A Contract-Based Methodology for Aircraft Electric Power System Design

Pierluigi Nuzzo, Huan Xu, Necmiye Ozay, John B. Finn,
Alberto L. Sangiovanni-Vincentelli, Richard M. Murray, Alexandre Donzé, Sanjit A. Seshia

***Abstract*—In an aircraft electric power system, one or more supervisory control units actuate a set of electromechanical switches to dynamically distribute power from generators to loads, while satisfying safety, reliability and real-time performance requirements. To reduce expensive re-design steps, this control problem is generally addressed by minor incremental changes on top of consolidated solutions. A more systematic approach is hindered by a lack of rigorous design methodologies that allow estimating the impact of earlier design decisions on the final implementation. To achieve an optimal implementation that satisfies a set of requirements, we propose a platform-based methodology for electric power system design, which enables independent implementation of system topology (i.e. interconnection among elements) and control protocol by using a compositional approach. In our flow, design space exploration is carried out as a sequence of refinement steps from the initial specification towards a final implementation by mapping higher-level behavioral and performance models into a set of either existing or virtual library components at the lower level of abstraction. Specifications are first expressed using the formalisms of linear temporal logic, signal temporal logic and arithmetic constraints on Boolean variables. To reason about different requirements, we use specialized analysis and synthesis frameworks and formulate assume-guarantee contracts at the articulation points in the design flow. We show the effectiveness of our approach on a proof-of-concept electric power system design.**

I. Introduction

The advent of high capability, reliable power electronics together with powerful embedded processors has enabled an increasing amount of "electrification" of vehicles such as cars and aircraft in recent years [1], [2]. Hydraulic, pneumatic and mechanical systems are being replaced by cyber-electrical components that increase the overall system efficiency [3]. However, the increased use of electrically-powered elements poses significant challenges to the aircraft electric power system in terms of the reliability of electrical power generation and distribution while satisfying safety requirements.

A severe limitation in common design practice is the lack of formalized specifications. System requirements are predominantly written in text-based languages that are not suitable for mathematical analysis and verification. Assessing system correctness is then left for simulations and prototype tests later in the design process, when modifications are significantly more expensive. Additionally, the inability to rigorously model the interactions among heterogeneous components and between the physical and the cyber sides of the system poses a serious obstacle. Thus, the traditional heuristic design process based on text-based requirement capture and designers' experience leads to implementations that are inefficient and sometimes do not even satisfy the requirements, yielding long re-design cycles, cost overruns and unacceptable delays.

We propose instead to carry out a rigorous design process that includes allocation of the requirements to the components and early validation of design constraints. By following the platform-based design paradigm [4], we proceed by subsequent refinement of design requirements using a library of available components. To perform this task, we define convenient abstractions for system exploration and compositional synthesis of system topology (interconnection among the various components) and control. In particular, we build a rich, multi-view set of component models that can be used by different, domain-specific analysis, synthesis and verification frameworks. We first synthesize an electric power system topology from system requirements formalized as arithmetic constraints on Boolean variables. For the given topology, we translate the requirements into temporal logic formulas, by which we synthesize and verify control protocols. To reason about different requirements in a compositional way, we use the concept of *contracts* [5] that formalize the notion of *interfaces* between models and tools in the design flow. A few theoretical results (Theorem III.1, Propositions VI.1 and VI.2) show how contracts can offer a natural framework to reason about distributed control architectures as well as the heterogeneous interface between the controller and its plant.

Our design methodology builds on a number of results that have opened the way for a more structured approach to the design of aircraft electric power systems. The adoption of model-based development and simulation for the analysis of aircraft performance and power optimization has already been advocated in [6], [7]. In the context of the More Open Electrical Technologies (MOET) project [2], a set of model libraries have been developed using the Modelica language [8] to support "more-electric" aircraft simulation, design and validation. Simulation is used for electric power system performance verification (e.g., stability and power quality) at the network level, by leveraging models with different levels of complexity to analyze different system properties, and validated with real

P. Nuzzo, J. B. Finn, A. L. Sangiovanni-Vincentelli, A. Donzé and S. A. Seshia are with the University of California at Berkeley, Department of Electrical Engineering and Computer Sciences, Cory Hall 1770, Berkeley, CA 94720. E-mail: nuzzo@eecs.berkeley.edu.

H. Xu is with the University of Maryland, Institute for Systems Research and Aerospace Engineering, College Park, MD.

N. Ozay is with the University of Michigan, Department of Electrical Engineering and Computer Science, 4229 EECS Building, Ann Arbor, MI 48109.

R. M. Murray is with the California Institute of Technology, Engineering and Applied Science Department, Pasadena, CA.

equipment measurements. However, design space exploration, optimization and analysis of faulty behaviors in these models can still become computationally unaffordable unless proper levels of abstraction are devised, based on the goals at each design step.

A library-based approach to instantiate, analyze and verify a system design was also adopted in [9], [10], within the META research program, with the aim to compress the product development and deployment timeline of defense systems. A simulation framework based on Modelica was developed to enable exploration of architectural design decisions, while a language based on SysML [11] was proposed to enable semantically robust integration of models, analytical methods and results provided by other domain specific languages and tools [12]. Such integration language incorporates assume-guarantee contracts to formalize system requirements and enable the generation of monitors. In this paper, we further extend the use of assume-guarantee contracts as a design aid in combination with platform-based design to yield system synthesis and optimization in addition to system simulation and verification.

An optimization-oriented power system design methodology following the platform-based paradigm was proposed in [13] where initial specifications are refined and mapped to the final implementation in four steps. At each step, a binary optimization problem is formulated to derive a class of candidate implementations for the next exploration step. The methodology deals with how to select the power generators and synthesize the electric power system topology. In this paper, we extend the flow in [13] to enable synthesis of electric power system topology and control, subject to heterogeneous sets of system requirements that are not always approximated by binary or mixed integer-linear constraints. To perform automatic synthesis of control protocols, we build on recent works on formal synthesis of aircraft vehicle management systems [14], distributed control synthesis [15], and reactive synthesis for electric power systems [16]. In particular, we express system specifications in linear temporal logic (LTL) [17], [18] and leverage a combination of tools from the computer science and formal methods domains.

The remainder of the paper is organized as follows. After a brief description of a typical electric power system and its design challenges in Section II, we provide some background on contract-based design and control synthesis in Section III. Section IV summarizes our electric power system design methodology while Section V and Section VI provide details on topology and control design. Section VII reports results from the application of our methodology to a prototype electric power system design, and is followed by concluding remarks in Section VIII.

II. The Aircraft Electric Power System

Figure 1 illustrates a sample architecture for power generation and distribution in a passenger aircraft in the form of a single-line diagram (SLD) [1], a simplified notation for three-phase power systems. Typically, aircraft electric power systems consist of generation, primary distribution and secondary distribution sub-systems. In this paper, we focus on the primary power distribution system, which includes the majority of the supervisory control logic.

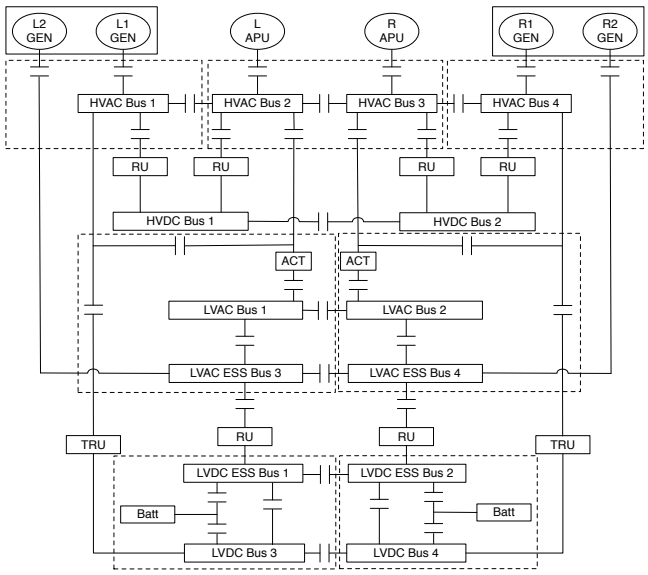


Fig. 1. Single-line diagram of an aircraft electric power system adapted from a Honeywell, Inc. patent [19].

A. Components

The main components of an electric power system are generators, contactors, buses, and loads. Primary *generators* are connected to the aircraft engine and can operate at high or low voltages. *Auxiliary generators* are mounted atop an *auxiliary power unit* (APU). The APU is normally used on ground (when no engines are available) to provide hydraulic and electric power, but can also be used in flight when one of the primary generators fails. With a small abuse of notation, we hereafter refer to auxiliary generators themselves as APUs. *Batteries* are primarily used at start-up and in case of emergency. AC and DC *buses* (both high and low-voltage) deliver power to a number of loads. Buses can be essential or non-essential. Essential buses supply loads that should always be powered, while non-essential ones supply loads that may be shed in the case of a fault or limited power capacity.

Contactors are electromechanical switches that connect components, and therefore determine the power flow from sources to loads. They are configured to be open or closed by one or multiple controllers (not shown in Fig. 1), denoted as *Bus Power Control Units* (BPCU).

Loads include subsystems such as lighting, heating, avionics and navigation. Bus loads also include power conversion devices: *Rectifier units* convert AC power to DC power, while AC *transformers* (ACTs) step down a high-voltage to a lower one, *Transformer Rectifier Units* (TRUs) both decrease the voltage level and convert it from AC to DC.

B. System Description

The main AC power sources at the top of Fig. 1 include two low-voltage generators, two high-voltage generators, and two APU-mounted auxiliary generators. Each engine connects to

a high-voltage AC (HVAC) generator (L1 and R1) and a low-voltage AC (LVAC) generator (L2 and R2). Panels, denoted as dashed square boxes, represent groups of components that are physically separated on the aircraft. The three panels below the generators include the HVAC buses, which can be selectively connected to the HVAC generators, to the auxiliary generators, and to each other via contactors, denoted by double bars.

Four rectifier units are selectively connected to buses as HVAC loads. The two panels below the high-voltage DC (HVDC) buses include the LVAC subsystem. A set of AC transformers (ACTs) convert HVAC power to LVAC power and are connected to four LVAC buses. LVAC ESS Bus 3 and LVAC ESS Bus 4 are essential and are selectively connected to the two low-voltage generators. The LVAC essential buses are also connected to rectifier units, and thus to low-voltage DC (LVDC) power. The LVDC subsystem also contains two batteries. Power can be selectively routed directly from the HVAC bus to the LVDC buses 3 and 4 using TRUs.

One or more bus power control units use sensors (which are not depicted in Fig. 1) to measure physical quantities, such as voltages and currents, and control the state (open or closed) of the contactors, to dynamically reconfigure the system based on the status and availability of the power sources. For the rest of the paper, we denote this centralized or distributed supervisory control unit as BPCU.

C. System Requirements

Given a set of loads, together with their power and reliability requirements, the goal is to determine the system's architecture and control such that the demand of the loads is satisfied for all flight conditions and a set of predetermined faults. To better formalize this design objective, we begin with a qualitative analysis of the main system requirements, by categorizing them in terms of safety and reliability requirements. For each of these categories, we provide a few examples that serve as a reference for the rest of the paper.

Safety specifications constrain the way each bus must be powered to avoid loss of essential features, and the maximum time interval allowed for power shortages. For instance, to avoid generator damage, we proscribe AC sources to be paralleled, i.e. no AC bus can be powered by multiple generators at the same time. Moreover, we refine the definition of essential loads and buses (such as flight-critical actuators) provided above by requiring that they be never unpowered for more than a specified time t_{max}.

Reliability specifications describe the bounds on the failure probabilities that can be tolerated for different portions of the system. Based on its failure modes, every component is characterized by a failure rate. A failure rate of λ indicates that a failure occurs, on average, every $1/\lambda$ hours. For a given mission profile, failure rates can be translated into failure probabilities so that system reliability specifications are also expressed in terms of the failure probabilities of the components. Based on the component failure rates, a typical specification would require that the failure probability for an essential load (i.e., the probability of being unpowered for longer than t_{max}) be smaller than 10^{-9} per flight hour. The actual probability value depends on the load criticality [1]. In our example, both the electric power system topology and the controller should be designed to accommodate any possible combination of faults potentially causing the failure of an essential component, and having a joint probability larger than 10^{-9} per flight hour.

III. Contract-Based Design of Cyber-Physical Systems

Inspired by recent results on assume-guarantee compositional reasoning and interface theories in the context of hybrid systems and software verification, our methodology is based on the use of assume-guarantee contracts for cyber-physical systems [5]. Informally, contracts mimic the thought process of a designer, who aims at *guaranteeing* certain performance figures for the design under specific *assumptions* on its environment. The essence of contracts is, therefore, a *compositional* approach, where design and verification complexity is reduced by decomposing system-level tasks into more manageable subproblems at the component level, under a set of assumptions. System properties can then be inferred or proved based on component properties. In this respect, contract-based design can be a rigorous and effective paradigm while dealing with the complexity of modern system design, and has been successfully applied to other embedded system domains, such as automotive applications [20] and mixed-signal integrated circuits [21].

A. Components

We summarize the main concepts behind contract-based design starting with the notion of components. A summary of the notation used in this Section and in the rest of the paper is given in Table I.

A *component* $\mathcal{M}$ can be seen as an abstraction, a hierarchical entity representing an element of a design, characterized by the following *component attributes*:

- a set of input *variables* $U \in \mathcal{U}$, output variables $Y \in \mathcal{Y}$, and internal variables (including state variables) $X \in \mathcal{X}$; a set of configuration *parameters* $\kappa \in \mathcal{K}$, and a set of input, output and bidirectional *ports* $\lambda \in \Lambda$ for connections with other components;
- a set of *behaviors*, which can be implicitly represented by a dynamic *behavioral model* $\mathcal{F}(U, Y, X, \kappa) = 0$, uniquely determining the value of the output and internal variables given the one of the input variables and configuration parameters. We assume that components can respond to every possible sequence of input variables, i.e., they are receptive to their input variables. Behaviors are generic, and could be continuous functions that result from solving differential equations, or sequences of values or events recognized by an automata model;
- a set of *non-functional models*, i.e. maps that allow computing non-functional properties of a component corresponding to particular valuations of its input variables and configuration parameters. Examples of non-functional maps include the *performance model* $\mathcal{P}$, computing a set of performance figures by solving the behavioral

model, or the *reliability model* $\mathcal{R}$, providing the failure probability of a component.

Components can be connected together by sharing certain ports under constraints on the values of certain variables. In what follows, we use *variables* to denote both component variables and ports. Moreover, components can be hierarchically organized to represent a system at different levels of abstraction. Given a set of components at level *l*, a system can then be composed by *parallel composition* and represented as a new component at level *l+1*. At each level of abstraction, components are also capable of exposing multiple, complementary *views*, associated to different concerns (e.g. safety, performance, and reliability), which can be expressed via different formalisms and analyzed by different tools.

A component may be associated to both implementations and contracts. An *implementation* M is an instantiation of a component $\mathcal{M}$ for a given set of configuration parameters. In what follows, we also denote with M the set of all its behaviors.

B. Contracts

A *contract* $\mathcal{C}$ for a component $\mathcal{M}$ is a pair of assertions (A, G), called the *assumptions* and the *guarantees*. An *assertion* H represents a specific set of behaviors over variables that satisfies H. Therefore, operations on assertions and contracts are set operations. An implementation M satisfies an assertion H whenever M and H are defined over the same set of variables and all the behaviors of M satisfy the assertion, i.e. when $M \subseteq H$. The set of all the legal *environments* for $\mathcal{C}$ collects all implementations E such that $E \subseteq A$. An implementation of a component satisfies a contract whenever it satisfies its guarantee, subject to the assumption. Formally, $M \cap A \subseteq G$, where M and $\mathcal{C}$ have the same variables. We denote such a *satisfaction* relation by writing $M \models \mathcal{C}$. Similarly, we relate a legal environment E to a contract $\mathcal{C}$ by the satisfaction relation $E \models_E \mathcal{C}$.

