A Component-based Safe Design method for train control systems

Salam Hajjar, Emil Dumitrescu, Eric Niel

A Component-based Safe Design method for train control systems

Salam HAJJAR, Emil DUMITRESCU, Eric NIEL
Laboratoire Ampère, Université de Lyon, INSA de Lyon
[fistname.lastname]@insa-lyon.fr

Abstract: In this paper, Formal verification and Discrete controller synthesis are combined within a component-based design method. Formal verification finds design errors and provides counter-examples. The Discrete Controller Synthesis technique attempts to enforce previously verified specifications which do not hold. It automatically produces control code, which is correct by construction with respect to the specification to enforce. This approach is presented and illustrated on an industrial railway system.

Key words: Discrete controller synthesis, formal verification, supervisory control, embedded systems, component-based design.

1. Introduction

Design errors have serious consequences when they appear in critical control systems, such as robotized plants, energy production plants, or transport systems. Such errors may cost human lives, or at best, a large amount of money to redesign the system. Such systems are often modeled as synchronous reactive systems, by using communicating finite-state machines. They call for *safe* design methods and techniques, ensuring functional correctness against a set of specifications. Besides simulation, the *property checking* formal verification technique is unavoidable for discovering subtle bugs, usually known as corner-case configurations, which are very difficult to uncover by simulation. However, the designer must correct these errors manually, which is an error-prone process in general. It is usual that by attempting to manually correct an error, another error is introduced, which creates a vicious circle situation. However, this design process allows the progressive creation of reusable building blocks considered as "certified": enough time has been spent in design/verification, without finding any more significant errors. Such re-usable blocks can be managed internally, or acquired from commercial vendors; they are known generically as commercial off-the-shelf (COTS) components. A COTS encompasses both a behavior, and a set of specifications. COTS can be assembled according to their specifications, in order create new behaviors, satisfying more complex specifications, and gain a lot of time through code reuse. However, even though each of them has been individually and thoroughly verified, they are rarely designed to perfectly operate together. Design errors can appear by simply assembling a collection of correct COTS and their manual correction is likely to be error-prone and time-consuming. Thus, COTS integration can remain an expensive task. The Discrete Controller Synthesis (DCS) technique is an emerging solution able to build correct-by-construction designs, by automatically generating a *controller*. It was first proposed by Ramadge and Wonham in 1989 [1], to generate controllers for manufacturing plants. DCS seems a promising approach for automatically producing correct designs, or correcting design errors. In this paper, we propose a component-based design approach for safe design of COTS-based hardware systems. This approach uses property checking in synergy with discrete controller synthesis. We demonstrate the validity of this approach on an example: the door control system of a train.

The rest of this paper is organized as follows; section 2 briefly recalls the formal verification basic concepts. Section 3 explains the discrete controller synthesis technique. Section 4 introduces the safe design method for component-based hardware systems. Section 5 illustrates our method on a train door control system. Section 6 describes the application of the proposed method and the results obtained. Finally, in section 7 we recall few former works related to ours

2. The property checking technique

This technique checks the behavior of a design against a formal specification written in temporal logic [2]. The design is modeled by a set of communicating finite state machines. The verification algorithm performs a symbolic exhaustive search inside the state space of the studied model, and returns the set of states satisfying the specification. Property checking is more powerful than simulation since it verifies the system against all its possible input values instead of only selected scenario. In addition [3], it is able to provide a diagnostic counterexample in case a specification is violated. However, its performance decreases dramatically with the size of the design under verification because of its exponential complexity in the number of design's variables. Property checking is suitable for verifying small/medium-sized designs, in order to find corner-case errors [4].

3. Discrete Controller Synthesis

3.1. Preliminary

Given a design M and a formal specification P expressing safety (possibly in temporal logic), the DCS technique attempts to build a supervisor SUP, which, once composed with M, guarantees the invariance of P as shown in figure1. The satisfaction of P is considered within a "game", where the supervisor, if it exists, plays against the environment; at each moment it is able to implement a "non-losing" strategy, preventing M to reach a state where P does not hold. The input set of M is divided into two disjoint subsets: controllable (X_c) and uncontrollable (X_{uc}) inputs. Controllable inputs are driven by the supervisor, whereas the uncontrollable inputs are driven by the environment. The DCS technique operates in two steps as show in figure2. First, an invariant under control set IUC is built; as long as M stays inside IUC, the game cannot be "lost": states not satisfying P cannot be reached. This is performed by selecting controllable values which always lead to IUC, whatever the uncontrollable values provided by the environment. The second step constructs the supervisor SUP, as the set of all transitions leading to IUC.

