM 2012) as a backend optimization tool. The synthesized architecture can then serve as a specification for the control design step.

In control design, $C_{syn}$ in Fig. 1 (a) can be represented by LTL contracts, encoding a large set of requirements typically arising in safety-critical applications, together with the discrete event (DE) models of the plant components (also specified by LTL formulas). $C_{syn}$ can be manipulated by reactive synthesis tools (Piterman and Pnueli 2006, Wongpiromsarn et al. 2011) to generate control protocols in the form of one (or more) state machines, satisfying the requirements by construction. Analogously, other control synthesis techniques can be incorporated, such as the ones from supervisory control of discrete-event systems (Cassandras and Lafortune 2008), or model-predictive control (Raman et al. 2014). Sometimes, it is possible to incorporate timing information by using timed specification languages (e.g., timed computation tree logic) for which synthesis tools are available, e.g. UPPAAL-TIGA (Upfaal-TIGA). However, several real-time constraints, mostly related to the physical plant and the hardware implementation of the controller, may require the full expressiveness of continuous and hybrid models. These kinds of properties are then better assessed by monitoring simulation traces, since synthesis and formal verification result into intractable problems.

Let $C_{sim}$ be a contract that must be checked by simulation; then, given an array of costs $R$, the mapping problem, in its more general terms, can be cast as a multi-objective robust optimization problem, to find a set of platform configuration (design) parameter vectors that are Pareto optimal with respect to the objectives in $R$, while guaranteeing that the system satisfies the guarantees of $C_{sim}$ for all possible traces satisfying the environment assumptions of $C_{sim}$. $C_{sim}$ can also be parameterized, e.g. by leveraging parametric STL (Asarin et al. 2007) to capture degrees of freedom that are available in the system specification, and whose final values can also be determined as a result of the optimization process. A multi-objective optimization algorithm with simulation in the loop can then be implemented to find the Pareto optimal solutions. While this may be expensive in general, it becomes the only affordable approach in many practical cases (Nuzzo et al. 2014).

The simulation-based mapping methodology above can be used to perform joint design exploration of the controller and its execution platform, while guaranteeing that their specifications, captured by vertical contracts, are consistent. In fact, the controller requirements are typically defined in terms of several aspects that are related to the execution platform, including the timing behavior of the control tasks and of the communication between tasks, their jitter, the accuracy and resolution of the computation, and, more generally, requirements on power and resource consumption. These requirements are taken as assumptions by the controller, which in turn provides guarantees in terms of the amount of requested computation, activation times and data dependencies. We have recently demonstrated a design exploration methodology that supports the association of functionality to architectural services and uses the METRONOMY framework (Guo et al. 2014) to evaluate the characteristics (such as latency, throughput, power, and energy) of a particular implementation by simulation, and to verify the satisfaction of timing contracts.
4. Application Examples

We illustrate the application of the methodology discussed in this paper to system-level design of an aircraft electric power system (Nuzzo et al. 2014) and air management system, by briefly summarizing the main steps used to map the top-level requirements into a lower level representation of both the plant architecture and the control algorithm, to be further refined during subsequent design steps.

4.1 Aircraft Electric Power Distribution System

Fig. 2 shows a sample structure of an aircraft electric power system (EPS) in the form of a single-line diagram, a simplified notation for three-phase power systems (Moir and Seabridge 2008). Generators (denoted as GEN in Fig. 2) deliver power to loads (e.g. avionics, lighting, heating and motors, not represented in Fig. 2) via high-voltage and low-voltage AC (HVAC, LVAC) and DC buses (HVDC, LVDC), while Auxiliary Power Units (APU) or batteries (Batt) are used when one of the generators fails. Essential buses (labelled with ESS in Fig. 2) supply loads that cannot be unpowered for more than a predefined period $t_{\text{max}}$, while non-essential buses supply loads that may be shed in the case of a fault. Contactors are electromechanical switches that are opened or closed to determine the power flow from sources to loads, and are shown as double bars in the figure. AC transformers (ACT) convert high-voltage to low-voltage AC power. Rectifier Units (RUs) convert and route AC power to DC buses. Transformer Rectifier Units (TRUs) act both as transformers and rectifiers. The goal of the supervisory controller (not represented in Fig. 2) is to react to changes in system conditions or failures and reroute power by appropriately actuating the contactors, to ensure that essential buses are adequately powered.