Any implementation M of a component such that $M \subseteq G \cup \neg A$, where $\neg A$ is the complement of A, is also an implementation for $\mathcal{C}$. In general, $M_{\mathcal{C}} = G \cup \neg A$ is the maximal implementation for $\mathcal{C}$. Two contracts $\mathcal{C}$ and $\mathcal{C}'$ with identical variables, identical assumptions, and such that $G' \cup \neg A = G \cup \neg A$, possess identical sets of implementations. Such two contracts are then *equivalent*. Therefore, any contract $\mathcal{C} = (A, G)$ is equivalent to a contract in *saturated form* (A, G'), which also satisfies $G' \supseteq \neg A$, or, equivalently, $G' \cup A = True$, the true assertion. To obtain the saturated form of a contract, it is enough to take $G' = G \cup \neg A$.

Contracts associated with different components can be combined according to different rules. Similar to parallel composition of components, *parallel composition* of contracts can be used to construct composite contracts out of simpler ones. Let $\mathcal{C}_1 = (A_1, G_1)$ and $\mathcal{C}_2 = (A_2, G_2)$ be contracts in saturated form, then the assumptions and the guarantees of the composite $\mathcal{C}_1 \otimes \mathcal{C}_2$ can be computed as follows [20]:

$$A = (A_1 \cap A_2) \cup \neg(G_1 \cap G_2), \tag{1}$$

$$G = G_1 \cap G_2. \tag{2}$$

The composite contract must clearly satisfy the guarantees of both. Moreover, since the environment should satisfy all the assumptions, we should expect that the assumptions of each contract would also combine by conjunction. In general, however, part of the assumptions A_1 will be already satisfied by composing $\mathcal{C}_1$ with $\mathcal{C}_2$, which acts as a partial environment for $\mathcal{C}_1$. Therefore, G_2 can relax the assumptions A_1, and vice-versa, which motivates equation (1). To use equation (1) and equation (2), the behaviors related to the original contracts need to be extended to a common set of variables. Such an extension, which is also called *alphabet equalization*, can be achieved by an operation of inverse projection [20].

Even if they need to be satisfied simultaneously, multiple views of the same component do not generally compose by parallel composition. Therefore, the *conjunction* ($\wedge$) of contracts can also be defined so that if $M \models \mathcal{C}_1 \wedge \mathcal{C}_2$, then $M \models \mathcal{C}_1$ and $M \models \mathcal{C}_2$. Contract conjunction can be computed by defining a preorder on contracts, which formalizes a notion of *refinement*. We say that $\mathcal{C}$ *refines* $\mathcal{C}'$, written $\mathcal{C} \preceq \mathcal{C}'$ (with $\mathcal{C}$ and $\mathcal{C}'$ both in saturated form), if $A \supseteq A'$ and $G \subseteq G'$. Refinement amounts to relaxing assumptions and reinforcing guarantees, therefore strengthening the contract. Clearly, if $M \models \mathcal{C}$ and $\mathcal{C} \preceq \mathcal{C}'$, then $M \models \mathcal{C}'$. On the other hand, if $E \models_E \mathcal{C}'$, then $E \models_E \mathcal{C}$. With the given ordering, we can compute the conjunction of contracts by taking the greatest lower bound of $\mathcal{C}_1$ and $\mathcal{C}_2$. For contracts in saturated form, we have

$$C_1 \wedge C_2 = (A_1 \cup A_2, G_1 \cap G_2), \tag{3}$$

i.e. *conjunction* of contracts amounts to taking the intersection of the guarantees and the union of the assumptions. Conjunction can be used to compute the overall contract for a component starting from the contracts related to multiple views (concerns, requirements) in a design.

In addition to satisfaction and refinement, *consistency* and *compatibility* are also relations involving contracts. Technically, these two notions refer to individual contracts. A contract is *consistent* when the set of implementations satisfying it is not empty, i.e. it is feasible to develop implementations for it. For contracts in saturated form, this amounts to verifying that $G \neq \emptyset$. $\mathcal{C}$ is *compatible* if there exists a legal environment E for $\mathcal{C}$, i.e. if and only if $A \neq \emptyset$. The intent is that a component satisfying contract $\mathcal{C}$ can only be used in the context of a compatible environment. In practice, however, violations of consistency and compatibility occur as a result of a parallel composition, so that we can refer to the collection of components forming a composite contract as being consistent or compatible.

C. Platform-Based Design and Contracts

We use contracts in the context of platform-based design [4], a paradigm that allows reasoning about design in a structured way. In platform-based design, design progresses in precisely defined abstraction levels; at each level, functionality (what the system is supposed to do) is strictly separated from architecture (how the functionality can be implemented). Differently than model-based development, platform-based design consists

of a *meet-in-the-middle* approach where successive top-down refinements of high-level specifications across design layers are mapped onto bottom-up abstractions and characterizations of potential implementations. Each layer is defined by a design *platform*, which is a *library* (collection) of *components*, models, representing functionality and performance of the components (as detailed in Section III-A), and *composition rules*.

In this context, contracts can play a fundamental role in: (i) determining *valid* compositions so that when the design space is explored, only *legal* (i.e. satisfying the composition rules) compositions that are *compatible* (i.e. satisfying the contracts) are taken into consideration; (ii) guaranteeing that a component at a higher level of *abstraction* is an accurate representation of a lower level component (or aggregation of components); (iii) checking that an architecture platform is indeed a correct *refinement* of a specification platform, and (iv) formalizing top-level system *requirements*.

Since compatibility is assessed among components at the same abstraction layer, the first category of contracts is denoted as *horizontal contracts*. If an environment violates a horizontal contract, it cannot host any of its implementations.

However, checking horizontal contracts is not sufficient, in general, to guarantee correct implementations. When analyzing the behavior of complex cyber-physical systems, simplified macro-models can be used to capture the relevant behavior of the components at higher levels of abstraction. Therefore, guarantees should also be provided on the accuracy of the macro-models with respect to models at lower levels of abstraction. These guarantees are captured via *bottom-up vertical contracts*. On the other hand, vertical contracts can also be used to encode top-down requirements that system architects introduce to craft the behavior of a chosen architecture according to the desired functionality. The above set of constraints can be expressed using *top-down vertical contracts*. They are used to ensure that an implementation is correct, by checking that the architecture platform is a refinement of the specification platform.

To partition system specifications, we identify which entity is responsible for a set of requirements, and which ones are just indirectly affected. By assigning information about requirements to components, we make it explicit what each component guarantees and what it assumes about its environment. Both of these aspects determine the top-down vertical contract for the component. If the assumptions are satisfied, then the component specification can be developed independently of other subsystems. In Section IV, we exploit this concept to independently develop the electric power system topology and its control protocol.

To formulate system and component requirements as contracts, we adopt different formalisms based on the computational models used to represent the components and the tools used to analyze and synthesize them. Example of formalisms include automata or temporal logic constructs (e.g. used for safety requirements), probabilistic constraints (e.g. used for reliability requirements), linear arithmetic constraints on Boolean variables (e.g. used for connectivity requirements), integro-differential equations, and linear or nonlinear constraints on real numbers (e.g. used for real-time requirements). In what follows, we review the formalisms adopted in Section VI for the analysis and synthesis of reactive controllers in a contract-based framework.

D. Requirement Formalization

We use two formal specification languages, namely, linear temporal logic (LTL) and signal temporal logic (STL), particularly suitable for capturing system and component requirements and reasoning about the correctness of their behaviors. As such, these languages will be used for defining contracts for control design.

1) Linear Temporal Logic: Temporal logic is a branch of logic that incorporates temporal aspects in order to reason about propositions in time, and was first used as a specification language by Pnueli [22]. In this section, we consider a version of temporal logic called linear temporal logic (LTL), whose formal semantics can be found in [23]. While in contract-based design the component is regarded as the fundamental element of a design, and systems are denoted as interconnections of components, as we describe the basics of LTL, we prefer to adhere to the classical terminology, which is historically consolidated [23], and define design abstractions in terms of systems.

Definition 1: A *system* consists of a set $\mathcal{S}$ of variables. The domain of $\mathcal{S}$, denoted by $dom(\mathcal{S})$, is the set of valuations of $\mathcal{S}$.

Definition 2: An *atomic proposition* is a statement on system variables that has a unique truth value (*True* or *False*) for a given value s. Let $s \in dom(\mathcal{S})$ be a state of the system (i.e., a specific valuation of its variables) and p be an atomic proposition. Then $s \models p$ if p is *True* at the state s. Otherwise, $s \not\models p$.

LTL also includes Boolean connectors such as negation ($\neg$), disjunction ($\vee$), conjunction ($\wedge$), material implication ($\rightarrow$), and two basic temporal modalities, *next* ($\bigcirc$) and *until* ($\mathcal{U}$). By combining these operators, it is possible to specify a wide range of requirements. Given a set AP of atomic propositions, LTL formulas are formed according to the following grammar:

$$\varphi := True \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \mathcal{U} \varphi_2$$

where $p \in \mathrm{AP}$. Formulas involving other operators, including *eventually* ($\Diamond$) and *always* ($\Box$), can be derived from these basic ones.

LTL formulas over AP are interpreted over infinite sequences of states. In the LTL abstraction, we denote such a sequence as a *behavior* of the system. Let $\sigma = s_0 s_1 s_2 \ldots$ be a behavior and φ be an LTL formula. We say that φ holds at position $i \geq 0$ of σ, written $s_i \models \varphi$, if and only if φ holds for the remainder of the sequence starting at position i. Then, a sequence σ satisfies φ, denoted by $\sigma \models \varphi$, if $s_0 \models \varphi$. Let Σ be the collection of all sequences σ such that $\sigma \in \Sigma$. Then, a system composed of the variables $\mathcal{S}$ is said to satisfy φ, written $\Sigma \models \varphi$, if all sequences satisfy φ.

2) Signal Temporal Logic: LTL allows formal reasoning about temporal behaviors of systems with Boolean, discrete-time signals (variables) or sequences of events. To deal with

dense-time real signals and hybrid dynamical model that mix the discrete dynamics of the controller with the continuous dynamics of the plant, several logics have been introduced over the years, such as Timed Propositional Temporal Logic [24], and Metric Temporal Logic [25]. Signal Temporal Logic (STL) [26] has been proposed more recently as a specification language for constraints on real-valued signals in the context of analog and mixed-signal circuits. In this paper, we refine LTL system requirements into constraints on physical variables (e.g. voltages and currents) expressed using STL constructs. Then, we monitor and process simulation traces to verify constraint satisfaction, while optimizing a set of design parameters.

For a hybrid dynamical model, we define a *signal* as a function mapping the time domain $\mathbb{T} = \mathbb{R}_{\geq 0}$ to the reals $\mathbb{R}$. A multi-dimensional signal $\boldsymbol{q}$ is then a function from $\mathbb{T}$ to $\mathbb{R}^n$ such that $\forall t \in \mathbb{T}$, $\boldsymbol{q}(t) = (q_1(t), \cdots, q_n(t))$, where $q_i(t)$ is the i-th component of vector $\boldsymbol{q}(t)$. It is convenient to represent the behavior of the system's variables over time using multi-dimensional signals. Therefore, we assume that a hybrid system behavioral model $\mathcal{F}$ (e.g. implemented in a simulator) takes as input a signal $\boldsymbol{u}(t)$ and computes an output signal $\boldsymbol{y}(t)$ and an internal signal $\boldsymbol{x}(t)$ such that $\mathcal{F}(\boldsymbol{u}(t), \boldsymbol{y}(t), \boldsymbol{x}(t), \boldsymbol{\kappa}) = 0$, where $\boldsymbol{\kappa}$ is a given vector of system configuration parameters. A collection of signals resulting from a simulation of the system is a *trace*, which can also be viewed as a multi-dimensional signal. A trace $\boldsymbol{s}(t)$ that includes all the system input, output and internal signals can also denote a system *behavior*.

In STL, constraints on real-valued signals, or *predicates*, can be reduced to the form $\mu = g(\boldsymbol{q}) \sim \pi$, where g is a scalar-valued function over the signal $\boldsymbol{q}$, $\sim \in \{<, \leq, \geq, >, =, \neq\}$, and π is a real number. As in LTL, temporal formulas are formed using temporal operators, *always*, *eventually* and *until*. However, each temporal operator is indexed by intervals of the form (a, b), $(a, b]$, $[a, b)$, $[a, b]$, (a, ∞) or $[a, \infty)$, where each of a, b is a non-negative real-valued constant. If I is an interval, then an STL formula is written using the following grammar:

$$\varphi := True \mid \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2$$

The *always* and *eventually* operators are defined as special cases of the *until* operator as follows: $\Box_I \varphi \triangleq \neg \Diamond_I \neg \varphi$, $\Diamond_I \varphi \triangleq True \mathcal{U}_I \varphi$. When the interval I is omitted, we use the default interval of $[0, +\infty)$.

The semantics of STL formulas are defined informally as follows. The signal $\boldsymbol{q}$ satisfies $\mu = g(\boldsymbol{q}) < 2$ at time t (where $t \geq 0$), written $(\boldsymbol{q}, t) \models \mu$, if $g(\boldsymbol{q}(t)) < 2$. It satisfies $\varphi = \Box_{[0,2)}\ (q > -1)$, written $(\boldsymbol{q}, t) \models \varphi$, if for all time $0 \leq t < 2$, $q(t) > -1$. The signal q_1 satisfies $\varphi = \Diamond_{[1,2)} q_1 > 0.4$ iff there exists time t such that $1 \leq t < 2$ and $q_1(t) > 0.4$. The two-dimensional signal $\boldsymbol{q} = (q_1, q_2)$ satisfies the formula $\varphi = (q_1 > 10)\ \mathcal{U}_{[2.3,4.5]}\ (q_2 < 1)$ iff there is some time t_0 where $2.3 \leq t_0 \leq 4.5$ and $q_2(t_0) < 1$, and for all time t in $[2.3, t_0)$, $q_1(t)$ is greater than 10. We write $\boldsymbol{q} \models \varphi$ as a shorthand of $(\boldsymbol{q}, 0) \models \varphi$. Formal semantics can be found in [26].

Parametric Signal Temporal Logic (PSTL) is an extension of STL introduced in [27] to define *template formulas* containing unknown parameters. Syntactically speaking, a PSTL formula is an STL formula where numeric constants, either in the constraints given by the predicates μ or in the time intervals of the temporal operators, can be replaced by symbolic parameters. These parameters are divided into two types:

- A *scale* parameter π is a parameter appearing in predicates of the form $\mu = g(\boldsymbol{q}) \sim \pi$,
- A *time* parameter τ is a parameter appearing in an interval of a temporal operator.

An STL formula is obtained by pairing a PSTL formula with a valuation function that assigns a value to each symbolic parameter. For example, consider the PSTL formula $\varphi(\pi, \tau) = \Box_{[0,\tau]} q > \pi$, with symbolic parameters π (scale) and τ (time). The STL formula $\Box_{[0,10]} q > 1.2$ is an instance of φ obtained with the valuation $w = \{\tau \mapsto 10,\ \pi \mapsto 1.2\}$.