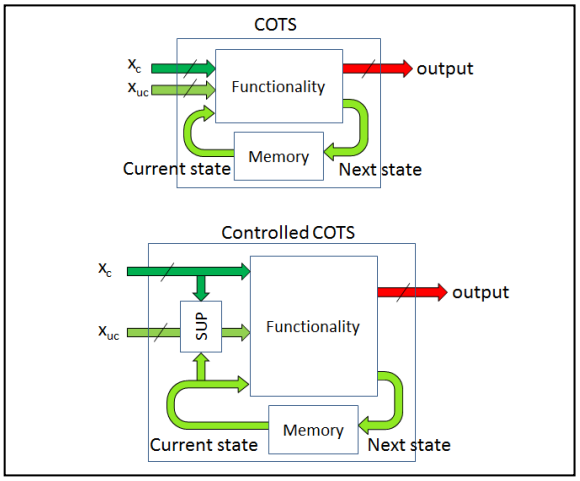

Figure 1 controller adding to a COTS design

3.2. *Notation*

A discrete event system can be modeled as a combination of finite state machines, *FSM*. Each of them represents a component COTS. We denote a *FSM* of a *DES* as a tuple G = {s_0, S, X, Y, T, O}; where s_0 is the initial state, S is a finite set of Binary state variables, X is the set of Binary input variables, $X = X_c \cup X_{uc}$, Y is set of Binary output variables. T is the transition function:

$$T: B^{|s|} \times B^{|Xc|} \times B^{|Xuc|} \times B^{|s|} \rightarrow B \quad (1)$$

O is the output function:

$$O: B^{|s|} \times B^{|Xc|} \times B^{|Xuc|} \rightarrow B^{|y|} \text{ (Mealy machine)} \quad (2)$$

$$\text{Or } O: B^{|s|} \rightarrow B^{|y|} \text{ (Moore machine)} \quad (3)$$

We denote the set of invariant under control as follows:

$$IUC^0 = \{s \mid P \text{ holds in } s\} \quad (4)$$

$$IUC^{i+1} = \{s \in IUC^i \mid \forall x_{uc} \exists x_c : T(s, x_c, x_{uc}, s') \rightarrow s' \in IUC^i\} \quad (5)$$

A supervisor does exist if *IUC* is not empty and contains s_0. The supervisor is the set of transitions denoted as follows:

$$SUP = \{(s, x_c, x_{uc}, s') \mid \forall x_{uc} \exists x_c \mid s' \in IUC\} \quad (6)$$

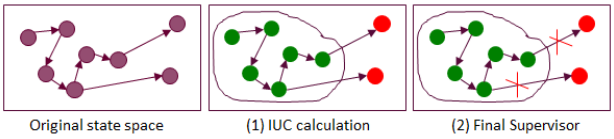

Figure 2 controller calculating for a DES

4. Safe component-based design

Figure 3 presents our design flow. In order to build a new control function, formally specified by a temporal logic assertion *spec*, the designer describes the functional behavior of the components by communicating, concurrent, synchronous, finite state machines, and instantiate these models in the design. Two or more COTS can be composed together, according to the new specifications to satisfy. This task amounts to a synchronous composition between FSMs. The composite design is formally verified against *spec*. If the verification passes, the newly obtained design can be inserted in the COTS library as a new reusable component. Otherwise, the property checking tool produces a counter-example. By visually analyzing the counter-example, the designer can isolate a set of *suspect* input variables, involved in the failure of *spec*. These variables are chosen as controllable candidates. A DCS step attempts to build a supervisor which attempts to enforce *spec* by controlling the variables previously chosen. If a supervisor exists, it is implemented as a set of control functions f [5]. Otherwise, the designer can attempt to enlarge the set of controllable variables and retry a DCS application. Sometimes DCS produces very restrictive supervisors which ensure *spec* by disabling most interesting behaviors. This is why the last simulation step is required in order to dynamically validate the newly obtained control solution. If no satisfying solution is possible, the designer must either modify the original design of a component or give up some properties and validate the composition with certain properties. In the following, this design method is illustrated on an example.