4.1.1. Top-level Requirements. Top-level requirements are captured in terms of a system contract $C_S$ using an electric power system DSL, which enables automatic translation of the specifications from a set of pre-defined primitives to one (or more) of the following backend formalisms: arithmetic constraints on Boolean variables and failure probabilities (mixed integer-linear inequalities), LTL and STL, as shown in Fig. 1 (b). The proposed DSL can smoothly interface with pre-existing tools, such as visual programs for single-line diagrams, typically used by system engineers. Representative examples of system assumptions characterize the number and kind of component failures allowed, assuming that component failure events are all independent. Moreover, they prescribe that when a component fails during the flight, it will not come back online (e.g. primitive $\text{env}()$). On the other hand, examples of system guarantees include that no AC bus can be powered by multiple generators at the same time to avoid generator damage (e.g. primitive $\text{noparallel}()$), and that DC essential buses may stay unpowered for no longer than $t_{\text{max}}$ in case of failure (e.g. primitive $\text{essbus}()$). The above system requirements are used to derive a contract $C_T$ for the system architecture (in terms of mixed integer-linear inequalities), and a contract $C_C$ for the control algorithm (as a conjunction of LTL and STL contracts). Architecture and control protocol are consistently designed to satisfy $C_S$, which can be guaranteed by proving that $C_T$ and $C_C$ are compatible and their composition refines $C_S$. While we develop the proof without automatic support in (Nuzzo et al. 2014), this reasoning is still beneficial to co-design of architecture and control in a compositional way, i.e., to make sure that the design steps summarized below can be independently deployed, while guaranteeing that the assembled system is correct and satisfies $C_S$. 
4.1.2. Architecture Design. The plant architecture is modelled as a graph, where each node represents a component (with the exception of contactors, which are associated with edges) and each edge represents an interconnection. At this level of abstraction, the EPS platform library includes, as attributes, generator power ratings, component costs and failure probabilities, in addition to interconnection rules. The supervisory controller consists of one or more finite state machines, and is characterized by the reaction time $T_r$. Following the approach presented in Sec. 3, safety, connectivity, power flow and reliability requirements in $C_T$ (both assumptions and guarantees) are expressed as linear inequalities on a set of Boolean variables, each denoting the presence or absence of an interconnection in the topology graph. The tradeoffs between redundancy and cost can then be explored by using the ARCHEx framework; it is possible to generate, in a few minutes, architectures for the primary distribution of an EPS for different reliability requirements, and with up to 50 nodes. The optimally selected architecture, e.g. shown in Fig. 1(c), is offered as a specification for the control design step.

4.1.3. Control Design. The controller contract $C_C$ includes the allowable behaviors of its environment (including the physical plant) and the desired behaviors of the closed loop system (i.e. the top-level requirements), encoded as a conjunction of an LTL contract $C_{LTL}$ and an STL contract $C_{STL}$. $C_{LTL}$ is used together with DE models of the plant components (also described by LTL formulas) to synthesize a reactive control protocol in the form of one (or more) state machines, as shown in Fig. 3 (d), using reactive synthesis techniques. The resulting controller will then satisfy $C_{LTL}$ by construction. On the other hand, $C_{STL}$ captures design aspects related to the plant and the hardware implementation of the control algorithm, which cannot be expressed using the Boolean, untimed or DE abstractions offered by LTL. Its satisfaction is then assessed on a hybrid model, including both the controller and an acausal, equation-based representation of the plant, by monitoring simulation traces while optimizing a set of system parameters. The resulting optimal controller configuration is returned as the final design.
By using the GR(1) fragment of LTL to encode \( C_{LTL} \), and the topologies generated by ArchEx, a set of centralized and distributed control protocols could be automatically synthesized using TuLiP in approximately 0.5 to 2 s, for controllers with a number of states ranging from 4 to 113. The hybrid model was implemented in Simulink, based on blocks from the SymPowerSystem library, as shown in Fig. 3 (g), to analyze and optimize the real-time performance of the controller, imported as a MATLAB function.