TABLE I
NOTATION

	Platform Components and Contracts
Symbol	Definition
$\mathcal{M}$	Generic platform component
U	Input variable set
$\mathcal{U}$	Input variable domain
Y	Output variable set
$\mathcal{Y}$	Output variable domain
X	Internal (and state) variable set
$\mathcal{X}$	Internal variable domain
$\mathcal{S} = U \cup Y \cup X$	Component (system) variable set
$\mathcal{S}$, $dom(\mathcal{S})$	Component (system) variable domain
κ	Configuration parameter set
$\mathcal{K}$	Configuration parameter domain
λ	Port set
Λ	Port domain
$\mathcal{F}(U, Y, X, \kappa) = 0$	Behavioral model
$\mathcal{P}$	Performance model (map)
$\mathcal{R}$	Reliability model (map)
M	Implementation (and set of its behaviors)
$\mathcal{C} = (A, G)$	Contract (assumptions, guarantees)
E	Set of legal environments for $\mathcal{C}$
	Discrete Event (LTL) Abstraction
Symbol	Definition
Σ	Generic system (and set of its behaviors)
$\mathcal{S}$	System variable set
$\mathcal{S}$, $dom(\mathcal{S})$	System variable domain
s	System state
$\sigma = s_0 s_1 s_2 \ldots$	System behavior
$\mathcal{E}$	Environment variable set
$\mathcal{E}$, $dom(\mathcal{E})$	Environment variable domain
e	Environment variable valuation
$\mathcal{D}$	Controlled variable set
$\mathcal{D}$, $dom(\mathcal{D})$	Controlled variable domain
d	Controlled variable valuation
$\mathcal{C}_{LTL} = (\varphi_e, \varphi_e \rightarrow \varphi_s)$	LTL contract
	Hybrid Model (STL) Abstraction
Symbol	Definition
$\boldsymbol{u}(t)$	Input signal
$\boldsymbol{y}(t)$	Output signal
$\boldsymbol{x}(t)$	Internal (state) signal
$\boldsymbol{s} = (\boldsymbol{u}, \boldsymbol{y}, \boldsymbol{x})$	System trace or behavior
$\boldsymbol{\kappa}$	Configuration parameter vector
$\mathcal{F}(\boldsymbol{u}, \boldsymbol{y}, \boldsymbol{x}, \boldsymbol{\kappa}) = 0$	Behavioral model
$\mathcal{C}_{STL} = (\varphi_e, \varphi_e \rightarrow \varphi_s)$	STL contract
$\mathcal{C}'_{LTL} = (\varphi'_e, \varphi'_e \rightarrow \varphi'_s)$	LTL contract refined into STL

E. Reactive Control Synthesis

Reactive systems are systems that maintain an ongoing relation with their environment by appropriately reacting to it. The controllers that regulate the behavior of such systems are called *reactive controllers*.

A *control system* is a composition of a physical plant, including sensors and actuators (e.g., an electric power system topology with fault sensors and contactors), and an embedded controller that runs a control protocol (control logic) to restrict the behaviors of the plant so that all the remaining behaviors satisfy a set of system specifications. System specifications can be expressed as a contract $\mathcal{C} = (A, G)$, where, roughly speaking, assumptions A encode the allowable behaviors of the environment the control system operates in, and guarantees G encode the system requirements.

The synthesis of reactive controls can then be interpreted in terms of assume-guarantee contracts. Given the system contract $\mathcal{C}$, control synthesis finds a control logic that, when implemented, ensures that the system satisfies $\mathcal{C}$; or declares that no such logic exists. It is possible to extend this idea to distributed control architectures. In distributed synthesis, different control subsystems can be composed if their contracts are compatible. Hence, the goal of distributed synthesis is to simultaneously refine a system contract into compatible horizontal contracts for the components (i.e., subsystems), and to find the control logics that realize those contracts.

1) Reactive Synthesis from LTL Specifications: Let $\mathcal{E}$ and $\mathcal{D}$ be sets of environment and controlled variables, respectively. Let $s = (e, d) \in dom(\mathcal{E}) \times dom(\mathcal{D})$ be a state of the system. Consider an LTL specification φ of assume-guarantee form

$$\varphi = (\varphi_e \rightarrow \varphi_s), \tag{4}$$

where φ_e characterizes the assumptions on the environment and φ_s characterizes the system requirements. The synthesis problem is concerned with constructing a control protocol (a partial function $f : (s_0 s_1 \dots s_{t-1}, e_t) \mapsto d_t$) which chooses the move of the controlled variables based on the state sequence so far and the behavior of the environment so that the system satisfies φ_s as long as the environment satisfies φ_e. If such a protocol exists, the specification φ is said to be *realizable*. Reactive synthesis can then be viewed as a two-player game between an environment that attempts to falsify the specification in equation (4) and a controlled plant that tries to satisfy it.

For general LTL, the synthesis problem has a doubly exponential complexity [28]. However, a subset of LTL, namely generalized reactivity (1) (GR(1)), generates problems that can be solved in polynomial time (i.e., polynomial in $|dom(\mathcal{E}) \times dom(\mathcal{D})|$, the number of valuations of the variables in $\mathcal{E}$ and $\mathcal{D}$) [29]. GR(1) specifications restrict φ_e and φ_s to take the following form, for $\alpha \in \{e, s\}$,

$$\varphi_\alpha := \varphi^\alpha_{\text{init}} \wedge \bigwedge_{i \in I_1^\alpha} \Box \varphi^\alpha_{1,i} \wedge \bigwedge_{i \in I_2^\alpha} \Box \Diamond \varphi^\alpha_{2,i},$$

where $\varphi^\alpha_{\text{init}}$ is a propositional formula characterizing the initial conditions; $\varphi^\alpha_{1,i}$ are transition relations characterizing safe, allowable moves and propositional formulas characterizing invariants; $\varphi^\alpha_{2,i}$ are propositional formulas characterizing states that should be attained infinitely often; I_1^α and I_2^α are index sets enumerating formulas $\varphi^\alpha_{1,i}$ and $\varphi^\alpha_{2,i}$, respectively.

Given a GR(1) specification, there are game solvers and digital design synthesis tools that generate a finite-state automaton that represents the control protocol for the system [30], [31].

2) Distributed Synthesis: To provide an inherent level of redundancy for system reliability, distributed control architectures are increasingly being adopted in modern aircraft electric power systems, thus motivating the extension of reactive synthesis techniques to the design of distributed controllers. Given a global specification and a system composed of subsystems, distributed synthesis proceeds by first finding local specifications for each subsystem, and then synthesizing local controllers for these subsystems separately. If the local specifications satisfy certain conditions, it can be shown that the local controllers realizing these local specifications can be implemented together and the overall system is guaranteed to satisfy the global specification, as detailed in [15]. We describe below a special case of distributed architecture, i.e. a serial interconnection of controllers, which is used in the design in Section VII-B2 to synthesize controllers for AC and DC subsystems separately. The following theorem is based on a result that was first reported in [15]. We introduce here a new proof that shows how contracts can offer a rigorous and effective framework to reason about distributed control architectures in a compositional manner.

Theorem III.1. *Given*

- *a system characterized by a set $\mathcal{S} = \mathcal{D} \cup \mathcal{E}$ of variables, where $\mathcal{D}$ and $\mathcal{E}$ are disjoint sets of controllable and environment variables,*
- *its two subsystems with variables $\mathcal{S}_1 = \mathcal{D}_1 \cup \mathcal{E}_1$ and $\mathcal{S}_2 = \mathcal{D}_2 \cup \mathcal{E}_2$, where for each $i \in \{1, 2\}$, $\mathcal{D}_i$ and $\mathcal{E}_i$ are disjoint sets of controllable and environment variables for the i^{th} subsystem, $\mathcal{D}_1$ and $\mathcal{D}_2$ are disjoint, and $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2$,*
- *a set $\mathcal{I}$ of pairs of variables representing the interconnection structure, that is, for a serial interconnection, $\mathcal{I} = \{(o_1, i_2) | o_1 \in O_1 \subseteq (\mathcal{D}_1 \cup \mathcal{E}_1), i_2 \in I_2 \subseteq \mathcal{E}_2\}$, where for all $(o, i) \in \mathcal{I}$, $o = i$,*
- *a global specification $\varphi : \varphi_e \rightarrow \varphi_s$, and two local specifications $\varphi_1 : \varphi_{e_1} \rightarrow \varphi_{s_1}$ and $\varphi_2 : \varphi_{e_2} \rightarrow \varphi_{s_2}$, where φ_e, φ_{e_1}, φ_{e_2}, φ_s, φ_{s_1}, and φ_{s_2} are LTL formulas containing variables only from their respective sets of environment variables $\mathcal{E}$, $\mathcal{E}_1$, $\mathcal{E}_2$ and system variables $\mathcal{S}$, $\mathcal{S}_1$, $\mathcal{S}_2$;*

if the following conditions hold:

1) *any behavior that satisfies φ_e also satisfies $(\varphi_{e_1} \wedge \varphi_{e_2})$,*
2) *any behavior that satisfies $(\varphi_{s_1} \wedge \varphi_{s_2})$ also satisfies φ_s,*
3) *there exist two controllers that make the local specifications $(\varphi_{e_1} \rightarrow \varphi_{s_1})$ and $(\varphi_{e_2} \rightarrow \varphi_{s_2})$ true under the interconnection structure $\mathcal{I}$;*

then, implementing the two controller together leads to a controller that satisfies the global specification $\varphi_e \rightarrow \varphi_s$.

Proof. The conditions on $\mathcal{D}$, $\mathcal{D}_1$, $\mathcal{D}_2$ ensure that the two controllers are composable, i.e. they do not try to control the same output (controllable) variables. We first derive contracts from global and local specifications, by defining the following

sets of behaviors in terms of assumptions and guarantees:

$$A = \{\sigma : \sigma \models \varphi_e\}; \quad A_i = \{\sigma : \sigma \models \varphi_{e_i}\};$$
$$G = \{\sigma : \sigma \models (\varphi_e \rightarrow \varphi_s)\}; \quad G_i = \{\sigma : \sigma \models (\varphi_{e_i} \rightarrow \varphi_{s_i})\};$$
$$A' = \{\sigma : \sigma \models (\varphi_{e_1} \wedge \varphi_{e_2})\};$$
$$G' = \{\sigma : \sigma \models ((\varphi_{e_1} \wedge \varphi_{e_2}) \rightarrow (\varphi_{s_1} \wedge \varphi_{s_2}))\}.$$

We immediately observe that $A' = A_1 \cap A_2$ while $G' \supseteq (G_1 \cap G_2)$. Now, let $\mathcal{C} = (A, G)$ be the global contract and $\mathcal{C}_1 = (A_1, G_1)$, $\mathcal{C}_2 = (A_2, G_2)$ the local contracts, all in saturated form. Clearly, for any implementation M_i, $M_i \models \mathcal{C}_i$ if and only if its set of behaviors $\sigma_{M_i} \subseteq G_i$, i.e. $\sigma_{M_i} \models \varphi_i$, after alphabet equalization. Moreover, because any implementations M_1 and M_2 of $\mathcal{C}_1$ and $\mathcal{C}_2$ are composable, contract composition using equations (1) and (2) is well defined and the composition $M_1 \times M_2$ (under the interconnection $\mathcal{I}$) is an implementation of $\mathcal{C}_1 \otimes \mathcal{C}_2$ (under the interconnection $\mathcal{I}$).

We now prove that

$$\mathcal{C}_1 \otimes \mathcal{C}_2 \preceq \mathcal{C},$$

i.e., $\mathcal{C}_1 \otimes \mathcal{C}_2 = (A_{12}, G_{12})$ refines $\mathcal{C}$. By the definition of refinement, this amounts to showing that $G_{12} \subseteq G$ and $A_{12} \supseteq A$. We obtain

$$G_{12} = (G_1 \cap G_2) \subseteq G' \subseteq G \tag{5}$$

by conditions 1 and 2 in the theorem statement, and

$$A_{12} = (A_1 \cap A_2 \cup \neg G_{12}) \supseteq (A_1 \cap A_2) = A' \supseteq A, \tag{6}$$

by condition 1. Moreover, if $\mathcal{C}$ is compatible (i.e. A is not empty), $\mathcal{C}_1 \otimes \mathcal{C}_2$ will also be compatible (i.e. A_{12} is not empty) by (6). Equations (5) and (6) allow us to conclude that $\mathcal{C}_1 \otimes \mathcal{C}_2$ is well defined and refines $\mathcal{C}$, hence for any implementations M_1 and M_2 of $\mathcal{C}_1$ and $\mathcal{C}_2$, $M_1 \times M_2$ satisfies the global specification. □

There are two sources of conservatism in distributed synthesis. The first one is due to the fact that local controllers have only local information. Therefore, even if there exists a centralized controller that realizes a global specification, there may not exist local controllers that do so. This is an inherent problem and can only be addressed by modifying the control architecture (e.g., by changing the mapping of controlled variables to controllers, by introducing new sensors, or by modifying the information flow between local controllers). The second source of conservatism is computational. Even when local controllers that realize the global specification exist, it might be difficult to find them (e.g., see [28] for some undecidability results). We note that the conditions provided in Theorem III.1 are only sufficient conditions. The choices of φ_{e_j} and φ_{s_j} for $j \in \{1, 2\}$ plays a role in the level of conservatism. In principle, φ_{e_j} and φ_{s_j} should be chosen such that A' is as "small" as possible, and G' is as "large" as possible in the sense of set inclusion. Hence, when conditions 1 and 2 are satisfied but condition 3 is not satisfied, one can gradually refine the local specifications. See [15] for further details and an example of such a refinement.

F. Design Space Exploration and Performance Optimization

Several real-time performance requirements (e.g. timing constraints), mostly relating to the dynamic behaviors of the physical plant and the hardware implementation of the control algorithm, are better assessed on hybrid dynamical models. For this purpose, we refine a subset of LTL requirements into STL constructs on physical (e.g. electrical, mechanical) quantities and leverage off-line or on-line monitoring techniques while optimizing the system.

A Boolean verdict on whether a property is satisfied may not be sufficient for design space exploration and system optimization. In fact, we are also interested in capturing the *robustness of satisfaction* of a formula φ by a signal $\boldsymbol{q}$, i.e., the amount of margin by which a property is satisfied. To do so, we refer to the *quantitative semantics* of STL. The quantitative semantics of STL are defined using a real-valued function ρ of a trace $\boldsymbol{q}$, a formula φ, and time t satisfying the following property:

$$\rho(\varphi, \boldsymbol{q}, t) \geq 0 \text{ iff } (\boldsymbol{q}, t) \models \varphi. \tag{7}$$

The underlying idea is that, whenever the absolute value of $\rho(\varphi, \boldsymbol{q}, t)$ is large, a change in $\boldsymbol{q}$ is less likely to affect the Boolean satisfaction (or violation) of φ by $\boldsymbol{q}$, i.e. the margin by which a design satisfies φ is larger.

Without loss of generality, an STL predicate μ can be identified to an inequality of the form $g(\boldsymbol{q}) \geq 0$ (the use of strict or non strict inequalities is a matter of choice and other inequalities can be trivially transformed into this form). From this form, a straightforward quantitative semantics for predicate μ is defined as

$$\rho(\mu, \boldsymbol{q}, t) = g(\boldsymbol{q}(t)). \tag{8}$$

Then ρ can be inductively defined for every STL formula using the following rules:

$$\rho(\neg\varphi, \boldsymbol{q}, t) = -\rho(\varphi, \boldsymbol{q}, t) \tag{9}$$
$$\rho(\varphi_1 \wedge \varphi_2, \boldsymbol{q}, t) = \min(\rho(\varphi_1, \boldsymbol{q}, t), \rho(\varphi_2, \boldsymbol{q}, t)) \tag{10}$$
$$\rho(\varphi_1 \mathcal{U}_I \varphi_2, \boldsymbol{q}, t) = \sup_{t' \in t+I} \Big[\min \big(\rho(\varphi_2, \boldsymbol{q}, t'), \inf_{t'' \in [t, t')} \rho(\varphi_1, \boldsymbol{q}, t'') \big) \Big]. \tag{11}$$

Additionally, by combining equation (11), and $\Box_I \varphi \triangleq \neg \Diamond_I \neg \varphi$, we get

$$\rho(\Box_I \varphi, \boldsymbol{q}, t) = \inf_{t' \in t+I} \rho(\varphi, \boldsymbol{q}, t') \tag{12}$$

Finally, for $\Diamond$, we get a similar expression using $\sup$ instead of $\inf$. It can be shown that ρ, as defined above, satisfies equation (7) and thus defines a quantitative semantics for STL [32].