5. Passenger's Access system

The Passenger's system consists of six components separately prebuilt. Components interact with the environment and between each other via sensors, push buttons, switches and internal signals. Fig.4 illustrates only the internal signals between components to avoid complicating the figure. Components are illustrated in table1.

Table 1 Passenger's Access omponents

Num	COTS name	Functionality
COTS1	z-switcher	Controls mainly a 3s timer
COTS2	door controller	controls the authorization to open and command the door close
COTS3	ringing signal	announces the imminence of closing inside the driving cabin
COTS4	light signal	indicates the authorization of opening the doors
COTS5	open inhibition	inhibits the open authorization
COTS6	open authorization	allows doors to open

Each one of these components is modeled as a finite state automaton as illustrated in figure.5. We suppose that the internal functionality of each component is correct and no local design errors occur. The global behavior of each assembly of components has to obey some temporal properties.

5.1. Behavior description

In this section, we explain how components behave individually. **COTS1** has three positions; *pos_o* represents the initial position of the component where doors are allowed to be open. *pos_s* intermediate position triggers the ringing signal. *pos_p*, as long as the switch is at this position, the driver can give the order to close doors. Its output is the signal *timer*. **COTS2**, it sends to the physical door the command to close and the authorization to open. Its input signals *p*, *o* correspond to the position of the z-switch, the signal *timeout* is the output signal received from a 3s timer, the signal *closed* indicates closed door sensor. Signal *f_open_aut* is the output of COTS6, which commands the doors to open. **COTS3**, it can be either ringing or silent. Its inputs are the *s, p and timeout* signals explained before, its output signal is *ring*. As soon as it is triggered to ring, it rings for at least 3s after that it stops ringing if it receives a true value on signal *p* from the z-switcher. **COTS4**, this light signal shines when the driver selects a group of doors to open, and fades when the open authorization is inhibited. **COTS5**, receives from the environment signals *pb_cancel_open_aut* and *pb_open_aut* that reflect the states of the push buttons open-authorization and cancel-open-authorization respectively, COTS5 controls the inhibition of open authorization and indicates the selection of a group of doors to be opened or closed. **COTS6** produces the command to authorize door-opening *f_open_aut*. Signals *L_speed*, *H_speed* the speed sensors, speed is less than 0.5 km/h and more than 2km/h respectively.

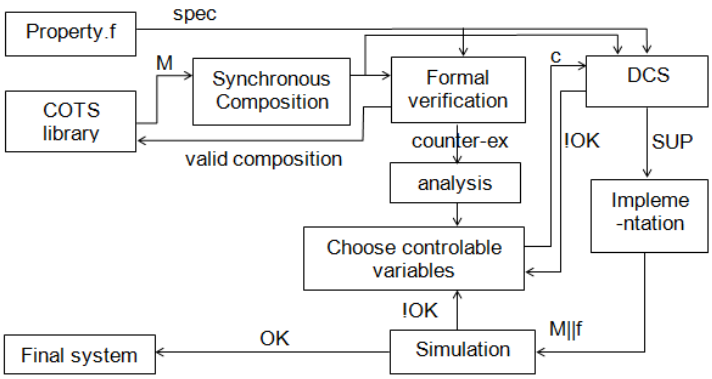

Figure 3 safe component-based design method

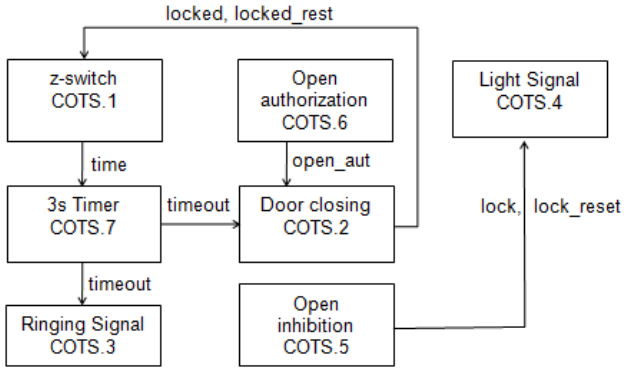

Figure 4 internal components interaction

5.2. Global System behavioral

For safety reasons, we suppose that the initial state of each component is an Idle state, which means the z-switch is in the O_position state, the door controller in state PO, doors are opened, light and ringing signals are turned-off, doors are inhibited of opening and no doors group is selected to be closed/opened. The train driver can first select group of doors to be closed, and then move the z-switch to *pos_s* to announce the imminence of closing, then he moves the switch to *pos_p* to command the closing. When the doors and the filling gaps are safely closed, the driver can then move the vehicle. As soon as the speed of the train passes the high-threshold, (speed > 2km/h) the doors will be inhibited of opening. Until the speed reaches again the

low-threshold (speed < 0.5 km/h) then the doors can be authorized to open. The driver must choose which line door side (left/right) to open regarding the platform in the station.

