ABSTRACT

Modern aircraft increasingly rely on electric power for subsystems that have traditionally run on mechanical power. The complexity and safety-criticality of aircraft electric power systems have therefore increased, rendering the design of these systems more challenging. This work is motivated by the potential that correct-by-construction reactive controller synthesis tools may have in increasing the effectiveness of the electric power system design cycle. In particular, we have built an experimental hardware platform that captures some key elements of aircraft electric power systems within a simplified setting. We intend to use this platform for validating the applicability of theoretical advances in correct-by-construction control synthesis and for studying implementation-related challenges. We demonstrate a simple design workflow from formal specifications to autogenerated code that can run on software models and be used in hardware implementation. We show some preliminary results with different control architectures on the developed hardware testbed.

Categories and Subject Descriptors
D.2.4 [Software Engineering]: Software/Program Verification—Formal methods

Keywords
reactive synthesis; testbed; aircraft electric power system

1. INTRODUCTION AND MOTIVATION

Aircraft electric power systems have become increasingly important over the years because they support various subsystems and essential services on aircraft. These electrical services and subsystems are commonly referred to as system loads. System loads are of two categories, namely, primary loads (some of these are safety- or mission-critical) and secondary (noncritical) loads. The system needs to ensure that the primary loads are supplied with power at all times; that is, if a fault affects a part of the system that powers a primary load, the system must be able to reconfigure and provide power to the load through another path. In order to reconfigure a system, it is necessary to reroute power, which is accomplished with high power electromagnetic devices called contactors. The contactors are arranged such that they are magnetically held in a preferred state by an applied signal. The state is either open or closed. To reconfigure the contactors to react to faults and modes of operation, the system uses control logic that can sense system conditions and environmental conditions under which the system operates. The electric power system, therefore, includes voltage and current sensors connected to the control logic. In current practice, the control logic is often designed by hand, resulting in lengthy design and verification cycles. As an alternative approach, [12] and [13] explored the application of correct-by-construction reactive controller synthesis techniques.

In this paper, we report on our recently developed simulation models and a hardware testbed for validating reactive controllers synthesized using TuLiP [12], a temporal logic planning toolbox, in order to investigate the validity of the assumptions made in controller synthesis. TuLiP is a collection of Python-based code used for automatic synthesis of correct-by-construction embedded control software. Automatic synthesis of reactive centralized and distributed controllers of aircraft electric power systems is described in detail in [13]. The particular distributed synthesis method adopted in this study is introduced in [4] and [5].

University-scale testbeds for research on correct-by-construction controller synthesis are fairly limited. An advanced diagnostics and prognostics testbed is described in [9]. Some applications of this testbed to the electric power systems of spacecraft and aircraft are detailed in [3]. However, the experiments focused on diagnostic queries of the system, while our work is focused on the implementation of correct-by-construction control protocols for fault-tolerant operations. A robotics testbed implementing correct-by-construction controllers is described in [2].

TuLiP can be used to synthesize logic so that the satisfaction of certain safety requirements is guaranteed. The synthesized logic enables the contactors to react to changes in system conditions such as the status of generators and rectifier units. This is commonly referred to as a reactive system. The safety requirements used in our simulation models and hardware testbed stipulate that the alternating current generators should never be paralleled and that the duration for which the bus is not powered should never

*An extended version of this paper is available at [10].
two basic temporal modalities next (•) and until (••). In order to validate the controller on the experimental hardware platform, we synthesized and tested it using TüLiP and SimPowerSystems, respectively. Thereafter, we investigated the validity of the assumptions used for controller synthesis on the experimental hardware platform.

An aircraft electric power system uses different voltage levels, which can broadly be divided into four categories, namely, high-voltage AC, high-voltage DC, low-voltage AC, and low-voltage DC. The topology in Figure 1 is of specific interest because it is representative of some of the key features of aircraft electric power systems in simplified settings. Therefore, the hardware testbed was built based on the above mentioned topology.

2. THEORETICAL BACKGROUND

We now discuss the formal specification language utilized for the synthesis of control protocols and how these protocols are implemented in the software models and on the testbed.

2.1 Linear Temporal Logic

In reactive systems, correctness depends, not only on inputs and outputs of a computation, but on execution of the system. Temporal logic is a branch of logic that incorporates temporal aspects to reason about propositions in time. In this paper, we consider a version of temporal logic called linear temporal logic (LTL) [1].