By leveraging such quantitative semantics, a design space exploration problem on a hybrid system model defined as in Section III-D2 can be formulated as follows. Let $\mathcal{C}_{STL} = (\varphi_e, \varphi_s)$ be an STL contract encoding a set of system requirements, with φ_e and φ_s PSTL formulas. Let $\boldsymbol{C}$ be an array of costs, and $\boldsymbol{\kappa} \in \mathcal{K}$ a vector of platform configuration parameters, i.e., a vector of variables in the hybrid system

model that are selected as a result of the design process. Our goal is to find a set of parameter vectors $\boldsymbol{\kappa}^*$ that are Pareto optimal with respect to the objectives in $\boldsymbol{C}$, while guaranteeing that the system satisfies φ_s for all possible system traces $\boldsymbol{s} \in \mathcal{S}$ satisfying the environment assumptions φ_e. Examples of design parameters could be the controller clock or a tunable delay in a component.

To formalize the above multi-objective optimization problem, we partition φ_s as

$$\varphi_s(\boldsymbol{\tau}, \boldsymbol{\pi}) = \varphi_{sc}(\boldsymbol{\tau}, \boldsymbol{\pi}) \wedge \bigwedge_{i=1}^{m} \varphi_{sr,i}(\boldsymbol{\tau}, \boldsymbol{\pi}), \tag{13}$$

where a set of time parameters $\boldsymbol{\tau} \in \mathcal{T}$ and scale parameters $\boldsymbol{\pi} \in \mathit{\Pi}$ can be used to capture degrees of freedom that are available in the system specifications, and whose final value can also be determined as a result of the optimization process. The formula φ_{sc} in (13) encodes the requirements that will be considered as "hard" optimization constraints for Boolean satisfaction, while $\varphi_{sr,i}$ are formulas that will also be considered for robust satisfaction, i.e., given a system trace $\boldsymbol{s'}$ and a parameter set $(\boldsymbol{\tau'}, \boldsymbol{\pi'})$, the robust satisfaction $\rho_i(\varphi_{sr,i}(\boldsymbol{\tau'}, \boldsymbol{\pi'}), \boldsymbol{s'}, 0)$ will also be computed. Similarly, the array of costs $\boldsymbol{C}$ can be partitioned as follows

$$\boldsymbol{C}(\boldsymbol{\kappa}, \boldsymbol{\tau}, \boldsymbol{\pi}) = \left(\boldsymbol{C}_c(\boldsymbol{\kappa}, \boldsymbol{\tau}, \boldsymbol{\pi}), C_i(\rho_i(\varphi_{s,ri}(\boldsymbol{\tau}, \boldsymbol{\pi}), \boldsymbol{s}(\boldsymbol{\kappa}), 0)) \Big|_{1 \le i \le m} \right), \tag{14}$$

where $\boldsymbol{C}_c(\boldsymbol{\kappa}, \boldsymbol{\tau}, \boldsymbol{\pi})$ is a vector of costs that depend only on the parameters of the model and the formulas; it can be used to capture, for instance, some performance figures (e.g., bandwidth, energy) as a function of the system design parameters, or the duration of a requirement violation. Each component $C_i(\rho_i(\varphi_{sr,i}, \boldsymbol{s}, 0))$ in (14) is instead a scalar function of the quantitative satisfaction of each formula $\varphi_{sr,i}$; it can be used to capture and maximize the margin by which $\varphi_{sr,i}$ is satisfied.

By putting it all together, the design exploration problem can be expressed as a multi-objective robust optimization problem

$$\min_{\boldsymbol{\kappa} \in \mathcal{K}, \boldsymbol{\tau} \in \mathcal{T}, \boldsymbol{\pi} \in \mathit{\Pi}} \quad \boldsymbol{C}(\boldsymbol{\kappa}, \boldsymbol{\tau}, \boldsymbol{\pi}) \tag{15}$$
$$\text{s.t.} \begin{cases} \mathcal{F}(\boldsymbol{s}, \boldsymbol{\kappa}) = 0 \\ \\ \boldsymbol{s} \models \varphi_s(\boldsymbol{\tau}, \boldsymbol{\pi}) \qquad \forall \boldsymbol{s} \quad \text{s.t.} \quad \boldsymbol{s} \models \varphi_e \end{cases}$$

where we aim to minimize a set of costs over all possible system and formula parameter valuations, for all the system behaviors satisfying the behavioral model and the contract $\mathcal{C}_{STL}$. For a given parameter valuation $\boldsymbol{\kappa'}$, $\boldsymbol{s'} = (\boldsymbol{u'}, \boldsymbol{y'}, \boldsymbol{x'})$ is the trace of input, output and internal signals that are obtained by simulating $\mathcal{F}(.)$. A multi-objective optimization algorithm with simulation in the loop can then be used to find the Pareto optimal solutions $\boldsymbol{\kappa}^*$. While this may be expensive in general, it becomes affordable in many practical cases, as will be shown in Section VI and Section VII.

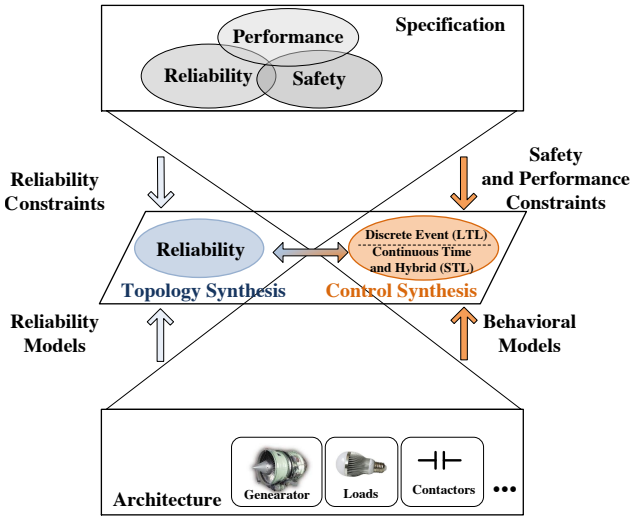


Fig. 2. Pictorial representation of the main steps in the electric power system design flow in Fig. 3.

IV. Platform-Based Flow for Electric Power System Design Using Contracts

Our design flow, pictorially represented in Fig. 2, consists of two main steps, namely, topology design and control design. The *topology design* step instantiates electric power system components and connections among them to generate an optimal topology while guaranteeing the desired reliability level. Given this topology, the BPCU state machine can then be synthesized in the *control design* phase to actuate contactors while guaranteeing that loads are correctly powered. The above two steps are, however, connected. The correctness of the controller needs to be enforced in conjunction with its boundary conditions, i.e., the assumptions on the entities that are not controlled, yet interact with it. An example of such an assumption is the number of paths from generators to a load made available by the electric power system architecture to the controller. Similarly, the reliability of an architecture must be assessed under the assumptions that the controller adequately configures the contactors to leverage the available paths. Therefore, to achieve independent implementation of architecture and controller, we address the synthesis problem in a compositional way, by using contracts to incorporate the information on the environment conditions under which each entity is expected to operate.

Our design process includes a top-down and a bottom-up phase. In the top-down phase, we associate the requirements to the different entities in the system and formulate top-down vertical contracts for them. In the bottom-up phase, we populate the library of architecture components including, for instance, generators, buses, power converters and contactors. Each component is characterized by its attributes, including multiple models or views, such as behavioral or reliability views, and finite state machine or continuous-time models, as detailed in Section III-A. Horizontal contracts specify legal compositions between components. Bottom-up vertical contracts define under which conditions a model is a faithful representation of a physical element in the system. In what

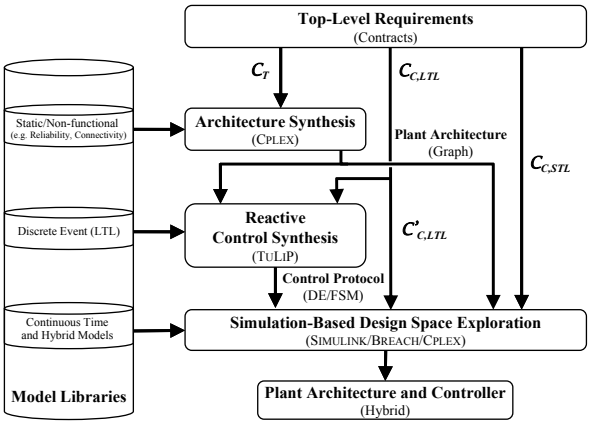


Fig. 3. Electric power system architecture and control design flow and tool chain.

follows, we provide details on the electric power system design space exploration.

A. Design Space Exploration

There is currently no automated procedure for optimal synthesis of control protocols simultaneously subject to reliability, safety and real-time performance constraints. Therefore, we reason about these three aspects of the design by using specialized analysis and synthesis frameworks that operate with different formalisms. Contracts specifying the interface between components and views help transfer requirements between different frameworks and verify correctness with respect to the full set of requirements. As also shown in Fig. 3, our design space exploration is organized as follows:

a) From system requirements, we generate a set of requirements for the electric power system architecture (denoted as a contract $\mathcal{C}_T$ in Fig. 3). Safety, connectivity and power flow requirements are expressed as arithmetic constraints on Boolean variables (mixed integer-linear inequalities); reliability constraints are inequalities on real numbers involving component failure probabilities. The trade-off between redundancy and cost can then be explored and an electric power system topology is synthesized to minimize the total component cost while satisfying the constraints above. The synthesized topology serves as a specification (assumption) for the subsequent control design step.

b) A subset of the original high-level system specifications are translated into LTL formulas for the topology generated in a) (contract $\mathcal{C}_{C,LTL}$ in Fig. 3). Using the results in Section III-E, a reactive control protocol is then synthesized from LTL constructs and made available as one (or more) state machines, satisfying safety and reliability specifications by construction. However, several architectural and real-time constraints (e.g. timing) related to the physical plant and the hardware implementation of the control algorithm are not available at this level of abstraction. Approaches to incorporate timing within reactive control synthesis, by using timed specification languages (e.g., timed computation tree logic) and related synthesis tools (e.g., UPPAAL-Tiga [33]), are currently under investigation. In this work, timing constraints are handled at a lower abstraction level, as detailed below.

c) The architecture in a) and the controller in b) are executed using continuous-time or hybrid behavioral models to assess satisfaction of (some of) the requirements at a lower abstraction level (contract $\mathcal{C}_{C,STL}$ in Fig. 3). The LTL requirements from b) are also refined into STL formulas (contract $\mathcal{C}'_{C,LTL}$ in Fig. 3). Simulation traces are monitored to verify and optimize the controller using the approach detailed in Section III-F. As an example, an optimal reaction period can be selected in the presence of delays in the switches and under the assumption of a synchronous controller implementation. The resulting architecture and controller pair is then returned as the final design.

We provide details on both topology and control synthesis in Section V and Section VI, including sufficient conditions for their co-design, while guaranteeing that top-level requirements for the controlled system are satisfied.

V. Electric Power System Topology Design

We cast the topology design problem as a mixed integer-linear optimization problem. Our goal is to derive an electric power system architecture that satisfies a set of connectivity, power flow and reliability requirements, while minimizing cost and complexity (i.e. number of components) of the overall network.

The electric power system architecture is modelled as a directed graph $\mathcal{G} = (V, E)$, where each node $v_i \in V$ represents a component (with the exception of contactors, which are associated with edges) and each edge $e_{ij} \in E$ represents the interconnection between v_i and v_j $(i, j \in \{1, \ldots, n\})$. Therefore, the set of Boolean variables $\{e_{ij}\}$, each denoting the presence or absence of an interconnection, are the decision variables for our optimization problem. While connectivity and power flow requirements generate constraints that are linear in the decision variables, or can be straightforwardly linearized, the situation is different for reliability constraints. A reliability constraint prescribes that the failure probability of a critical load, i.e. the probability that a load stays unpowered longer than specified because of failures, should be less than a desired threshold. As further discussed in Section V-B, evaluating such a failure probability produces high-order polynomial inequalities in terms of the decision variables. Such constraints would either call for a nonlinear solver or for several symbolic manipulations and linearization techniques, possibly involving large sets of auxiliary variables. Therefore, instead of formulating a single, "flat" optimization problem, we propose an iterative algorithm inspired by the *mixed integer-linear programming modulo theory* approach [34], [35], summarized in Algorithm 1.

The topology design algorithm receives as inputs: (part of) the electric power system platform library $\mathcal{L}$, including generator power ratings $\boldsymbol{g}$, component costs $\boldsymbol{w}$ and failure probabilities $\boldsymbol{P}$; a topology template $\mathcal{T}$ with the maximum

Algorithm 1 Topology Design

Input: Topology template $\mathcal{T}$, arrays of generator power ratings $\boldsymbol{g}$, component failure probabilities $\boldsymbol{P}$ and costs $\boldsymbol{w}$, required reliability r^*, set of connectivity and power requirements $\boldsymbol{R}$
Output: Topology $\mathcal{G}$

```
while r ≥ r* do                                   ▷ failure probability
    [Cost, Cons]=FORMMILP(T, w, g, R)
    G=SOLVE(Cost, Cons)
    [r, R_new] = RELANALYSIS(G, P)
    R = ADDCONST(R, R_new)                        ▷ add new constraints
end while
```

number of allowed components for each category and their composition rules; the set of requirements, including connectivity constraints, load power and reliability requirements. Reliability requirements are generally specified at critical loads or essential buses; to simplify, in Algorithm 1, we assume that an overall system reliability requirement r^* is provided, as defined in Section V-B.

A mixed integer-linear program (MILP) generates minimum cost topologies for the given set of connectivity and power flow constraints. The MILP is solved in a loop with a reliability analysis algorithm, which receives as input a candidate topology, evaluates the failure probability of critical loads and implements strategies to improve the reliability, by providing additional constraints for the MILP, until all requirements are satisfied.

The contract for the topology design step can then be expressed as a pair $\mathcal{C}_T = (A_T, G_T)$, where A_T represents the set of topology graphs that conform to the template $\mathcal{T}$ and are labelled with the generator power ratings $\boldsymbol{g}$, the component costs $\boldsymbol{w}$ and failure probabilities $\boldsymbol{P}$. G_T represents the topology graphs that satisfy the load reliability requirements and power requirements (in nominal conditions). Both A_T and G_T can be concretely expressed using mixed integer-linear or nonlinear constraints (originated from probability computations) in the decision variables and the graph model parameters. In what follows, we detail the two key components of our synthesis flow, namely the MILP formulation function FORMMILP and the reliability analysis function RELANALYSIS.

A. Mixed Integer-Linear Program Formulation

FORMMILP formulates the optimization problem by assuming an initial graph template $\mathcal{T}$ for the electric power system topology, comprising a maximal number of virtual nodes and edges together with their composition rules. Some nodes and edges are activated during an optimization run to generate a candidate topology. The others remain inactive or can be used in subsequent optimization runs to provide redundant paths, by increasing or reconfiguring the electric power system interconnections until all the reliability requirements are satisfied. The candidate topology resulting from an optimization step is a minimal topology, in which unnecessary nodes and edges are pruned away to minimize the overall network cost, while satisfying a set of connectivity and power flow constraints.

To simplify our notation, we partition the adjacency matrix of $\mathcal{G}$ into smaller blocks to represent interconnections between subsets of components, as summarized in Table II. For instance, the interconnections between n_{gen} generators and n_{acb} AC buses can be represented by a $n_{gen} \times n_{acb}$ connectivity sub-matrix denoted as $\mathbf{M}^{gb}$. We further assume that any interconnection (edge) between two components is associated to a contactor. Relaxing this assumption entails minor modifications in our formulation to handle contactors as separate nodes in $\mathcal{G}$.

TABLE II
CONNECTIVITY SUB-MATRICES

Variables	Interconnection	Dimension
$\mathbf{M}^{gb}$	Generator - AC Buses	$n_{gen} \times n_{acb}$
$\mathbf{M}^{bb}$	AC Buses - AC Buses	$n_{acb} \times n_{acb}$
$\mathbf{M}^{br}$	AC Buses - Rectifiers	$n_{acb} \times n_{rec}$
$\mathbf{M}^{rd}$	Rectifiers - DC Buses	$n_{rec} \times n_{dcb}$
$\mathbf{M}^{dd}$	DC Buses - DC Buses	$n_{dcb} \times n_{dcb}$
$\mathbf{M}^{dl}$	DC Buses - Loads	$n_{dcb} \times n_{load}$

The cost function is the sum of the costs of all components (associated with the nodes) and contactors (associated with the edges) used in the electric power system architecture, i.e.

$$\sum_{i=1}^{|V|} \delta_i w_i + \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} e_{ij} \tilde{w}_{ij} \tag{16}$$

where $|V|$ is the number of nodes, w_i is the cost of component i, $\tilde{w}_{ij}$ is the cost of contactor on edge e_{ij} and δ_i is a binary variable equal to one if the component is instantiated in a topology and zero otherwise.