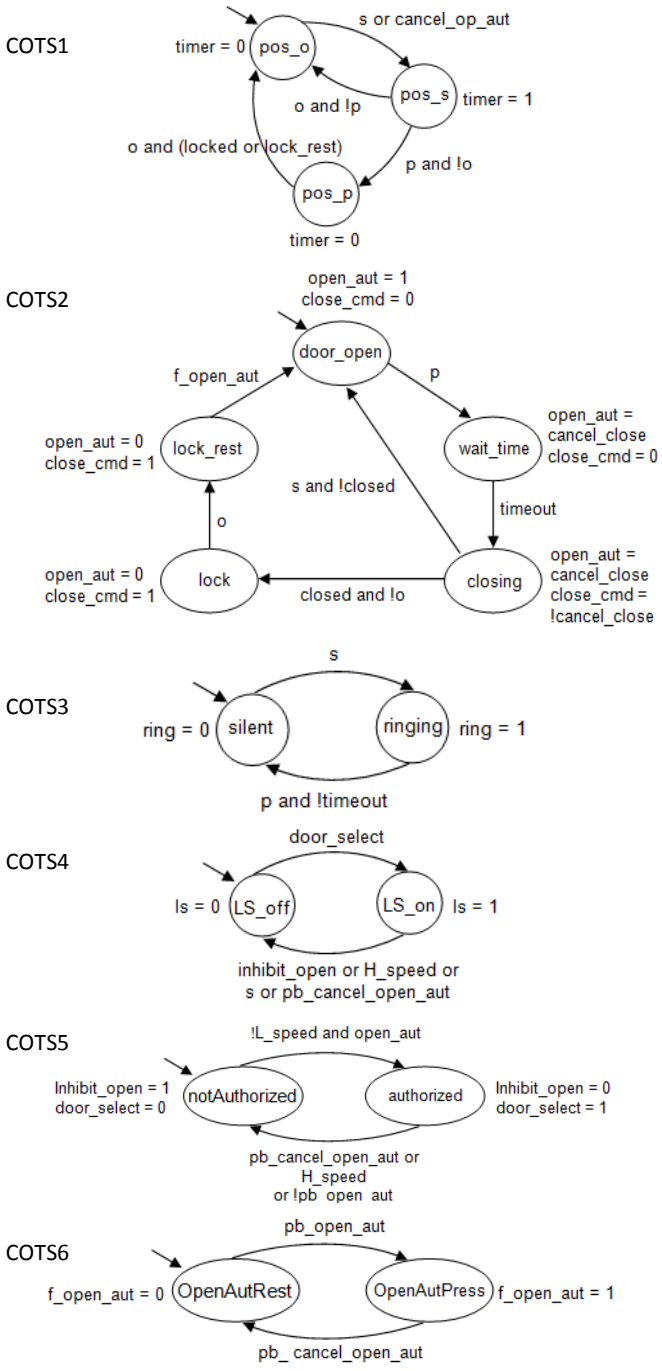

Figure 5 components FSM model

5.3. Functional requirements (properties)

We mention here some safety properties we wish to enforce when combining several COTS each property is expressed in LTL logic.

- *P1*, prevents the states *pos_s* of COTS1 and *silent* of COTS3 to be present at the same moment. *P1*: G!(*pos_s* ∧ *silent*).
- *P2*, when combining COTS3 and COTS4; the light signal must not be shining when the closing signal is ringing. P2: G !(*ringing* ∧ *LS_on*)
- *P3*, when combining COTS1 and COTS4. If z-switch in *pos_s* or *pos_p* this means doors are commanded to close, which provokes the light signal not to be shining. *P3*: G !((*LS_on*) ∧ (*pos_p* ∨ *pos_s*)).
- *P4*, it combines the 3 former properties together *G!((pos_s and silent) or (ringing and LS_on) or ((LS_on and (pos_s or pos_p))))*

6. Safe Design of Passenger's Access

The component-based method is applied to passenger's access system. COTS models are described and assembled using the synchronous language Mode Automata [6] and let operate simultaneously. The formal verification demonstrates that the studied properties are violated by the global system. Tables.2,3,4 show the variable values involved in violating the properties *p1*, *p2*, *p3* respectively. This calls the DCS step to generate controllers, explicitly connected to groups of COTS and heal the concurrency mismatches. Each of these controllers is called *glue*.