The plant model includes the effects of non-ideal contactor response, implementing a fixed delay \( T_d \) to the open/close commands from the controller. It is then possible to explore the \( T_r \)-versus-\( T_d \) design space and find the maximum allowed controller reaction time \( T_{r,\text{max}} \) for a fixed \( T_{d,\text{fix}} \) in such a way that an essential DC bus is never out of range for more than \( t_{\text{max}} \). To do so, we cast an optimization problem, where the constraints are expressed as a conjunction of parametric STL formulas, and use the Breach toolbox (Donzé 2010) to facilitate post-processing of simulation traces and verify the satisfaction of the formulas. As an example, the red signal in Fig. 3 (e) denotes the amount of elapsed time while the DC bus voltage is out of range, i.e. the requirement on the DC bus is violated. If such a time is larger than \( t_{\text{max}} = 70 \text{ ms} \), the design is “unsafe” or incorrect. In Fig. 4 (f), the \( T_r \)-versus-\( T_d \) design space is explored in approximately 4 hours and the “safe” region for the designer to select the controller clock as a function of the contactor delay is marked in blue.

### 4.2 Aircraft Air Management System

Figure 3 (a) shows the simplified architecture of a Pressurization and Air Conditioning Kit (PACK) of an aircraft air management system (AMS). Engine bleed air enters the PACK through Valve 1, and gets partitioned into two flows, one passing through the bypass Valve 2, and the other passing through the heat exchanger (HX). Then, the flows recombine in a Mixer and enter the aircraft cabin. In a PACK, the cabin pressure is typically controlled by a set of electrical compressors (not shown in Fig. 3), while the cabin temperature is regulated via the HX and, possibly, expansion cooling across one (or more) turbines (Moir and Seabridge 2008). In our simplified diagram, Valve 1 is responsible for the flow-rate \( W_i \) into the system, whereas Valve 2 directly influences the cabin temperature by controlling the fraction of inflow that is cooled in the HX by the cold ambient mass flow \( W_a \). The AMS needs to be designed to supply sufficient pressure, and fresh oxygen, to the cabin at a comfortable temperature and humidity, while being resilient to faults, such as freezing or warping of critical components.

Differently than the EPS design of Sec. 4.1, in this example, we assume that the plant topology, hence its reliability, is fixed. On the other hand, the controller has a hierarchical structure, where a high-level supervisor decides the AMS operating mode (e.g. climbing, cruising) and provides the appropriate set points to the lower-level controllers, implemented using model predictive control or proportional-integral-derivative (PID) architectures. Therefore, our goal is to determine the plant sizing parameters and the control strategy for valves 1 and 2 and flow-rate \( W_a \) to satisfy a set of top-level requirements, under the assumptions that the temperature and pressure of the bleed air are piecewise constant (possibly changing over the duration of a flight) while the temperature of the cold air flowing into the heat exchanger is affected by the altitude of the airplane and the time of the day.
Since the PID controllers are right at the interface with the plant, we need to incorporate their effect as we determine the plant sizing, in order to enable independent implementation of plant architecture and control algorithm. Moreover, to accurately capture the plant dynamics, we resort to a continuous, nonlinear optimization scheme, where the system steady-state and transient performance is evaluated from simulation traces using an acausal, parameterized model of the plant in closed loop with its controllers, implemented in MODELICA. The system requirements include, among others, bounds on the desired cabin temperature (e.g. within 291 K and 298 K), the maximum time allowed before reaching the steady state, the minimum HX outlet temperature to avoid freezing.