All components and paths in the electric power system need to obey the composition rules in our library. In particular, *connectivity* constraints enforce legal connections among components and are formalized as arithmetic constraints on the Boolean decision variables. As an example, we prescribe that any DC load must be directly connected to only one DC bus as follows:

$$\sum_{i=1}^{n_{dcb}} M^{dl}_{i,j} = 1 \quad \forall\ j \in \mathbb{N},\ j \in [1, n_{load}].$$

Moreover, all DC buses that are connected to the network (e.g. to a load or another DC bus) must be connected to at least one TRU to receive power from an AC bus i.e. $\forall\ j \in \mathbb{N}, j \in [1, n_{dcb}]$

$$\sum_{i=1}^{n_{rec}} M^{rd}_{i,j} \geq \sum_{i=1}^{n_{load}} M^{dl}_{j,i}, \quad \sum_{i=1}^{n_{rec}} M^{rd}_{i,j} \geq \sum_{i=1}^{n_{dcb}} M^{dd}_{j,i}.$$

All TRUs that are connected to a DC bus must be connected to at least one AC bus, i.e. $\forall\ j \in \mathbb{N},\ j \in [1, n_{rec}]$

$$\sum_{i=1}^{n_{acb}} M^{br}_{i,j} \geq \sum_{i=1}^{n_{dcb}} M^{rd}_{j,i}.$$

Similarly, all AC buses that are connected to a TRU or another AC bus must be connected to one generator, i.e. $\forall\ j \in \mathbb{N},\ j \in [1, n_{acb}]$

$$\sum_{i=1}^{n_{gen}} M^{gb}_{i,j} \geq \sum_{i=1}^{n_{rec}} M^{br}_{j,i}, \quad \sum_{i=1}^{n_{gen}} M^{gb}_{i,j} \geq \sum_{i=1}^{n_{acb}} M^{bb}_{j,i},$$

while a rectifier cannot be directly connected to more than one DC bus and to more than one AC bus, i.e. $\forall\ j \in \mathbb{N},\ j \in [1, n_{rec}]$

$$\sum_{i=1}^{n_{dcb}} M_{j,i}^{rd} \leq 1, \quad \sum_{i=1}^{n_{acb}} M_{i,j}^{br} \leq 1.$$

Power-flow constraints are used to enforce 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 non-sheddable loads on both sides of the system.

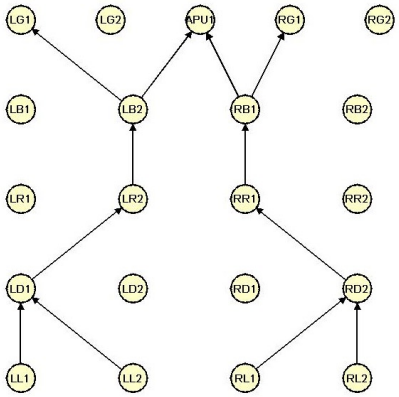


Fig. 4. Directed graph representation of an electric power system architecture. Unconnected nodes represent virtual components.

B. Reliability Analysis

As discussed in Section III-A, every library component is characterized by a reliability model estimating the failure probability during its operation. Experimental data on the failure rates of the physical components (e.g. contactors, generators, buses) have been collected over the years and made available in the literature. Failure rates can be related to probabilities as follows. We assume that the time at which a component can fail is a random variable with an exponential distribution, whose parameter λ is the failure rate [36]. Therefore, the probability that a failure is observed in a time interval T can be computed as $P_{fail} = 1 - \mathrm{e}^{-\lambda T}$. The objective of the reliability analysis function RELANALYSIS is then to compute the probability of composite events of failure in the system, starting from the failure probabilities of its components. We denote as overall *system failure* F an event in which there is no possibility for any of the available generators to deliver power to a critical load or an essential bus. Therefore, the overall system failure probability r, also denoted as *reliability level*, is defined as

$$r = \mathbb{P}(F) = \mathbb{P}\left(\bigcup_{k=1}^{m} F_k\right), \tag{17}$$

where F_k is a failure event at the critical load (or essential bus) j, and m is the total number of critical loads (or essential buses). We assume that when a component fails, it is no longer possible to deliver power from and through that component, i.e. the component becomes an open circuit in the schematic and cannot be recovered. Moreover, failures in different components are considered as independent.

To compute the reliability at a critical load, we adopt an extension of traditional fault tree analysis (FTA) that supports hierarchical composition, similar to the approach in [37]. Besides handling decomposition with respect to the hierarchy of failure influences, our formulation is able to compute failure probabilities directly from the electric power system topology. Our assumption is that the reliability level of an electric power system can be statically determined by its topological structure and the redundancy of the paths used to power a critical load.

To compute the event F_i of a system failure at component i, we first convert the original electric power system graph $\mathcal{G} = (V, E)$ into a directed graph $\mathcal{G}'$. An edge is directed from v_i to v_j if v_i receives power by (or through) v_j when traversing the graph from a critical load to a generator. An example of such a graph $\mathcal{G}'$ is shown in Fig. 4. Let $\mathbf{A}$ be the adjacency matrix for $\mathcal{G}'$, P_i be the event that component i fails (self-induced failure), and let $p_i = \mathbb{P}(P_i)$. Then, the event F_i of a system failure at component i can be recursively computed as follows

$$F_i = P_i \cup \left(\bigcap_{\substack{j=1 \\ A_{i,j} \neq 0}}^{n} F_j \right) \tag{18}$$

where $A_{i,j}$ is i^{th}-row, j^{th}-column element of $\mathbf{A}$. In other words, component i ceases to be powered when either a failure is generated by itself, or when failures are induced in all its neighbor nodes. We denote as neighbors only those nodes through which i can actually receive power.

When $\mathcal{G}'$ is a tree, computing the failure probability for a critical load i is straightforward. The tree is traversed from the critical load (the root of the tree) to the generators (the leaves of the tree), and the probability of failure at node i can be directly derived from equation (18) as

$$\mathbb{P}(F_i) = \begin{cases} p_i & \text{if } A_{i,j} = 0\ \forall j \\ p_i + (1 - p_i) \prod_{j=1}^{n} [\mathbb{P}(F_j)]^{A_{i,j}} & \text{otherwise.} \end{cases} \tag{19}$$

If $\mathcal{G}'$ includes cycles, failure probabilities of critical loads can still be computed by traversing the graph using a similar procedure as above, as sketched by the recursive implementation in Algorithm 2.

To compute the failure probability at a critical load i, COMPRELIABILITY stores in $\boldsymbol{L}$ all the neighbor nodes of i that have not been visited yet (provided by the function UNVNEIGH). Then, COMPRELIABILITY generates all possible combinations of failure events due to components in $\boldsymbol{L}$ (provided by the function GENEVENT) and compute their probability, by multiplying the contributions due to independent components and summing up the contributions due to disjoint events. Whenever one (or more) components are healthy (EXISTSHEALTHY

Algorithm 2 COMPRELIABILITY

Input: A directed graph $\mathcal{G}'$, an array of component failure probabilities $\boldsymbol{P}$, an array of currently visited nodes $\boldsymbol{C}$ (at which failure probabilities must be computed), an array of previously visited nodes $\boldsymbol{W}$
Output: $\boldsymbol{probFail}$, array of probabilities of all failure events induced by the neighbors of the nodes in $\boldsymbol{C}$

```
probFail ← [ ]
L ← UNVNEIGH(C, W)        ▷ unvisited neighbors of nodes in C
W ← [W, L]                ▷ update visited nodes

if ISEMPTY(L) then
    probFail ← 1          ▷ return the neutral element
else
    for all event in GENEVENT(L) do        ▷ all failure events
        fail ← 1
        for all k in L do
            if ISFAILING(k, event) then
                fail ← fail * P(k)         ▷ component fails
            else
                if ISGENERATOR(k) then
                    fail ← 0               ▷ healthy generator
                else
                    fail ← fail * (1 − P(k))
                end if
                C ← [C, k]
            end if
        end for
        if EXISTSHEALTHY(event) then       ▷ not all fail
            fail ← fail * SUM(COMPRELIABILITY(P, C, W))
        end if
        probFail ← [probFail, fail]
        C ← [ ]
    end for
end if
```

returns one), COMPRELIABILITY is recursively called by using such healthy components as current nodes. Recursion stops when either a healthy generator is found or $\boldsymbol{L}$ is empty. In the first case, there exists a path of healthy components from a generator to a critical load and, therefore, its failure probability is zero. Otherwise, a "healthy" path is found which does not include a generator; its contribution to the overall failure probability is then irrelevant and COMPRELIABILITY returns the neutral element 1.

At each iteration, if the optimal architecture satisfies the reliability constraints, it is returned as the final solution. Otherwise, RELANALYSIS estimates the number of 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 connectivity constraints. Such strategies are subsequently deployed until the target failure probability is reached. As a first strategy, the number of available paths is increased by introducing additional interconnections (and contactors) between the right and the left side buses of the system, where critical loads are connected. Afterwards, redundancy in the DC and AC buses on each side of the topology is increased. Finally, redundant components may be added if available, whenever they are compatible with other cost or weight constraints.

As an example of constraints generated to improve reliability, we enforce that the number of connections between left-side and right-side DC buses be incremented by adding

$$\sum_{i=1}^{n_{dcb}^{L}} \sum_{j=1}^{n_{dcb}^{R}} M_{i,j}^{d,LR,new} \geq 1 + \sum_{i=1}^{n_{dcb}^{L}} \sum_{j=1}^{n_{dcb}^{R}} M_{i,j}^{d,LR,old}. \tag{20}$$

However, if a right-side DC bus is connected to a left-side DC bus then it should also be connected to a right-side DC bus or load, i.e.,

$$M_{i,j}^{d,LR,new} \leq \max\{\max_{k} M_{j,k}^{dl,R}, \max_{k} M_{j,k}^{dd,R}\} \tag{21}$$

$\forall\ i \in \mathbb{N}, i \in [1, n_{dcb}^{L}], \forall\ j \in \mathbb{N}, j \in [1, n_{dcb}^{R}]$. Moreover, if a left-side DC bus is connected to a right-side DC bus, then it should also be connected to a rectifier, i.e.,

$$M_{i,j}^{d,LR,new} \leq \max_{k} M_{k,i}^{rd,L} \tag{22}$$

$\forall\ i \in \mathbb{N}, i \in [1, n_{dcb}^{L}], \forall\ j \in \mathbb{N}, j \in [1, n_{dcb}^{R}]$. The superscripts R and L in equation (21) and equation (22) denote left and right-side matrices, $M^{d,LR}$ is the left-right DC bus connectivity matrix, n_{dcb}^{L} and n_{dcb}^{R} are the number of left and right DC buses, respectively. Equations (20)-(22) are encoded and added to the optimization constraints for the next iteration every time the number of connections between left-side and right-side DC buses must be incremented. Similar constraints must also hold after replacing R with L, and vice-versa, in equations (21) and (22).

VI. ELECTRIC POWER SYSTEM CONTROLLER DESIGN

Power requirements of different loads might differ in an aircraft based on the mode of operation. Similarly, the availability of the generators and the health conditions of several components might vary during the flight. The goal of the BPCU (i.e., controller) is to reconfigure the electric power system and reroute power by appropriately reacting to such changes in system conditions to ensure that safety-critical loads are always powered. In this section we first describe how the control logic for the BPCU can be automatically synthesized within the proposed framework. Then, we present a domain specific language for electric power system to facilitate requirement formalization for reactive synthesis based on LTL. Finally, we discuss the use of STL and simulation-based design space exploration to check or enforce real-time constraints (e.g. timing) for controller implementation.

A. Synthesis of Reactive Protocols for Electric Power Distribution

The control protocol synthesis problem for electric power system can be stated as follows: given an electric power system topology (generated as discussed in Section V) and a formal specification describing assumptions on the components and requirements for the system, build a controller that reconfigures the system (via turning on and off the contactors) by sensing and reacting to the faults and the changes in system status so as to ensure that the specification is met. Next, we discuss how to formalize the requirements to recast the above problem as a reactive synthesis problem.

1) Variables: Environment variables include the health statuses of components that are uncontrolled. In our formulation, we consider only generators, APUs, and rectifier units as environment variables. They can each take values of healthy (1) and unhealthy (0), and may change at any point in time[1]. *Controlled* variables are contactors, and can each take values of open (0) or closed (1). A closed contactor allows power to pass through, while an open one does not. *Dependent* variables are buses that can be either powered (1) or unpowered (0). Bus values will depend on the status of their neighboring contactors, buses, as well as the health status of connecting generators, APUs, or rectifier units.

Timing considerations play a key part in the specifications for an electric power system. LTL, however, only addresses the notion of temporal ordering of events. To reconcile this discrepancy, we handle timing annotations by introducing clock variables. Verification of actual timing constraints related to the controller implementation is then performed at a lower abstraction level, as detailed in Section III-F.

Based on the set of variables above, system specifications are expressed as a contract $\mathcal{C}_C = (A_C, G_C)$, where assumptions A_C encode the allowable behaviors of the environment the control system operates in, and guarantees G_C encode the controller requirements. By defining $A_C = \{\sigma | \sigma \models \varphi_e\}$ and $G_C = \{\sigma | \sigma \models (\varphi_e \rightarrow \varphi_s)\}$, with φ_e and φ_s as in equation (4), a behavior $\sigma \models \varphi$ if and only if σ is in the guarantees of $\mathcal{C}_C$ (already in saturated form). Therefore, solving the reactive synthesis problem is equivalent to generating an implementation for $\mathcal{C}_C$. The following lists the temporal logic formulas used to concretely express the contracts for controller synthesis for the primary distribution problem in an electric power system.

2) Environment Assumptions: Let $\mathcal{I}$ be an index set enumerating the set of environment variables described in Section VI-A1. For each environment variable e_i, $i \in \mathcal{I}$, let p_i be its probability of failure in a given time interval T as defined in Section V-B. Let r_S be the overall reliability level the system has to achieve, that is, the probability of the overall system failure should be less than or equal to r_S. Assuming independence of component failures, the overall reliability level determines the allowable environment assumptions by providing a bound on the number of simultaneous component failures allowed. More formally, denote a single configuration of the environment (i.e., an environment state) by $\mathbf{e}$. For a given subset $\mathcal{I}' \subseteq \mathcal{I}$ of the environment variables, we define $\mathbf{e}_{\mathcal{I}'} = (e_1, \ldots, e_{|\mathcal{I}|})$, where $e_i = 0$ (unhealthy) if $i \in \mathcal{I}'$; and $e_i = 1$ (healthy) otherwise. Let $h : [0,1] \rightarrow 2^{2^{\mathcal{I}}}$ be the function that maps the system reliability level to the possible environment configurations. We can then enumerate all allowable environment configurations based on the required reliability level, as

$$\mathcal{E}_S = \{\mathbf{e}_{\mathcal{I}'} | \mathcal{I}' \in h(r_S)\}. \tag{23}$$

With this definitions, an environment assumption can be written in LTL as $\square(\mathbf{e} \in \mathcal{E}_S)$.

[1]Generators can be taken offline by the pilot or may stop functioning due to a fault. We do not differentiate between these cases and simply call a generator unhealthy when it is unavailable or malfunctioning.

