Safe Design Method of Embedded Control Systems based on COTS

Salam Hajjar, Emil Dumitrescu and Eric Niel

INSA de Lyon, Lyon, FRANCE
firstname.lastname@insa-lyon.fr

Abstract

In this paper, we propose an approach based on formal verification and discrete controller synthesis that are combined within a component-based design method. Formal verification finds design errors and provides counterexamples while 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 a train controller subsystem.

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 model checking (formal verification) technique is unavoidable for discovering subtle bugs, usually known as corner-case configurations, which are very difficult to uncover by simulation. The model checking technique was used in the industrial domain to detect design errors of avionic systems and . 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 to 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, 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 model checking in synergy with discrete controller
synthesis. We demonstrate the validity of this approach on an railway control system. The
discrete controller synthesis has been suggested to synthesize some temporal properties over
toy robot [1], and to solve mismatches between interacting protocols [18]. An optimization over
the DCS method has been proposed in [10].

The rest of the paper is organized as follows: In section 2 we define the notion of COTS.
Section 3 briefly recalls the model checking basic concepts. Section 4 explains the discrete
controller synthesis technique. Section 5 introduces our safe design method for component-
based hardware systems. Section 6 illustrates the method on a train control system, it presents
an application of the proposed method and the results obtained. Section 7 mentions some
related work. Section 8 concludes the article.

2 Definition

A control COTS can be found in a generic COTS library. It is characterized by four elements:
(1) COTS interface (I). (2) environment assumptions, or sometimes named preconditions (A).
(3) behavioral guarantees, or sometimes named post-conditions (G). (4) COTS functional be-
havior (M). Given C, a control component in a hardware embedded system, with (n) input and
(m) output, we define its elements as follows: a COTS interface (I) is the list of inputs and
outputs though which the component is connected to the physical environment, as illustrated
in Fig 1.

Figure 1: COTS architecture, control part and physical part

Many hardware COTS have a discrete event behavior which can be described as a discrete
event system DES, it can be modeled as a set of synchronous finite state machines (FSM)
with outputs. A FSM is defined as a tuple $M = \{ q_0, X, Q, \delta, PROP, \lambda \}$ where:

- $q_0$ is the initial state;
- $Q$ is a finite set of Binary state variables;
- $X$ is the set of Binary input variables;
- $X = X_c \cup X_{uc}$ \footnote{$X_c, X_{uc}$ are the sets of controllable and uncontrollable inputs, explained in section 5};
- $PROP$ is set of Binary output propositions;
- $\delta$ is the translation relation in the Binary domain
  $$\delta : B|q| \times B|x_c| \times B|x_{uc}| \times B|\delta| \rightarrow B|PROP|$$
• \( \lambda \) is the output function:
  \[
  \lambda : B^{|s|} \times B^{|x_c|} \times B^{|x_{uc}|} \rightarrow B^{|PROP|}
  \]
  Mealy machine
  
  \[
  \lambda : B^{|s|} \rightarrow B^{|PROP|}
  \]
  Moor machine

### 3 The model checking technique

This technique checks the behavior of a design against a formal specification written in temporal logic [12]. 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.

Model checking is more powerful than simulation since it verifies the system against all its possible input values instead of only selected scenario. In addition, it is able to provide a diagnostic counterexample in case a specification is violated. This counterexample contains all the system inputs that are involved in the property violation and chains of their values, starting from the system initial state till the error state. However, its performance decreases dramatically with the size of the design under verification because of its exponential complexity in the number of design variables. Model checking is suitable for verifying small/medium-sized designs, in order to find corner-case errors [reference]. Although it can find all the design errors, but it leaves a harder mission which is correcting these errors to the designer.

### 4 Discrete controller synthesis

Given \((M)\) a FSM modeling a system, \((P)\) a formal specification expressing safety (possibly in temporal logic), the DCS technique attempts to build a supervisor \(\text{SUP}\), which, once composed with \(M\), guarantees the invariance of \(P\). The satisfaction of \(P\) is considered within a game, where the supervisor, if it exists, plays against the environment; at each moment 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. First, an invariant under control set \(\text{IUC}\) is built; as long as \(M\) stays inside \(\text{IUC}\), the game cannot be lost: states not satisfying \(P\) cannot be reached. This is performed by selecting controllable values which always lead to \(\text{IUC}\), whatever the uncontrollable values provided by the environment. The second step constructs the supervisor \(\text{SUP}\), as the set of all transitions leading to \(\text{IUC}\). We integrate this technique in the design method to correct automatically the design errors detected in the model checking techniques.

Regarding the formalization mentioned above we define the invariant under control set as the fixed point of the equation:

\[
\text{IUC}_0 = \{q \mid P \text{ holds in } q\}.
\]

\[
\text{IUC}_{i+1} = \{q \in \text{IUC}_i \mid \forall x_{uc} \exists x_c : \delta(q, x_c, x_{uc}) \rightarrow q' : q' \in \text{IUC}_i\}.\]

A supervisor does exist if \(\text{IUC}\) is empty or does not contain \(q_0\).