LTL includes Boolean connectors like negation (¬), disjunction (⊔), conjunction (⊓), material implication (→), and two basic temporal modalities next (•) and until (••). By combining these operators, it is possible to specify a wide range of requirements. Formulas involving other operators can be derived from these basic ones, including eventually (♢) and always (□).

An atomic proposition is a statement on system variables v that has a unique truth value (True or False) for a given value v. For a set π of atomic propositions, any atomic proposition p ∈ π is an LTL formula. Given a propositional formula describing properties of interest, widely used temporal specifications can be defined in terms of their corresponding LTL formulas as follows. A safety formula asserts that a property will remain true throughout the entire execution (i.e., nothing bad will happen). A response formula states that at some point in the execution following a state where a property is true, there exists a point where a second property is true. A response formula is used to describe how systems need to react to changes in environment or operating conditions. A response property, for example, can be used to describe how the system should react to a generator failure: if a generator fails, then at some point a corresponding contactor should open [12], [13].

2.2 Reactive Synthesis

A system consists of a set V of variables. The domain of V, denoted by dom(V), is the set of valuations of V. Let E and P be sets of environment and controlled variables, respectively. Let s = (e, p) ∈ dom(E) × dom(P) be a state of the system. Consider a LTL specification ϕ of assume-guarantee form ϕ = ϕe → ϕs, where, roughly speaking, ϕe characterizes the assumptions on the environment and ϕs characterizes the system requirements. LTL formulas are interpreted over infinite sequences of states, where s0, s1, s2, . . . is an infinite sequence of valuations of environment and controlled variables. The synthesis problem is then concerned with constructing a strategy in the form of a partial function f : (s0s1 . . . st−1, et) → pt, 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 ϕs as long as the environment satisfies ϕe.

For general LTL, the synthesis problem has a doubly exponential complexity [7]. A subset of LTL, namely, generalized reactivity (GR(1)), can be solved in polynomial time (polynomial in the number of valuations of the variables in E and P) [6]. GR(1) specifications restrict ϕe and ϕs to take the following form, for α ∈ {e, s},

ϕα := ϕα init ∧ E∈I e ϕα E,i ∧ P∈I p ϕα P,i,

where ϕα init is a propositional formula characterizing the initial conditions; ϕα E,i are transition relations characterizing safe, allowable moves and propositional formulas characterizing invariants; and ϕα P,i are propositional formulas characterizing states that should be attained infinitely often. For the specifications considered in this paper, the safety fragment of GR(1) suffices.

Given a GR(1) specification, the digital design synthesis tool implemented in JTLV (a framework for developing temporal verification algorithms) [8] generates a finite-state automaton that represents a switching strategy for the system. TüLiP provides an interface to JTLV.

2.3 Testbed Specifications

Consider the single-line diagram in Figure 1 in which environment variables are health statuses of generators and rectifier units, and controlled variables are the state of contactors. Consider also two different controller implementations: a centralized logic that runs the system with a single automaton and a distributed logic that has two different
automata, one for the AC subsystem and one for the DC subsystem, running sequentially.

For the centralized logic, the environment assumptions are: (i) at least one generator must always be healthy, and (ii) at least one rectifier unit must always be healthy. In LTL, this can be written as

$$\Box((\text{gen}_1 = \text{healthy}) \lor (\text{gen}_2 = \text{healthy})) \land (\Box(\text{ru}_1 = \text{healthy}) \lor (\Box(\text{ru}_2 = \text{healthy}))),$$

where \( \text{gen}_1, \text{gen}_2, \text{ru}_1, \) and \( \text{ru}_2 \) are health statuses of the two generators and the two rectifier units, respectively. To ensure non-parallelizing of AC sources, we disallow any configuration of contactors in which a path may be created between the two generators. The contactors \( \text{c}_1 \) and \( \text{c}_2 \) are below the generators in Figure 1, and \( \text{c}_3 \) is between the AC buses. Therefore, contactors \( \text{c}_1, \text{c}_2, \) and \( \text{c}_3 \) can never be closed at the same time. This is written as

$$\neg((\text{c}_1 = \text{closed}) \land (\text{c}_2 = \text{closed}) \land (\text{c}_3 = \text{closed})).$$

The last specification ensures that all buses can be unpowered for no more than a time \( T \). The limit that unpowered time can be set to depends on timing characteristics of the testbed, which is explained in Section 4.1. To synthesize centralized logic, we used the assumption that this time is zero; thus, the specifications that all buses \( b_i \) fulfill \( \Box(b_i = \text{powered}), \) for \( i \in \{1, 2, 3, 4\} \) can be set.