As the function h can be difficult to compute, alternatively, one can reason about the probability r_C of an environment configuration and map it back to the system reliability level r_S. To this effect, we enumerate all environment configurations that occur with probability more than a given level r_C. Then, if the control synthesis problem is realizable with the assumption $\square(\mathbf{e} \in \mathcal{E}_C)$, this implies that the system level reliability is

$$r_S = \sum_{\mathbf{e} \notin \mathcal{E}_C} \prod_{j:e_j=0} p_j \prod_{j:e_j=1} (1 - p_j).$$

The second environment assumption is also related to failure analysis. We assume that when a component fails during the flight (the interval T), it will not come back online. This can be expressed in LTL as

$$\square \bigwedge_{i \in \mathcal{I}} ((e_i = 0) \rightarrow \bigcirc(e_i = 0)). \tag{24}$$

3) Controller Guarantees: We consider the following system requirements as LTL guarantees for the controller.

Power Status of Buses: An AC bus can only be powered if there exists a *live path* (i.e., all contactors closed along a path) that connects the bus to a healthy AC generator or a healthy APU. Similarly, a DC bus can only be powered if there exists a live path that connects it to a healthy rectifier unit, which itself is connected to a powered AC bus. Let $\tilde{p}_{i,B}$ denote the set of all components (i.e., contactors and buses) along a path between bus B and environment variable e_i for $i \in \mathcal{I}$, excluding B and e_i. Furthermore, let $\mathcal{G} \subseteq \mathcal{I}$ and $\mathcal{R} \subseteq \mathcal{I}$ represent the sets of generators and rectifier units. AC bus B is powered if there exists a live path between B and e_i for $i \in \mathcal{G}$, written as[2]

$$\square \left\{ \bigvee_{i \in \mathcal{G}} \left((e_i = 1) \wedge \bigwedge_{X \in \tilde{p}_{i,B}} (x = 1) \right) \rightarrow (b = 1) \right\}. \tag{25}$$

If there exists no live path between B and a generator e_i for $i \in \mathcal{G}$, then B will be unpowered

$$\square \left\{ \neg \bigvee_{i \in \mathcal{G}} \left((e_i = 1) \wedge \bigwedge_{X \in \tilde{p}_{i,B}} (x = 1) \right) \rightarrow (b = 0) \right\}. \tag{26}$$

A similar set of specifications for DC buses holds in which environment variables e_i span $i \in \mathcal{R}$.

Balanced Power Flow in Nominal Conditions: Under nominal conditions (i.e., when all generators and rectifier units are healthy), the power drawn from each generator by the buses connected to it should be less than the capacity of that generator. Let $\tilde{P}_B$ be a constant that corresponds to the maximum power required by the loads connected to the bus B and $\tilde{P}_{e_i}$ be a constant corresponding to the power generator i can nominally provide. Using the live path constructs, we define the power variables $l_{i,B} \in \{0, \tilde{P}_B\}$ such that $\bigwedge_{X \in \tilde{p}_{i,B}} (x = 1) \rightarrow (l_{i,B} = \tilde{P}_B)$, and $\neg \bigwedge_{X \in \tilde{p}_{i,B}} (x = 1) \rightarrow (l_{i,B} = 0)$. Then, the power flow requirement can be written as

$$\square \left\{ \bigwedge_{i \in \mathcal{I}} (e_i = 1) \rightarrow \bigwedge_{i \in \mathcal{G}} (\tilde{P}_{e_i} \geq \sum_{B \in \mathcal{B}} l_{i,B}) \right\},$$

[2]Per abuse of notation, we denote components by uppercase letters (e.g., C, B) and component statuses by lowercase letters (e.g., c, b).

where $\mathcal{B}$ represents the set of buses.

No Paralleling of AC Sources: To avoid paralleling, we explicitly enumerate and disallow all bad configurations. In Fig. 1, paralleling can occur if there exists a live path that connects two AC generators or APUs. Let $\tilde{p}_{i,j}$ represent the set of components along a path between generators e_i, e_j, for $i, j \in \mathcal{G}$ and $i \neq j$. We disallow configurations in which all contactors $C \in \tilde{p}_{i,j}$ create a live path. These specifications are written as

$$\square \bigwedge_{i,j \in \mathcal{G}} \left\{ \neg \bigwedge_{C \in \tilde{p}_{i,j}} (c = 1) \right\}. \tag{27}$$

Safety-Criticality of Buses: A safety-critical bus can be unpowered for no longer than T_s time steps. This is implemented through the use of an additional clock variable x_B for each bus B, where each "tick" of the clock represents δ time. If the bus is unpowered, then at the next time step clock x_B increases by δ. If B is unpowered, then at the next time step clock x_B resets to zero. Then, we limit the number of steps B can remain unpowered in order to ensure that x_B never becomes larger than T_s. Thus, for all safety-critical buses,

$$\square \{(b = 0) \rightarrow (\bigcirc x_B = x_B + \delta)\}, \tag{28a}$$
$$\square \{(b = 1) \rightarrow (\bigcirc x_B = 0)\}, \tag{28b}$$
$$\square (x_B \leq T_s). \tag{28c}$$

Unhealthy Sources: A bus connected to an unhealthy source (generator or rectifier unit) will create a short-circuit failure, leading to excessive electrical currents, overheating, and possible fires. While generators have internal protections to avoid such failures, we require that appropriate contactors open when a generator or APU becomes unhealthy to isolate the unhealthy source and prevent its use. Let $\mathcal{N}(e_i)$ represent the set of contactors directly connected, or neighboring, environment variable e_i for $i \in \mathcal{I}$. We write the specifications to disconnect all unhealthy sources as

$$\square \bigwedge_{i \in \mathcal{I}} \left\{ (e_i = 0) \rightarrow \bigwedge_{C \in \mathcal{N}(e_i)} (c = 0) \right\}. \tag{29}$$

The above mentioned specifications can be put in assume-guarantee form as in equation (4). Moreover, since they are within the GR(1) fragment of LTL, digital synthesis tools, such as the one implemented in JTLV [30], can be used to automatically synthesize the control protocol. For the examples discussed in this paper, we used the Temporal Logic Planning (TuLiP) Toolbox [31], a collection of Python-based code for automatic synthesis of embedded control software, which provides an interface to JTLV.

4) Capturing Actuation Delays: In the discussion above, we assumed ideal contactors that can be instantaneously controlled. It is possible to capture delays in contactor opening and closing times, as well as the communication delays between the controller and the contactors. To this effect, one can introduce a controlled variable $\tilde{C}$ to represent the controller intent for contactor C and treat the contactor as an environment variable. The uncertain delay between the controller intent and contactor state can be handled by the use of an additional clock variable x_C for each contactor C, where each "tick" of the clock represents δ time. If the contactor intent is open and the contactor state is closed, the contactor opens within $[T_{o_{min}}, T_{o_{max}}]$ units of time unless a close command is issued before it opens. If the contactor intent is closed and the contactor state is open, the contactor closes within $[T_{c_{min}}, T_{c_{max}}]$ units of time unless an open command is issued before it closes. Once the contactor intent is set, if the contactor state does not match the intent, at the next step clock x_C will increase by δ. If contactor state and intent match, then at the next step clock x_C resets to zero:

$$\square \{(\bigcirc c = \tilde{c}) \rightarrow (\bigcirc x_C = 0)\}.$$

When the control command is the same as the contactor state, the contactor state remains the same, i.e.,

$$\square \{(\tilde{c} = c) \rightarrow (\bigcirc c = c)\}.$$

Finally, the assumption capturing the contactor closing behavior in relation to the controller input intent is given by

$$\begin{aligned} \square \{(\tilde{c} = 1 \wedge c = 0 \wedge (x_C < T_{c_{min}})) \rightarrow \\ (\bigcirc c = 0 \wedge \bigcirc x_C = x_C + \delta)\}, \\ \square \{(\tilde{c} = 1 \wedge c = 0 \wedge (x_C \geq T_{c_{min}})) \rightarrow \\ (\bigcirc c = 1 \vee \bigcirc x_C = x_C + \delta)\}, \\ \square (x_C \leq T_{c_{max}}). \end{aligned}$$

The contactor opening behavior can be formally captured in a similar manner. The formulas mentioned in this section enter to the control synthesis problem as new environment assumptions when delays are taken into account. It should also be noted that unhealthy sources can only be disconnected with a delay in this case, therefore formula (29) should be adjusted accordingly.

B. Domain-Specific Language

The lack of familiarity with formal methods among system engineers provides a challenge to the actual adoption of reactive synthesis techniques. Therefore, we also propose an electric power system domain-specific language that enables automatic generation of the LTL specifications described in Sections VI-A2 and VI-A3 out of a set of pre-defined primitives. Our language can smoothly interface with pre-existing tools, such as visual programs for single-line diagrams, which engineers are familiar with, as well as with the topology design framework in Section V.

A graph data structure $\mathcal{G} = (V, E)$ as the one used in Section V can be generated from a visual representation of the topology, provided by the user, or directly imported as a result of the design procedure in Section V. The set of nodes V represents the set of components, consisting of generators, buses, and rectifier units; the set of edges E represents the set of contactors as well as solid wire links between components. The adjacency matrix $\mathbf{A}$ is a square matrix whose diagonal entries are zeros, and whose non-diagonal entries are ones or zeros depending on whether a connection (with or without contactors) exists between vertices. The component properties that are used to formulate the LTL specifications are directly

referenced from the component attributes in the platform library, as described in Section III-A. Given the electric power system topology and the component attributes, the LTL specifications in Section VI-A can be converted from a set of primitives, a representative subset of which are provided in the following.

Environment assumptions: In the environment primitive, the first input is the system reliability level, followed by all subsets of components that are uncontrolled and can fail. As an example, when only generators and rectifier units are assumed to fail, this can be written as $\textbf{env}(r_S, \mathcal{G}_e, \mathcal{R}_e)$, where $\mathcal{G}_e \subseteq \mathcal{G}$ and $\mathcal{R}_e \subseteq \mathcal{R}$, $\mathcal{G}$ and $\mathcal{R}$ being the sets of all generators and rectifier units, respectively.

No-paralleling of AC sources: A "non-paralleling" primitive accepts as inputs any subset of $\mathcal{G}$, and can be written as $\textbf{noparallel}(\mathcal{G}_p)$, where $\mathcal{G}_p \subseteq \mathcal{G}$.

Essential (safety-critical) buses: Let the set of all buses be $\mathcal{B}$. An "essential bus" primitive can input any subset of $\mathcal{B}$ such that the bus elements can be unpowered for no longer than the maximum allowable time as specified in the component library. This primitive is written as $\textbf{essbus}(\mathcal{B}_e)$, where $\mathcal{B}_e \subseteq \mathcal{B}$.

Disconnect unhealthy sources: A "disconnect" primitive can take as input the union of subsets of $\mathcal{G}$ and $\mathcal{R}$. This primitive is written as $\textbf{disconnect}(\mathcal{G}_d, \mathcal{R}_d)$, where $\mathcal{G}_d \subseteq \mathcal{G}$ and $\mathcal{R}_d \subseteq \mathcal{R}$.

C. Co-design of Topology and Control

As mentioned in Section IV, topology and control protocol need to be coherently designed to satisfy the top-level requirements of an electric power system. In this section, we deal with the co-design problem for both system topology and control protocol to satisfy a system contract $\mathcal{C}_S$ with an overall reliability requirement r_S. We show that if system-level requirements are partitioned according to the contracts $\mathcal{C}_T$ and $\mathcal{C}_C$, as defined, respectively, in Section V and Section VI-A, then the electric power system can be designed in a *compositional* way, i.e., the methodologies illustrated in in Section V and Section VI-A can be *independently* deployed, while guaranteeing that the assembled system is correct and satisfies $\mathcal{C}_S$.

In particular, Propositions VI.1 and VI.2 below discuss conditions for the controlled system to satisfy the system-level contract $\mathcal{C}_S$ if the selected topology and control protocol satisfy their contracts $\mathcal{C}_T$ and $\mathcal{C}_C$. First, in Proposition VI.1, we assume that actuation delays are ignored in control synthesis. We then remove this assumption in Proposition VI.2.

Proposition VI.1. *Assume contactor delays are ignored in control synthesis (i.e., $T_{o_{min}} = T_{o_{max}} = T_{c_{min}} = T_{c_{max}} = 0$, therefore no contactor intent variable is introduced). If the topology implements its contract $\mathcal{C}_T$ with a reliability level r_T, then a centralized control implementing its contract $\mathcal{C}_C$ for this topology is always realizable when a reliability level $r_S \geq r_T$ is used while generating the environment assumptions as in* (23)*. Moreover, the controlled system will satisfy the system-level requirements with a reliability level r_S.*

Proof. As shown in Fig. 2 and 3, both the topology synthesis and control synthesis steps are based on a consistent set of models and share the same labelled topology template $\mathcal{T}$. In fact, the configurations conforming to $\mathcal{T}$ are the assumptions for the topology contract $\mathcal{C}_T$, while the synthesized topology is used to generate the LTL formulas for the controller contract $\mathcal{C}_C$. We prove the realizability of the controller by discussing the system-level requirements listed in Section VI-A3 and Section V as follows:

(a) *Reliability Requirements.* In both the topology and control design steps, we assume that when a component fails it will not come back online. Therefore, reliability requirements are treated as static requirements in failure condition. If the topology guarantees a reliability level r_T, then there are enough components and paths from generators to critical loads such that any combination of component faults causing a system failure has a joint probability $p < r_T$. Let $\overline{\mathcal{E}}_T$ be set of all the environment configurations that correspond to these combinations of component faults, and let $\mathcal{E}_T$ be its complement. Then any combination of faults associated with a configuration in $\mathcal{E}_T$ does not cause any loss in system functionality because of the available redundancy. Since $r_S \geq r_T$ is used in $\mathcal{C}_C$, $\mathcal{E}_S \subseteq \mathcal{E}_T$ will also hold, hence accommodating any combination of faults associated with an environment configuration in $\mathcal{E}_S$ will also be feasible. Therefore, a centralized controller assuming a reliability level r_S in $\mathcal{C}_C$ will always realize this specification, thus guaranteeing an overall reliability level $r = r_S$ for the controlled system.

(b) *Balanced Power Flow in Nominal Conditions.* Power requirements are treated as static requirements in nominal condition. Power flow constraints in the topology optimization problem enforce that loads on each side of the topology graph are selectively connected to one or more generators on the same side, in such a way that the total power capability of the generators is equal or larger than the required power from the respective loads. It is, therefore, enough to use the available paths in the synthesized topology for a centralized controller to realize this specification.

(c) *Unhealthy Sources.* Connectivity constraints in the topology optimization problem enforce that any edge (interconnection) originating from a source node (generator or rectifier unit) is associated with a contactor. Therefore, it is always possible for a centralized controller to open such contactors to isolate unhealthy sources and realize this specification. Since contactors can be instantaneously operated, full isolation of unhealthy sources is guaranteed within one time step (δ time).

(d) *No Paralleling of AC Sources.* As discussed above, all AC sources can be isolated by opening the related contactors. Moreover, connectivity constraints prescribe that AC buses be also connected via contactors. This makes it possible for a centralized controller to always realize this specification by isolating buses connected to different AC sources as well as isolating unhealthy sources while inserting healthy ones.

(e) *Safety-Criticality of Buses.* Since all contactors are assumed as ideal and instantaneously controllable, it is always possible for a centralized controller to configure the topology and realize this specification whenever $T_s \geq \delta$.

We then conclude that the conjunction of the LTL formulas used in $\mathcal{C}_C$ to formalize requirements (b)-(e) under the assumptions in (a) can always be realized by a centralized controller

if $r_T \leq r_S$ holds. □

Based on Proposition VI.1, for the controlled system to satisfy a contract $\mathcal{C}_S$ with a reliability level r_S, it is enough to select a topology that implements its contract $\mathcal{C}_T$ with a reliability level $r_T \leq r_S$, and then synthesize a centralized controller for the selected topology by using a reliability level r_S to generate the environment assumptions. When contactor delays are not ignored in control synthesis, a similar proposition holds if an additional condition is assumed on the maximum bus unpowered time T_s allowed in (28), as discussed below.