Table 2 counter-example P1

State variable	t_0	t_1	t_2	t_3
pos_o	1	1	0	0
pos_s	0	0	1	1
pos_p	0	0	0	0
silent	1	1	0	1
ringing	0	0	1	0
Input variable				
s	1	1	0	1
o	0	0	1	0
p	0	0	1	0
timeout	0	0	0	1

Table 3 counter-example P2

State Variables	t_0	t_1
silent	1	0
ringing	0	1
LS_OFF	1	0
LS_ON	0	1
Input Variables		
s	1	0
p	0	0
timeout	0	0

door_select	1	1
inhibit_open	0	0
H_speed	0	0
pb_cancel_open_aut	0	0

Table3 counter-example P3

State Variables	t_0	t_1
pos_o	1	0
pos_s	0	1
pos_p	0	0
LS_OFF	1	0
LS_ON	0	1
Input Variables		
s	1	1
o	0	0
p	0	0
door_select	1	1
inhibit_open	0	0
H_speed	0	0
pb_cancel_open_aut	0	0

6.1. Glue Automatic generating

The DCS step is achieved by the tool SIGALI [7]. We produce a controller for every assembly of components in order to enforce a specification as shown in table 3. The properties are provided to the DCS tool as a set of states to be made invariant. The process to calculating the controller is realized by the two steps explained in section 3.

Table 4 Assembly of COTS and properties to respect

COTS assembled	**controller**	**Property**
COTS1, COTS3	*Cont.1*	*P1*
COTS3, COTS4	*Cont.2*	*P2*
COTS1, COTS4	*Cont.3*	*P3*
COTS1, COTS3, COTS4	*Cont.4*	*P4*

6.1.1. Hypothesis

The different signals mentioned in section 5.1 are the inputs and outputs of the system, and with respect to its industrial nature, we suppose few hypotheses over the controllability of those signals. We treat the signals as events arrive to the system

- All events represent a push button action or a switch action, are controllable.
- All signals received from sensors are uncontrollable.
- All output signals are uncontrollable.
- All signals, which are outputs of some COTS and inputs for other COTS, are uncontrollable.

Over these assumptions, we distinguish two sets of input signals for the passenger access system, illustrated in table.4.

Table 5. Controllable and uncontrollable inputs

Controllable Input	**Uncontrollable Input**
s	*closed*
p	*H_speed*
o	*L_speed*
pb_cancel_open_aut	
pb_open_aut	

6.2. The correct circuit

We produce one controller, named glue, for every group of components that cooperate in order to realize a task or to form more complex component and enforce one or multiple functional requirement related to the global functionality of assembled COTS. The properties are provided to the DCS tool as a bunch of forbidden states.

The produced controllers prevent the system from reaching the undesired states. We mention four cases illustrating the robustness of controllers w.r.t the specified properties.

- *Cont.1:* In order to respect *p1*, when the system in a situation where COTS1 *pos_s* and COTS2 at *Ringing* state, and if signals *p* and *timeout* are both true, the controller forces the controllable o to false, so the system does not reach the undesired case (*pos_s and silent*)
- *Cont.2: P2* aims to prevent the system from reaching the case (*ringing and LS_on*), the controller enforces signal *p* to true value when the signal *door_select* is true, so the undesired situation will not be reached.
- *Cont.3: P3* aims to prevent the system to reach the case *(LS_on and (pos_s or pos_p))*. Whenever the system in state *(pos_s and LS_off)* the controller enforces values false, true to signals *p, o* respectively to achieve property respecting.
- *Cont.4:* When assembling the original components COTS1, COTS3, COTS4 and generating the controller cont.4, the simulation step shows that this controller is very restrictive; it blocks the system in the states where position *pos_p* holds. This means, we achieve the goal of respecting the three properties together, but we lose the functionality of the system. We have tried to enlarge the list of controllable variables but we keep receiving a blocking controller. In such dead case, we advise the designer to modify the original design of the components.