The supervisor is the set of transitions denoted as follows:

\[
\text{SUP} = \{(q, x_c, x_{uc}, q') \mid \forall x_{uc} \exists x_c : q' \in \text{IUC}\}
\]

The solution \(\text{SUP}\) obtained above is a relation, represented as a characteristic equation. A final step implements the supervisor \(\text{SUP}\) by transforming it into a set of control functions, one for each \(x_c\), according to the technique described in [9]. The DCS technique has been suggested to synthesis some temporal properties over toy robot [11], and to solve mismatches between interacting protocols [13].
5 Component-based design flow

As mentioned above, each COTS has its local environment assumptions (A) and local guarantees (G). The fact of connecting COTS to each other can entail incorrect global behavior of the assembly, because some local assumptions of a COTS conflict some guarantees of another COTS and vice versa. For example, if \( G_2 \) conflicts \( A_1 \) in figure 2, a global design error appears when assembling the COTS despite the correct local behavior of each individual COTS.

Designing a safe system based on COTS means assembling the components and ensuring that the assembly operates correctly, i.e., the assembly satisfies certain defined global properties and preserves the local guarantees of each individual component. Figure 3 presents our design flow. The design method is a semi-automatic method which had two main goals. The first one is to provide a formal COTS library which contains not only the executable model of the component with a textual documentation, but also a formal representation of the component’s functional requirements. The second goal is to construct hardware control systems based on prebuilt COTS components. The first step of the design method is selecting the needed component(s) and formalizing the environment assumptions, the behavioral guarantees and additional new properties. Two or more COTS can be composed together, according to some new specifications to satisfy. This task amounts to a synchronous composition between FSMs, in the cases where the final system will be implemented on chips like FPGA or ASIC. In the second step the composite design is formally verified against a specification spec in order to detect the design errors. If the verification passes, the newly obtained design can be inserted in the COTS library as a new reusable component with its formal documentation for the environment assumptions, behavioral guarantees and additional properties. Otherwise, the model checking tool produces a counterexample which means a property under verification is violated. If the detected error is related to a liveness property the designer must correct this error manually. But, if the error is related to a safety property, the designer passes to an automatic correction step based on the DCS method. As said before the DCS tool needs two sets of inputs (controllable, uncontrollable inputs). As this tool is still a research demonstration tool, the set of controllable inputs is chosen by the researchers by testing randomly different inputs and choose the set that fits better the synthesis process, i.e., the set with which the DCS tool can provide a correcting solution. In hardware control systems, the designer cannot impose the controllability of the system inputs, since he is working with a real system and some inputs could be impossibly controlled like sensors, we suggest to use the counterexample provided by
the model checking tool. Using the counterexample provided by the model checking tool the
designer can construct a set of controllable signals. As the counter example, provides a set
of only and all the signals responsible of the property violation, it can direct the designer to
choose the signals candidate to be controllable. The model checking tool "Cadence SMV" [15]
that we use for this study categorizes the counterexample signals’ variables in (Input variables,
state variables). To construct the set of controllable signals \( X_c \), the designer should select the
list of input and internal variables in the counterexample list and remove some those signals
what are the signals that are impossible to be controllable. The designer should respect the
following an algorithm to build the set \( X_c \):

**Construction the set of controllable signals \( X_c \)**

1. \( X_c \) initial = the complete set of signals provided by the counterexample.
2. remove from \( X_c \) the uncontrollable signals by following the rules:
3. *Begin removing:*
   4. Every variable of a sensor signal must be eliminated from the \( X_c \) set;
   5. Every variable of data signal must be eliminated from \( X_c \) set;
   6. Every variable of an alert or alarm information must not be eliminated;
   7. Every state variable must be eliminated;
4. *End removing.*
9. All the remaining signals form the set of controllable signals \( X_c \).

This algorithm can be used systematically when the studied system has a hardware nature.

The DCS step attempts to build a supervisor which enforces spec by controlling the variables
previously chosen. If a supervisor exists, it is implemented as a set of control functions \( f \)
and assembled to the original executable model of the components. Otherwise, the designer
concludes that spec cannot be satisfied by this system. Sometimes, the DCS produces very
restrictive supervisors which ensure spec by disabling most interesting behaviors. We propose
to verify some key Liveness properties after the synthesis step to ensure that the system still
achieves certain behavior. We also verify the controller passiveness, i.e, the controller do not
invent events to the system, it only delays or prevents events. Finally, we call a simulation
step in order to dynamically validate the newly obtained control solution. In the following, this
design method is illustrated on an example of building a train controller subsystem over COTS.