Proposition VI.2. *Assume delays in the contactors are taken into account in control synthesis (i.e., $T_{o_{min}} > 0$ and $T_{c_{min}} > 0$). If the topology implements its contract $\mathcal{C}_T$ with a reliability level r_T, then there exists a large enough time T^* such that a centralized control implementing its contract $\mathcal{C}_C$ for this topology is realizable when a reliability level $r_S \geq r_T$ in equation* (23) *and a bus unpowered time $T_s \geq T^*$ in equation* (28) *are used while generating $\mathcal{C}_C$. Moreover, the controlled system will satisfy the system-level requirements with a reliability level r_S.*

Proof. As in Proposition VI.1, both the topology and control synthesis steps are based on a consistent set of models and share the same template $\mathcal{T}$. Moreover, we can prove the realizability of the controller by discussing the requirements in Section VI-A3 and Section V one at a time. In particular, the static specifications in (a) and (b) will be always realizable by the same arguments used in Proposition VI.1. To show that the dynamic requirements in (c), (d) and (e) will also be realizable when actuation delays are taken into account, we proceed as follows:

(c)-(d) *Unhealthy Source Isolation and AC Sources Paralleling.* By the same arguments used in Proposition VI.1 (c), all sources (including AC sources) can be isolated by opening the related contactors. Moreover, the topology connectivity constraints prescribe that AC buses should also be connected via contactors. Even if contactors can only be opened or closed with a delay, it is still possible for a centralized controller to realize this specification by disconnecting two AC buses at least $T_{o_{max}}$ time before connecting them to different AC sources or by isolating unhealthy sources at least $T_{o_{max}}$ time before connecting the healthy ones.

(e) *Safety-Criticality of Buses.* To guarantee that safety-critical buses are unpowered for no longer than the desired time T_s, the controller needs to reconfigure the topology by opening and closing sets of contactors to deactivate existing components and paths and activate new ones. Because of the actuation delay, topology reconfigurations cannot occur instantaneously; some sets of contactors will need to be actuated in sequence to guarantee isolation of unhealthy sources and prevent paralleling of AC sources, as required in (c) and (d). Since there is a finite number of topology configurations, there will also be a finite number of possible reconfigurations $\mathcal{R}$. Consider the step i from an initial configuration A_i to a final configuration Z_i. Let n_o^i and n_c^i be the minimum number of contactor sets that must be, respectively, opened and closed in sequence in order to provide power to a critical bus during reconfiguration i. Then, the minimum (worst-case) time during which at least one critical bus stays unpowered will be

$$T_i = n_o^i \left\lceil \frac{T_{o_{max}}}{\delta} \right\rceil \delta + n_c^i \left\lceil \frac{T_{c_{max}}}{\delta} \right\rceil \delta.$$

Let $T^* = \max_{i \in \mathcal{R}} T_i$; then, T^* is the minimum bus unpowered time that can always be guaranteed across all possible topology reconfigurations. Therefore, a centralized controller can always be realizable when $T_s \geq T^*$ is chosen in $\mathcal{C}_C$.

As in Proposition VI.1, by combining the arguments above with the ones used in (a) and (b), we can conclude that the conjunction of the LTL formulas used in $\mathcal{C}_C$ to formalize requirements (b)-(e) under the assumptions in (a) can always be realized by a centralized controller if $r_T \leq r_S$ and $T_s \geq T^*$ hold. □

D. Simulation-Based Design Space Exploration

As shown in Section VI-C, the design steps in Section V and Section VI-A allow synthesizing electric power system architectures and control protocols that jointly satisfy the top-level system specifications, represented by contracts $\mathcal{C}_T$ and $\mathcal{C}_{C,LTL}$. To assess the satisfaction of real-time performance constraints, we monitor STL formulas from the controller contracts $\mathcal{C}_{C,STL}$ and $\mathcal{C}'_{C,LTL}$ on voltage and current waveforms over time, as discussed in Section III-F and IV. As an example, we investigate here the maximum reaction time allowed to the controller. For this purpose, we assume a synchronous implementation of the controller, running at a fixed period T_r. In our hybrid model, the BPCU is connected in closed loop with the power system plant, while failure events can be injected by setting the input signal $\boldsymbol{u}(t)$. Moreover, we assume that all contactors respond with a fixed delay T_d to the open/close commands from the BPCU. We then consider the requirement that a DC essential bus must never be unpowered for more than t_{max} under any possible failure scenario. In a continuous setting, such a requirement is translated by stating that the DC bus voltage V_{DC} should never deviate from the desired value V_d by more than a margin ϵ for more than t_{max}. The predicate specifying that the current value of the voltage stays in the desired range is: $|V_{DC}(t) - V_d| < \epsilon$. Then, the STL formula expressing this to be false for at least t_{max} is:

$$\chi = \Box_{[0,t_{max}]} \neg (|V_{DC}(t) - V_d| < \epsilon). \tag{30}$$

Since we need to enforce that V_{DC} is never out of range only after the initial start-up transient time τ_i, we require

$$\phi(\tau_i) = \neg (\Diamond_{[\tau_i, \infty)} \ \chi) \tag{31}$$

to be true.

To compute the maximum amount of time elapsed while the DC bus voltage is out of range, i.e. for how long at most the voltage requirement on the DC bus is violated, we turn (30) into a PSTL formula, by introducing the timing parameter τ_e, after which an out-of-range voltage event is detected, as follows:

$$\psi(\tau_e) = \Box_{[0,\tau_e]} \neg (|V_{DC}(t) - V_d| < \epsilon). \tag{32}$$

The initial start-up transient time τ_i is estimated from simulation as a function of T_r and T_d. Then, the maximum violation period $\tau_e^*(T_r, T_d)$ can be computed as the

$$\sup\{\tau_e \geq 0 \mid \phi\left(\tau_i(T_r, T_d), \tau_e\right) = False\}, \tag{33}$$

where

$$\phi(\tau_i, \tau_e) = \neg(\diamondsuit_{[\tau_i, \infty)}\ \psi(\tau_e)). \tag{34}$$

The formula in (34) allows exploring the T_r-versus-T_d design space and finding the maximum allowed controller reaction time T_r^* for a fixed T_d^*, in such a way that the essential DC bus is never out of range for more than t_{max}. To do so, we cast an optimization problem following the formulation in (15)

$$\min_{T_r>0} \quad 1/T_r \tag{35}$$
$$\text{s.t.} \begin{cases} \mathcal{F}(\boldsymbol{u}, V_{DC}, T_r) = 0 \\ \\ V_{DC} \models \phi\left(\tau_i(T_r, T_d^*), \tau_e\right) \quad \forall \tau_e \geq t_{max} \\ \forall \boldsymbol{u} \text{ s.t. } \boldsymbol{u} \models \varphi_e' \end{cases}$$

where $\boldsymbol{C} = 1/T_r$ is the cost function, $\boldsymbol{\kappa} = T_r$ is the design parameter, $\varphi_s(\boldsymbol{\tau}) = \bigwedge_{\tau_e \geq t_{max}} \phi\left(\tau_i(T_r, T_d^*), \tau_e\right)$ is the conjunction of PSTL formulas that must be satisfied, each parameterized by $\boldsymbol{\tau} = T_r$, and φ_e' refines the environment assumption formula φ_e in Section VI-A. In this case, the system behavior $\boldsymbol{s}$ is the trace $\boldsymbol{s} = (\boldsymbol{u}, V_{DC})$, where V_{DC} is the output signal to be observed during simulation, and $\boldsymbol{u}$ spans the set of all admissible failure injection traces that are consistent with the environment assumptions in Section VI-A2.

The formulation in (35) includes an infinite set of formulas that must be satisfied for all admissible failure traces and values of $\tau_e \geq t_{max}$. However, such formulation can be further simplified, by observing that (35) is equivalent to

$$\max_{T_r>0} \quad T_r \tag{36}$$
$$\text{s.t.} \begin{cases} \mathcal{F}(\boldsymbol{u}, V_{DC}, T_r) = 0 \\ \\ \tau_e^*(T_r, T_d^*) \leq t_{max} \qquad \forall \boldsymbol{u} \text{ s.t. } \boldsymbol{u} \models \varphi_e' \end{cases}$$

where τ_e^* is defined in (33). Moreover, as shown in Section VII-C, it is enough to compute $V_{DC}(t)$ and τ_e^* under the worst case failure scenario, rather than for all possible failure traces, whenever the worst case assumptions on $\boldsymbol{u}(t)$ can be determined a priori. Problem (36) can then be solved by first solving the optimization problem in (33) to compute τ_e^* as a function of T_r and T_d^* in the worst case input scenario, and then by computing the value T_r^* of the controller reaction time that makes τ_e^* equal to t_{max}. For the example discussed in this paper, we used the BREACH toolbox [38] to facilitate post-processing of simulation traces and verify the satisfaction of the STL formulas.

VII. DESIGN EXAMPLE AND RESULTS

We illustrate our methodology on the proof-of-concept design of the primary power distribution of an electric power system, involving the configuration of contactors to deliver power to high-voltage AC and DC buses and loads.

TABLE III
LOAD REQUIREMENTS

Component	Requirement (W)
LL1	30000
LL2	40000
RL1	20000
RL2	30000

TABLE IV
GENERATOR POWER RATINGS

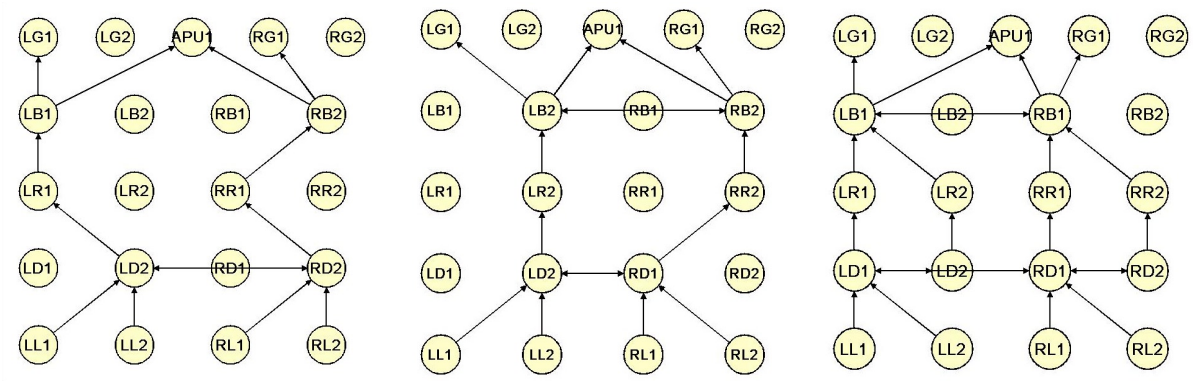
Component	Capability (W)
LG1	70000
LG2	30000
RG1	50000
RG2	40000
APU	100000

A. Topology Synthesis

The topology synthesis algorithm has been implemented in MATLAB and leverages CPLEX [39] to solve the MILP at each iteration. We present the result obtained for an electric power system topology template $\mathcal{T}$ consisting of two generators, two AC buses, two rectifiers, two DC buses and two loads on each side. Tables III and IV report the load power requirements and the generator power ratings in our example; Table V shows the component costs, while the failure probabilities are reported in Table VI.

Figures 4 and 5 show the topologies obtained after running the synthesis algorithm when a set of strategies are sequentially implemented after every MILP iteration to increase reliability and satisfy the specified level $r^* = 10^{-9}$. By solving the MILP including only connectivity and power flow constraints, we obtain the topology in Fig. 4, the simplest possible architecture, which only provides a single path from a load to a generator (or APU) on each side. Such a topology presents the highest load and system failure probabilities, as shown in Table VII.

In Fig. 5 a) and b) horizontal connections are added between the DC buses and AC buses of the left and right hand sides of the system. Because increasing the number of components is expensive, the algorithm first tries to increase reliability by adding connections among existing components at the cost of additional contactors. Additional components (e.g. buses and rectifiers) are finally used in Fig. 5 c) to achieve the desired specification. In Table VII, we report the achieved reliability level (failure probability) at load $LL1$ and the overall achieved system reliability level r_T, as computed for the topologies in Fig. 4 and 5. The total computation time to generate the topologies in Fig. 4 and 5 was 19.7 s on an Intel Core i7 2.8-GHz Processor with 6-GB memory. In a typical run, the number of necessary paths to achieve r^* is estimated after the first MILP step and convergence to the final topology occurs in no more than two iterations, which reduces the overall computation time to approximately 10 s.



(a) Topology 2 (b) Topology 3 (c) Topology 4

Fig. 5. Candidate topologies for an electric power system consisting of rows of (from top to bottom) generators, AC buses, rectifier units, DC buses, and DC loads.

TABLE V
COMPONENT COSTS

Component	Cost
Generator	Generator power/100
APU	APU power/100
AC-Bus	200
Rectifier	200
DC-Bus	200
Contactor	100

TABLE VI
COMPONENT FAILURE PROBABILITIES

Component	Failure Probability
Generator/APU	10^{-5}
Rectifier	2×10^{-4}

TABLE VII
LOAD AND SYSTEM FAILURE PROBABILITIES FOR THE TOPOLOGIES IN FIG. 4 AND 5

Topology	Load Failure Probability	System Failure Probability (r_T)
1 (Fig. 4)	2×10^{-4}	4×10^{-4}
2 (Fig. 5)	4×10^{-8}	4×10^{-8}
3 (Fig. 5)	4×10^{-8}	4×10^{-8}
4 (Fig. 5)	2.6×10^{-15}	2.6×10^{-15}

B. Control Synthesis

To validate our approach, for each of the four topologies in Section VII-A, we formalize a set of environment assumptions and system specifications to synthesize centralized and distributed control protocols for an overall reliability level $r_S = r_T$, as discussed in Section VI-C. For the purpose of brevity, we present the variables and formal specifications, written in LTL, for the topology depicted in Fig. 5 b) only.

1) Centralized Synthesis: Environment Variables: Generators $LG1, LG2, APU1$ and rectifier units $LR2$ and $RR2$ are uncontrolled variables that can switch between healthy (1) and unhealthy (0).

Controlled Variables: Contactors $C_{i,j}$[3] (depicted only as wires in Fig. 5) are variables that are set to open (0) or closed (1).

Dependent Variables: Buses are either powered (1) or unpowered (0) depending on the status of environment and controlled variables.

Environment Assumption: We allow environment configurations which are mapped back from the function h in Section VI-A2 to an overall system reliability level r_S. Topology 3 from Fig. 5 b) has a total of 32 environment configurations. For a reliability level $r_S = 4 \times 10^{-8}$, $h(r_S)$ maps to a set of 21 allowable configurations. The specification can be written as a conjunction of all configurations. More compactly, the environment assumption disallows configurations in which either both rectifiers fail or all generators fail. Thus, we can equivalently write the environment assumption for Topology 3 as:

$$\square\neg((LG1 = 0) \wedge (APU1 = 0) \wedge (RG1 = 0)) \wedge \square\neg((LR2 = 0) \wedge (RR2 = 0)).$$

No Paralleling of AC Sources: No combination of contactors can be closed so that a path exists between generators:

$$\square\neg((C_{LG1,LB2} = 1) \wedge (C_{APU1,LB2} = 1)) \wedge \square\neg((C_{APU1,RB2} = 1) \wedge (C_{RG1,RB2} = 1)).$$

Power Status of Buses: A bus can only be powered if there exists a path (in which a contactor is closed) between a bus and a generator. In Fig. 5 b), bus $LB2$ is powered if either generator $LG1$ or $APU1$ is powered, and the contactor between generator and bus is closed:

$$\square((LG1 = 1) \wedge (C_{LG1,LB2} = 1) \rightarrow (LB2 = 1)),$$
$$\square((APU1 = 1) \wedge (C_{APU1,LB2} = 1) \rightarrow (LB2 = 1)).$$

If neither of these two cases is true, then $LB2$ will be unpowered. These specifications are written as

$$\square(\neg(((LG1 = 1) \wedge (C_{LG1,LB2} = 1) \vee ((APU1 = 1) \wedge (C_{APU1,LB2} = 1))) \rightarrow (LB2 = 0)).$$

Similar specifications may be written for buses $RB2, LD2$, and $RD1$.