7. Related Work

A similar work has been proposed in [5] a discrete serial to parallel converter was studied, where the original plant specification faces a date lose problem, during the transformation process between two components of the system. A discrete controller was automatically generated to avoid this loose, by forcing the first component to wait a signal, from the second one, to indicate the capability to receive new data. In [8] authors have applied the controller synthesis technique to realize safe functionality of a power transfer station, and to provide a best quality of service. In [9] authors solve the compatibility problem between software components (two communicating protocols). They propose a refinement relation, generated by a converter, to ensure the satisfaction of a property. They demonstrate that the existence of such a relation is necessary and sufficient to confirm the compatibility between communicating components. The solution is based on the idea of defining enforceable events among the controllable events, buffer those enforceable events and then use them to enforce transitions. The converter they propose can realize three missions: 1) disabling a transition. 2) Forcing a transition. 3) Buffering events to be used latter. Whereas the controller calculated by the DCS can only disable the undesired transitions. A question that one can ask: how the designer can choose the enforceable events. In [10] authors use the Fractal/Cecilia framework to implement a reconfigurable Comanche HTTP Server they decompose the system into Hardware and software components, and use the Heptagon synchronous language and its extension BZR to model the components and the properties respectively. The studied properties are related to the system reconfiguration and quality of service. It was not clearly mentioned the use of Sigali to calculate the controller, but to our knowledge, BZR is linked to Sigali tool to calculate such controllers. Therefore, a manager (controller) is calculated, in C code, and then wrapped to the components and integrated into the distributed environment.

8. Conclusion and perspectives

In this article, we present a compositional safe design framework over prebuilt, reusable COTS components. The method uses property checking in synergy with Discrete Controller Synthesis for finding and automatically correcting design errors. The method proposed is illustrated on a railway system. The results obtained demonstrate the validity of the proposed method in producing correct by construction designs. Current investigations include possible performance enhancements for the DCS step, in order to tackle its theoretic exponential complexity. We aim in future work synthesize livness properties by using the optimal controller synthesis method

Acknowledgment

Authors would like to thank the community FerroCOTS, registered in i-Trans project, for financial support. We are grateful to Ken McMillan, Eric Rutten and Gwenaël Delaval for their help.

***References*:**

[1] P. J. . Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[2] A. Pnueli, "The Temporal Semantics of Concurrent Programs," in *Proceedings of the International Sympoisum on Semantics of Concurrent Computation*, 1979, pp. 1–20.

[3] O. Laurent, P. Michel, and V. Wiels, "Using Formal Verification Techniques to Reduce Simulation and Test Effort," in *Proceedings of the International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity*, 2001, pp. 465–477.

[4] J. Yang, P. Twohey, D. Engler, and M. Musuvathi, "Using model checking to find serious file system errors," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, San Francisco, CA, 2004, pp. 19–19.

[5] E. Dumitrescu, M. Ren, L. Pietrac, and E. Niel, "A supervisor implementation approach in Discrete Controller Synthesis," in *IEEE International Conference on Emerging Technologies and Factory Automation, 2008. ETFA 2008*, 2008, pp. 1433-1440.

[6] F. Maraninchi and Y. Rémond, "Mode-Automata: About Modes and States for Reactive Systems," *IN EUROPEAN SYMPOSIUM ON PROGRAMMING*, 1998.

[7] H. Marchand, P. Bournai, M. LeBorgne, P. LeGuernic, and P. Le Guernic, "Synthesis of Discrete-Event Controllers based on the Signal Environment," *IN DISCRETE EVENT DYNAMIC SYSTEM: THEORY AND APPLICATIONS*, vol. 10, no. 10, p. 325--346, 2000.

[8] H. Marchand and M. Samaan, “Incremental Design of a Power Transformer Station Controller Using a Controller Synthesis Methodology,” *IEEE Trans. Softw. Eng.*, vol. 26, no. 8, pp. 729–741, Aug. 2000.

[9] P. Roop, A. Girault, R. Sinha, and G. Goessler, “Specification Enforcing Refinement for Convertibility Verification,” in *Proceedings of the 2009 Ninth International Conference on Application of Concurrency to System Design*, 2009, pp. 148–157.

[10] T. Bouhadiba, Q. Sabah, G. Delaval, and É. Rutten, “Synchronous Control of Reconfiguration in Fractal Component-based Systems -- a Case Study,” 30-May-2011. [Online]. Available: http://hal.inria.fr/inria-00596883. [Accessed: 16-Jun-2011].