To synthesize distributed logic, we separate the system into two subsystems, seen in Figure 1. The AC subsystem contains all AC components (generators, AC contactors, AC buses, and loads). The DC subsystem contains all rectifier units, DC contactors, buses, and loads. All specifications from the centralized case decompose and carry over to the distributed case. However, in order to ensure that the overall specification is realizable, we impose additional restrictions on the components located at the interface between subsystems. The rectifier units contain capacitors that can be chosen so that they create a delay \( T_{RU} \), in which the DC buses stays powered even after that an AC bus gets unpowered.

If \( T_{RU} > T \) the additional interface refinement comes in the form of a guarantee specification that all DC buses \( b_i \), for \( i \in \{1, 2\} \) will always be powered \( \Box(b_i = \text{powered}) \), provided that both rectifier units stay healthy, i.e.,

$$\Box((\text{ru}_1 = \text{healthy}) \land (\text{ru}_2 = \text{healthy})).$$

This guarantee is written as an environment for the DC subsystem. With this refinement, both subsystems can be synthesized independently, and the overall system specifications are satisfied when they are implemented together. We assume that the time a generator remains healthy is not arbitrarily short so that the AC bus powered time (i.e., the time between two intervals when AC bus is unpowered) is large enough to keep the capacitors on rectifier units charged.

### 2.4 Implementing Formal Specifications

TuLiP generates finite-state automata in the form of a text file that enumerates the possible states of the system and how the transitions could be carried out according to the current state. It also generates a text file that specifies environment variables (e.g., generators and rectifier units) and system variables (e.g., contactors). In order to implement the control logic in SimPowerSystems, we automatically translate these files into a MATLAB-compatible script. A preliminary solution uses a Python script for this translation. A

![Sample code generated using TuLiP controller shown in Figure 2.](http://tulip-control.sf.net)

Figure 2: Sample of a TuLiP output in two-generator and three-contactor case. The generator status variables are \( \text{gen}_1 \) and \( \text{gen}_2 \), and the contactor status variables are \( \text{c}_1, \text{c}_2, \) and \( \text{c}_3 \). Each state has successors, which define where the controller can transit depending on current state. In addition, no-successor states exist.

### 3. DESIGN AND IMPLEMENTATION

The single-line diagram in Figure 1 is a simplified notation for representing a three-phase power system. However, as described in Section 3.1, power supply to the hardware testbed is not three-phase. In order to represent the installations of the sensors, circuit protection devices, and fault-injection switches, we present a detailed schematic of the testbed in Figure 6. Descriptions of the components shown in Figure 6 are given in Figure 7.

The hardware testbed has two different voltage levels: 24 VAC and 2.5 VDC. The DC section is connected to the AC section by rectifier units. Aircraft contactors are designed to switch three-phase electric power with relatively high currents. Relays are generally used for switching lower currents. These operate in a similar fashion to contactors but are lighter, simpler, and less expensive. Therefore, it was more convenient to handle the switching in the hardware.
global state;
while 1 do
gen1 = readgen1();
gen2 = readgen2();
[c1, c2, c3] = mscript(gen1, gen2);
writeboard(c1, c2, c3);
end while

Figure 4: Code that implements the control software running on hardware model.

Figure 5: Hardware setup corresponding to the single-line diagram shown in Figure 1.

model with relays. It was possible to connect the control logic to the relays with the use of a relay board2, which is a set of computer-controlled relays that can communicate with programming languages supporting serial communications, e.g., MATLAB. Analog-to-digital (A/D) connections on the relay board are used to monitor the system conditions. A photo of the setup3 is shown in Figure 5. The transformers in Figure 5 are connected to power cords; these can be unplugged to simulate a generator failure. The rectifier units are connected to a switch, which can be used to generate a fault on the DC subsystem. Next, we describe how we monitor and sense the status of generators and rectifier units.