In the architecture design step, top-level requirements are then translated into STL contracts and their satisfaction is checked on the simulation traces, while searching the space for an optimal plant sizing that minimizes the overall volume and component cost. In the control design step, for a given plant architecture, the same requirements can be used, together with a plant model based on ordinary differential equations, to synthesize both the DE supervisory control and the lower-level MPC schemes, using the techniques mentioned in Sec. 3. The result will be a fully sized MODELICA model of the plant, and a hybrid model of the overall controller. As an example, Fig. 3 (b) shows design exploration results capturing the impact of the HX geometry, i.e. cross-sectional area and length, on the system correctness at steady state. Fig. 3 (c) shows instead the feasible design space in a co-design scenario, when the steady state can be selected together with the plant sizing, thus making it possible to explore tradeoffs between plant sizing and control set points. In both the figures, the safe regions, where no requirements are violated and the system behaves correctly, are highlighted in green. Each exploration run, including 250 simulations, required

Figure 3. Simplified air management system architecture (figure from Mickelin et al. 2014) (a), and design exploration results: heat exchanger length versus cross-sectional area (b) and ambient flow rate (c).
approximately two minutes on an Intel Core i5 2.53-GHz processor with 4-GB memory.

5. Conclusions

In this paper, we presented a methodology for the design of complex cyber-physical systems originated by the iCyPhy consortium. The methodology supports a meet-in-the-middle process using the Platform-Based Design paradigm enriched with the use of A/G contracts to provide the formal framework that is necessary to ensure quality and correctness of complex designs. To document the application of the methodology, we used two concrete examples recommended by one of the partners of iCyPhy (UTC). Inspired by the design examples, we plan to push the methodology further to incorporate a design management feature that we call a frontend “orchestrator” routine, which directly interacts with the designer, helps coordinate the set of backend specialized tools, and consistently processes their results. For such an orchestrator to be developed, as a future research direction, we will continue developing tools to effectively guide designers towards requirement formalization, and algorithms that can leverage the modularity offered by contracts to perform compatibility, consistency and refinement checks on system portions of manageable size and complexity. Finally, we will continue developing algorithms for correct-by-construction control synthesis, to improve on their optimality, scalability and support for richer specification languages.

Acknowledgments. The authors wish to acknowledge Eelco Scholte (United Technologies Corporation (UTC)), Edward Lee (UCB), Amit Fisher (IBM), Clas Jacobson (UTC), John Finn (UCB), A. Iannopollo (UCB), Yilin Mo (Caltech), Jeff Ernst (UTC), and Earl Lavallee (UTC) for their collaboration in developing the design flow and the accompanying tools. This work was supported in part by IBM and UTC via the iCyPhy consortium and by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP), a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References


Biographies

Pierluigi Nuzzo received the M.Sc. degree in electrical engineering from the University of Pisa, Italy, in 2003. He is Ph.D. candidate in electrical engineering and computer sciences at the University of California at Berkeley. His research interests include methodologies and tools for the design of cyber-physical systems and mixed-signal integrated circuits. He held research positions at the University of Pisa and IMEC, Leuven, Belgium, working on the design of energy-efficient A/D converters and frequency synthesizers for reconfigurable radio. He has received the IBM Ph.D. Fellowship in 2012 and 2014.

Alberto Sangiovanni Vincentelli received the Laurea degree in electrical engineering and computer sciences from the Politecnico di Milano, Italy in 1971. He currently holds the Edgar L. and Harold H. Buttnert Chair of Electrical Engineering and Computer Sciences at the University of California at Berkeley. He was a co-founder of Cadence and Synopsys, he is the Chief Technology Adviser of Cadence, a member of the Board of Directors of Cadence and a member of the Science and Technology Advisory Board of General Motors. He is an author of over 880 papers and 15 books in the area of design tools and methodologies, large-scale systems, embedded systems, hybrid systems and innovation. He has won numerous awards, including, among others, the IEEE/RSJ Wolfson James Clerk Maxwell Award, the Kaufman Award and the ACM/IEEE Richard Newton Technical Impact Award in Electronic Design Automation.

Richard M. Murray received the B.S. degree in Electrical Engineering from California Institute of Technology in 1985 and the M.S. and Ph.D. degrees in Electrical Engineering and Computer Sciences from the University of California, Berkeley, in 1988 and 1991, respectively. He is currently the Thomas E. and Doris Everhart Professor of Control & Dynamical Systems and Bioengineering at Caltech. Murray's research is in the application of feedback and control to networked systems, with applications in biology and autonomy. Current projects include analysis and design of biomolecular feedback circuits; specification, design and synthesis of networked control systems; and novel architectures for control using slow computing.