[3] i and j denote the name of the components contactor $C_{i,j}$ connects.

Safety-Criticality of Buses: We consider all buses to be safety-critical; at the abstraction level of LTL, this is equivalent to require that at no time can any bus be unpowered

$$\square((LB2 = 1) \wedge (RB2 = 1) \wedge (LD2 = 1) \wedge (RD1 = 1)).$$

The resulting controller has 32 states with a computation time of 1.6 s on a 2.2-GHz Intel Core Processor with 4-GB memory.

2) Distributed Synthesis: For the topology in Fig. 5 b), the distributed control synthesis problem can be solved by splitting the topology into two subsystems S_1 and S_2. The sets $\mathcal{E}_{S_1}, \mathcal{S}_{S_1}$, and $\mathcal{E}_{S_2}, \mathcal{S}_{S_2}$ contain all environment and system variables for subsystems S_1 and S_2, respectively. $\mathcal{E}_{S_1}$ is composed of generators $LG1$, $APU1$ and $RG1$. $\mathcal{S}_{S_1}$ contains AC buses $LB2, RB2$, and contactors $C_{LG1,LB2}, C_{APU1,LB2}, C_{RG1,RB2}$, $C_{LB2,RB2}$. $\mathcal{E}_{S_2}$ is composed of rectifiers $LR2, RR2$ and AC buses $LB2, RB2$, while $\mathcal{S}_{S_2}$ contains DC buses $LD2, RD1$ and contactors $C_{LR2,LD2}, C_{RR2,RD1}$, $C_{LD2,RD1}$. We assume the link between AC buses and rectifier units is a solid wire.

The environment assumption $\varphi_{e_{S_1}}$ for subsystem S_1 enforces that at least one generator will always remain healthy. Environment assumption $\varphi_{e_{S_2}}$ enforces that at least one rectifier unit will always remain healthy. In addition, it also assumes that both AC buses will always be powered. This is an additional guarantee S_1 must provide to S_2 for the distributed synthesis problem to become realizable. All other specifications remain the same as the centralized control problem.

The synthesized controllers for S_1 and S_2 contains 4 and 8 states, respectively. Each controller has a computation time of approximately 0.5 s on a 2.2-GHz Intel Core Processor with 4-GB memory.

C. Design Space Exploration for Real-Time Performance

Continuous-time models of the plant are implemented in SIMULINK, by exploiting the SimPowerSystems extension. As an example, the continuous-time model of a generator consists of a mechanical engine (turbine), a three-phase synchronous machine, in addition to the generator control unit, driving the field voltage of the generator. In addition to timing properties, our power network model allows measuring current and voltage levels at the different circuit loads. It can be discretized to speed up simulations and can seamlessly interface also with MATLAB functions or StateFlow models implementing the controller.

In what follows, we focus on the centralized controller for topology 3 in Fig. 5 b), and provide results for the design exploration problem in Section VI-D. In particular, we are interested in finding the maximum controller reaction time T_r^* as a function of T_d, so that the essential DC bus $LD2$ is never out of range for more than $t_{max} = 70$ ms. Based on the environment assumptions discussed in Section VII-B, the worst case failure scenario for the left DC bus $LD2$ occurs when cascaded failures in two generators (e.g. $LG1$ and $APU1$) and one rectifier ($LR2$) correlate so as to maximize the time the bus voltage is out of the specified range. The controller reacts to a generator fault by routing power from another generator and connects the two DC buses $LD2$ and $RD1$ when one of the rectifiers fails. Therefore, the worst case failure scenario occurs when the rectifier fault happens at the end, and any fault after the first one happens right before $LD2$ fully recovers from the previous fault, while trying to reach the desired voltage level.

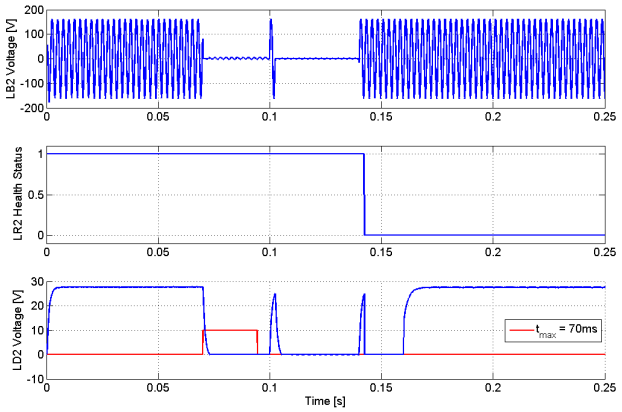


Fig. 6. Real-time requirement violation at the DC bus $LD2$ due to two generator faults followed by a rectifier fault.

Figure 6 shows the simulated voltage V_{LD2} of bus $LD2$ as a function of time, in the worst case scenario, for $T_r = 20$ ms and $T_d = 20$ ms, both defined as in Section VI-D. The waveforms at the top and bottom of the figure are the voltage signals at the $LB2$ (AC) and $LD2$ (DC) buses, respectively. The signal in the middle represents the health status of $LR2$. Both the AC and DC voltages decay to zero because of the generators' faults. When a fault is also injected into $LR2$, an additional drop in the DC voltage is observed. The red signal at the bottom of the figure is interpreted as a Boolean signal, which is high (one) when χ in equation (30) holds (i.e. the requirement is violated) and low (zero) otherwise. To evaluate the formula (30), we used $V_d = 28$ V, $\epsilon = 2$ V and $t_{max} = 70$ ms. The requirement on the DC bus is violated for 24.4 ms. Therefore, ($T_r = 20$ ms, $T_d = 20$ ms) is an unsafe parameter set.

The T_r versus T_d design space is explored in Fig. 7 and 8 by following the optimization procedure in Section VI-D. We sampled the parameter space in approximately 4 hours to obtain a 15×15 point grid. The first plot represents the maximum amount of elapsed time τ_e^*, while the DC bus voltage is out of range, i.e. for how long the requirement on the DC bus is violated, as computed in equation (33). Such a violation period is then compared with the "hard" threshold $t_{max} = 70$ ms in Fig. 8, thus providing the designer with the "safe" region (marked in blue in Fig. 8) for the controller reaction time as a function of the contactor delay. As an example, for a specific value of $T_d = 20$ ms the maximum BPCU reaction time T_r^* allowed for safe operation is 6.5 ms.

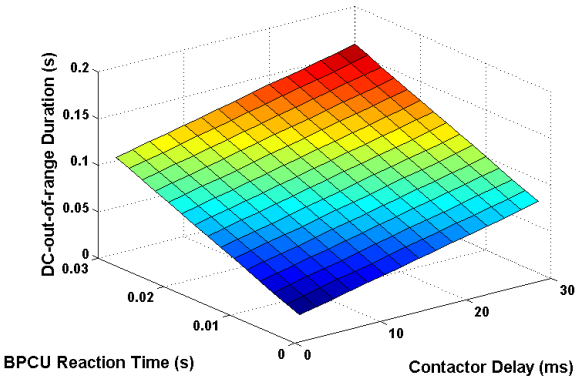


Fig. 7. Maximum duration of the violation of the DC bus voltage requirement.

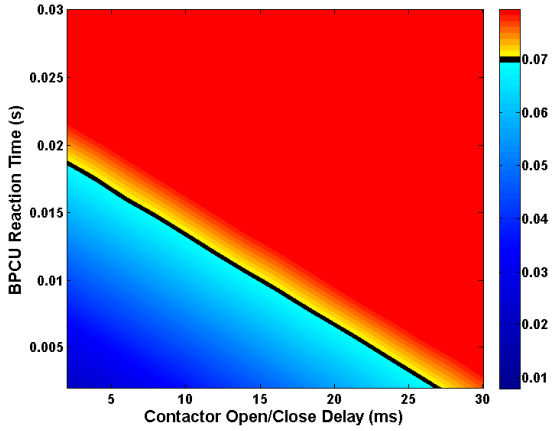


Fig. 8. BPCU reaction times and contactor delays in the blue region satisfy the DC bus requirement.

VIII. Conclusions

We presented a rigorous platform-based methodology for the design of an aircraft electric power system. Our flow consists of three main phases: topology synthesis, control synthesis, and simulation-based design space exploration and verification. To express system requirements, we adopt different formalisms supported by specialized synthesis and analysis frameworks. To generate the system topology, we formulate a mixed integer-linear program that minimizes the overall cost while satisfying a set of connectivity, power flow and reliability requirements, expressed in terms of linear arithmetic constraints on Boolean variables and probabilistic constraints. To synthesize a controller for a given topology, we leverage results from reactive synthesis of control logic from linear temporal logic specifications. We then refine these LTL specifications into signal-temporal logic constructs to assess the real-time system performance and explore the design space at a lower abstraction level, based on high fidelity behavioral models. Our compositional approach uses contracts to guarantee independent implementation of system topology and control, since both topology synthesis and control synthesis rely on a consistent set of models and design constraints.

We plan to extend our control synthesis algorithms to support richer formal languages (e.g., timed logic, branching logic), continuous-time specifications and continuous dynamics (e.g., transients, network and communication delays). Moreover, we plan to investigate techniques for automatic generation of local contracts for the synthesis of distributed and hierarchical control architectures.

IX. Acknowledgments

The authors wish to acknowledge Rich Poisson and Eelco Scholte from United Technologies Corporation (UTC), Mohammad Mozumdar, Antonio Iannopollo and Ufuk Topcu for helpful discussions. 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

[1] I. Moir and A. Seabridge, *Aircraft Systems: Mechanical, Electrical and Avionics Subsystems Integration. Third Edition*. Chichester, England: John Wiley and Sons, Ltd, 2008.

[2] T. Jomier *et al.*, "Final MOET technical report," Tech. Rep., Dec. 2009. [Online]. Available: http://www.eurtd.com/moet/

[3] K. Sampigethaya and R. Poovendran, "Aviation cyber-physical systems: Foundations for future aircraft and air transport," *Proc. IEEE*, vol. 101, no. 8, pp. 1834–1855, 2013.

[4] A. Sangiovanni-Vincentelli, "Quo vadis, SLD? Reasoning about the trends and challenges of system level design," *Proc. IEEE*, no. 3, pp. 467–506, 2007.

[5] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming Dr. Frankenstein: Contract–based design for cyber–physical systems," in *Conf. Decision and Control*, Dec. 2011.

[6] P. Krus and J. Nyman, "Complete aircraft system simulation for aircraft design - paradigms for modelling of complex systems," in *Int. Congress of Aeronautical Sciences*, 2000.

[7] J. Bals, G. Hofer, A. Pfeiffer, and C. Schallert, "Virtual iron bird - a multidisciplinary modelling and simulation platform for new aircraft system architectures," in *German Aerospace Conference*, 2005.

[8] *Modelica Language*. [Online]. Available: http://www.modelica.org

[9] T. Kurtoglu, P. Bunus, and J. de Kleer, "Simulation-based design of aircraft electrical power systems," in *Int. Modelica Conf.*, Mar. 2011, pp. 92–106.

[10] S. Uckun, "META II: Formal co-verification of correctness of large-scale cyber-physical systems during design," Tech. Rep., Sep. 2011. [Online]. Available: http://www.darpa.mil/uploadedFiles/Content/Our_Work/TTO/Programs/AVM/PARC META Final Report.pdf

[11] *OMG Systems Modeling Language*. [Online]. Available: http://www.sysml.org/

[12] M. Masin, A. Sangiovanni-Vincentelli, A. Ferrari, L. Mangeruca, H. Broodney, L. Greenberg, M. Sambur, D. Dotan, S. Zolotnizky, and S. Zadorozhniy, "META II: Lingua franca design and integration language," Tech. Rep., Aug. 2011. [Online]. Available: http://www.darpa.mil/uploadedFiles/Content/Our_Work/TTO/Programs/AVM/IBM META Final Report.pdf

[13] A. Pinto, S. Becz, and H. M. Reeve, "Correct-by-construction design of aircraft electric power systems," in *AIAA Aviation Technology, Integration, and Operations Conf.*, 2010.

[14] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Formal synthesis of embedded control software for vehicle management systems," in *AIAA Infotech@Aerospace*, 2011.

[15] N. Ozay, U. Topcu, and R. M. Murray, "Distributed power allocation for vehicle management systems," in *Int. Conf. Decision and Control*, 2011, pp. 4841–4848.

[16] H. Xu, U. Topcu, and R. M. Murray, "A case study on reactive protocols for aircraft electric power distribution," in *Int. Conf. Decision and Control*, 2012.

[17] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurrent systems: Specification*. Springer, 1992, vol. 1.

[18] E. A. Emerson, "Temporal and modal logic," *Handbook of theoretical computer science*, vol. 2, pp. 995–1072, 1990.

[19] R. G. Michalko, "Electrical starting, generation, conversion and distribution system architecture for a more electric vehicle," *US Patent 7,439,634 B2*, Oct. 2008.

[20] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K. Larsen, "Contracts for systems design," *Proc. IEEE*, to appear 2013.

[21] P. Nuzzo, A. Sangiovanni-Vincentelli, X. Sun, and A. Puggelli, "Methodology for the design of analog integrated interfaces using contracts," *IEEE Sensors J.*, vol. 12, no. 12, pp. 3329–3345, Dec. 2012.

[22] A. Pnueli, "The temporal logic of programs," in *Annual Symp. on Foundations of Computer Science*, Nov. 1977, pp. 46–57.

[23] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Massachussetts, USA: The MIT Press, 2008.

[24] R. Alur and T. A. Henzinger, "A really temporal logic," in *Symposium on Foundations of Computer Science*, 1989, pp. 164–169.

[25] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Syst.*, vol. 2, no. 4, pp. 255–299, 1990.

[26] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Modeling and Analysis of Timed Systems*, 2004, pp. 152–166.

[27] E. Asarin, A. Donzé, O. Maler, and D. Nickovic, "Parametric identification of temporal properties," in *Runtime Verification*, 2011, pp. 147–160.

[28] A. Pnueli and R. Rosner, "Distributed reactive systems are hard to synthesize," in *Proc. Annual Symp. on Foundations of Computer Science*, vol. 2, Oct. 1990, pp. 746–757.

[29] N. Piterman and A. Pnueli, "Synthesis of reactive(1) designs," in *In Proc. Verification, Model Checking, and Abstract Interpretation*. Springer, 2006, pp. 364–380.

[30] A. Pnueli, Y. Saar, and L. D. Zuck, "Jtlv: A framework for developing verification algorithms," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, T. Touili, B. Cook, and P. Jackson, Eds. Springer Berlin Heidelberg, 2010, vol. 6174, pp. 171–174.

[31] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray, "TuLiP: a software toolbox for receding horizon temporal logic planning," in *Proc. Int. Conference on Hybrid Systems: Computation and Control*. New York, NY, USA: ACM, 2011, pp. 313–314.

[32] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Formal Modeling and Analysis of Timed Systems*, 2010, pp. 92–106.

[33] Uppaal-tiga, a synthesis tool for timed games. [Online]. Available: http://people.cs.aau.dk/~adavid/tiga/

[34] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Satisfiability*, A. Biere, H. van Maaren, and T. Walsh, Eds. IOS Press, 2009, vol. 4, ch. 8.

[35] C. Hang, P. Manolios, and V. Papavasileiou, "Synthesizing cyber-physical architectural models with real-time constraints," in *Proc. Int. Conf. Comput.-Aided Verification*, Dec. 2011.

[36] M. R. Lyu *et al.*, *Handbook of software reliability engineering*. IEEE Computer Society Press CA, 1996, vol. 3.

[37] B. Kaiser, P. Liggesmeyer, and O. Mäckel, "A new component concept for fault trees," in *Proc. Australian Workshop on Safety Critical Systems and Software*, 2003.

[38] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *Proc. Int. Conf. Comput.-Aided Verification*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 167–170.

[39] (2012, Feb.) IBM ILOG CPLEX Optimizer. [Online]. Available: www.ibm.com/software/integration/optimization/cplex-optimizer/