3.1 Generation and Circuit Protection
Each generation unit consists of a 12 V battery connected to an inverter that generates 120 VAC; that is then transformed down to 24 VAC to ensure safety. If the controller violates one of the safety requirements and connects these two sources in parallel, it would result in a short-circuit and cause the fuses installed next to the generators, shown in Figure 6(a), to blow. This observation makes it possible to monitor the correctness of the controllers at run time.

3.2 Sensing
The relay board needs to react consistently to faults injected into the system; this requirement implies that sensor placement, functionality, accuracy, and time delay play crucial roles in design. Two types of faults can be injected in the system, namely, rectifier unit failures and generator failures. Voltage sensing for generator failures is handled using additional relays. These relays close a 3.6 V circuit to a battery when triggered by the voltage from the transformers. If a fault occurs and a generator does not work properly, the 3.6 V circuit opens and the system reacts accordingly. The voltage sensors of the rectifier units are directly connected to the A/D ports of the relay board because the voltage can be tuned to the appropriate value using an adjustable output on the rectifier units. Figure 6(b) illustrates the sensing configuration on the testbed.

4. EXPERIMENTS
We next describe the characteristics of the hardware testbed and show some preliminary test runs with different control architectures.

4.1 Testbed Characteristics
The first step before the implementation and testing of different controllers is characterizing the timing properties of the hardware testbed. Every relay has a time delay between the time a command is sent by the computer and the time an action (i.e., relay opening or closing) is taken, this is referred to as the relay delay time, $T_d$. Furthermore, the system has delays resulting from control cycle times, $T_c$ and $T'_c$, defined as

$$T_c = T_r + T_I + T_w,$$

$$T'_c = T_r + T_I,$$  \hspace{1cm} (2)
where $T_r$ is the time it takes to read the health statuses from all of the four environment variables, $T_I$ is the time it takes to run the logic (the time can be interpreted as the time taken to run the code shown in Figure 3), and $T_w$ is the time it takes to write information to the board (see Figure 4). Writing information to the board is not needed in every iteration (for instance, if the system state remains the same), therefore the control cycle time also include $T_w$.

The control cycle times $T_c$ and $T_c'$ are listed in Table 1. The relay delay time can be found from the board specifications and shall be less than 20 ms.

An important safety requirement in an aircraft is that a bus should never lose power for more than a certain duration, e.g., typically 50 ms. In the hardware testbed, the time for which the bus is unpowered depends on the control cycle times and the relay delay time, and because the control cycle times exceed 50 ms, we cannot use the typically specified time for which an aircraft can be unpowered. Therefore, it was necessary to adopt a suitable limit. As illustrated with two environment variables in Figure 4 the relay board read the health status from each environment variable in a specified order. It is therefore necessary to include a part of $T_r'$ from the previous control cycle in this limit. The time $T_I$ in Equation (2) is negligible compared to $T_r$ and $T_w$, the time taken to read the health status from one environment variable can therefore be approximated as $T_r'/4$. A reasonable value of an acceptable unpowered time for the hardware testbed can be

$$T \approx \max (T_d) + \max (T_e) + \frac{4 - n}{4} \cdot \max (T_r'),$$

where $n \in \{1, 2, 3, 4\}$ is the number which denotes the order of when the environment variable that is faulty is read in the code.

### 4.2 Controller Tests

Two controllers were tested, one with distributed logic and one with centralized logic. The controller with centralized logic had a 16-state automaton synthesized as explained in Section 2.3. The controller with distributed logic had two four-state automata that run on each subsystem. Both of these automata were synthesized in a similar fashion to the 16-state controller.

If the environment-related assumption is violated, the controller may end up in a state with no outgoing transitions, referred to as the no-successor state. The environment-related assumptions for the testbed are expressed in Equation (1) of Section 2.3. A violation of Equation (1) results in the controller entering a no-successor state, which happens when both generators or both rectifier units are faulty. If a centralized controller senses that both rectifier units are faulty, the whole system stops working because a no-successor state has been reached. This is not the case when distributed logic is used, because the AC system continues working even if the DC environment assumption is violated and the DC part reaches a no-successor state. The distributed logic implementation has two different automata that represent the logic, one for each subsystem, with coupling between them. However, the distributed logic is centralized in that it consists of single control software running on a single computer and communicating with the hardware through a single channel.

Figure 8 shows the voltage measurement for the centralized 16-state controller. The measurement was taken on the AC bus when the generator, which health status is read at second place ($n = 2$ in Equation (3)) of the four environment variables in the code, was switched off and then on again. The generator was switched off at $t = 2.83$ s, at which point the bus becomes unpowered. The second vertical line from the left indicates when the controller reacts and power up the bus using the other generator, which happens at $t = 3.1$ s. The generator was switched on again at $t = 3.73$ s; this was accompanied by a discernible change in the sine curve. Once a generator is switched on again after a fault, the time for which the bus is without power is not noticeable because the controller sends simultaneous commands to two relays.

The measured bus-unpowered times are listed in Table 2, which show a maximum value of $T_{\text{max}} = 414.9$ ms. An acceptable unpowered time when $n = 2$ and $\max (T_d) = 20$ ms.
can be calculated with Equation (3). It follows that 
\[ T \approx \max(T_u) + \max(T_c) + \frac{1}{2} \max(T'_u) = 470.35 \text{ ms} \]
and hence, \( T_{max} < T \). We used a digital storage oscilloscope
(Rigol DS1052E 50 MHz) for measurements. The measurement
data are imported into MATLAB for analysis and to
estimate the unpowered times.

5. LIMITATIONS AND EXTENSIONS

As discussed earlier, when we implemented distributed
logic with the hardware model, it was still centralized in
that only one relay board was connected to one computer.
However, it is possible to use two relay boards connected
to two different computers, with each of them controlled by
different automata. The part that contributes the most to
the control cycle times (\( T_u \) and \( T_c \)), is the time it takes to
read data from the board (\( T_u \)); if the controller is operated
with two relay boards, \( T_c \) would be split in half, which would
cause \( T_u \) and \( T'_u \) to decrease significantly. The distributed
control architecture would also be more like that of an air-
craft electric power system.

We injected faults in the hardware testbed by unplugging
the power cords and changing the switches; however, a more
accurate approach to generate faults would be the use of an
additional relay board. Using an additional fault injec-
tion board, we can systematically study synchronous, corre-
lated, and cascaded failures and their influence on controller
performance; with the current method of fault injection, it
could be difficult to switch off a generator and a rectifier
unit within the same control cycle.

On an aircraft, the controller is an embedded system des-
ignated for a specific task. To increase its reliability and
performance, the hardware model could be adapted to run
the relay boards through microcontrollers. Embedded code
for these microcontrollers can be generated using MATLAB.

At last, we want to emphasize the fact that it is entirely
possible to synthesize the controller with another synthesis
tool and test it on the testbed. It was convenient as an initial
demonstration to choose LTL, reactive synthesis, and TuLiP
because they have been applied to electric power systems in
the past [13].

6. ACKNOWLEDGMENTS

The authors wish to acknowledge the funding from MuSyC, the Boeing Corporation, and AFOSR (award #
FA9550-12-1-0392), and thank Rich Poisson from Hamilton-
Sundstrand for helpful discussions about the development of
the hardware testbed.

7. REFERENCES

autonomous cars in a robotic urban-like environment. In IEEE Intl. Conf. on Robotics and Automation,
power systems of spacecraft and aircraft. In Innovative Applications of Artificial Intelligence Conference,
pages 1699–1705, Chicago, IL, 2008.
power allocation for vehicle management systems. In IEEE Conference on Decision and Control, 2011.
reactive(1) designs. Verification, Model Checking and Abstract Interpretation, 3855, 2006.
are hard to synthesize. In IEEE Symposium on
developing verification algorithms. In Intl. Conf.
D. Hall, C. Lee, et al. Advanced diagnostics and
prognostics testbed. In International Workshop on
Murray. An Aircraft Electric Power Testbed for
Validating Automatically Synthesized Reactive Control
Inc., Natick, Massachusetts, 2011.
R. Murray. TuLiP: a software toolbox for receding horizon temporal logic planning. In Intl. Conf. on
reactive protocols for aircraft electric power
distribution. In IEEE Conference on Decision and
Control, 2012.