6 Case study : Passengers’ access system

The system consists of components separately prebuilt. The components interact with the
environment and between each other via sensors and request buttons as shown in figure[4]. The
control door component controls the opening and the closing of the physical door. It receives
an opening and a closing requests from the train driver \((req.o,d,req.c,d)\) respectively and reads
the values of sensors \( sns.o.d, sns.c.d \) which indicate the full opening and the full closing states
of the physical door. The sensor \( sns.obst \) alerts that an obstacle (a person or an object) is
passing through the door, the information received from this sensor is necessary to prevent the
doors of closing if something is passing into or out of the train. Similarly, the control filling-
gap component, controls the opening and the closing of the physical filling gap. It receives an
opening and a closing requests from the train driver \((req.o.fg,req.c.fg)\) respectively and reads
the values of sensors \( sns.o.fg, sns.c.fg \) which indicate the full opening and the full closing states
of the physical filling-gap. We suppose that the internal functionality of each component
is correct and no local errors occur. The global behavior of each assembly of components has to
obey a safety temporal property which is: the door cannot start opening before the full opening

6.1 Behavioral description of COTS

The door and the filling gap control components’ behavior is represented by finite state machines shown in figure 4. While the train is in mode "stop in the station", the train driver send requests to the door and the filling-gap to open/close. The control components of the door and the filling-gap send commands to the physical environment regarding the driver requests. The FSMS Door/filling-gap control components affect the commands $cmd_{do}, cmd_{dfg}, cmd_{cd}, cmd_{cfg}$ for opening and closing the door and the filling-gap respectively. The Door/filling-gap observer FSMS observes the physical environment of the system. They show that the physical door can be in one of four states ($d_{closed}, d_{opening}, d_{open}, d_{closing}$), and the physical filling-gap can be in one of four states ($fg_{closed}, fg_{opening}, fg_{open}, fg_{closing}$). Passing from one state to another happens only if the transition condition is satisfied.

In order to preserve safe behavior of the system, the door must not start opening before a full opening of the filling-gap. To ensure such a safe behavior a global safety property must be synthesized over the assembly of the components.

6.2 Safe design of the system

The executable model ($M^{exe}$) of the door and the filling-gap are shown in figure 5, they are built using the tool Xilinx StateCAD [19]. To construct the system regarding the method we first formalize the components as following:

The door COTS : $M^d = \{I^d, M^{exe}, A^d, G^d\}$ where:
The filling-gap is modeled as follows: $M_{fg} = \{I_{fg}, M_{exe}, A_{fg}, G_{fg}\}$ where:

- $I_{fg} = \{(req_{o_{fg}}, req_{c_{fg}}, cmd_{o_{fg}}, cmd_{c_{fg}})\}$
- $A_{fg} = \{F(req_{o_{fg}}), F(req_{c_{fg}})\}$
- $G_{fg} = \{F(req_{o_{fg}} \rightarrow cmd_{o_{fg}}), G(req_{c_{fg}} \rightarrow cmd_{c_{fg}})\}$

The components are assembled together. The assembly model is $M_{sys} = \{I_{sys}, M_{exe}, A_{sys}, G_{sys}\}$ where:

- $I_{sys} = \{I_d \cup I_{fg}\}$
- $A_{sys} = \{A_d \cup A_{fg}\}$
- $G_{sys} = \{G_d \cup G_{fg}\}$

The global safety property is modeled in LTL logic as: $p = G \neg (d_{open} \land fg_{closing})$ In the second step of the method, $p$ is firstly verified over the assembly model $M_{sys}$ using a model checking tool Cadance SMV, a counterexample is obtained. It contained the set of the system inputs and outputs which caused the property violation. This set represents the initial list of controllable variables $X_{init} = \{req_{o_{fg}}, req_{c_{fg}}, cmd_{o_{fg}}, cmd_{c_{fg}}, sns_{c_{fg}}, sns_{o_{fg}}, sns_{c_d}, sns_{o_d}, sns_{obst}\}$. We notice that the set contains all the assembly inputs and outputs, this result is expected since the example is relatively small, so all its signals are involved in the property violation. To build the final set of controllable variables, we remove all the variables of the sensors since they represent information which cannot be neither delayed, nor ignored. We remove the variables of the commands since they are outputs of the control components. The final set of controllable variables is $X_c = \{req_{o_{d}}, req_{c_{d}}, req_{o_{fg}}, req_{c_{fg}}\}$. All remaining variables are considered uncontrollable variables.
In the third step in the design method, a controller is automatically generated using the DCS to synthesis the safety property \((p)\) over the assembly model. We use for this study the tool \(\text{Sigali}\) \([4]\) The process of generating the controller can be seen in figure 6.

The controller controls the values of \((\text{req\_o\_d, req\_c\_d, req\_o\_fg, req\_c\_fg})\). If the value received from the environment is the expected value, the controller let it pass to the component, otherwise, it change it, in order to keep the behavior of the system safe. An extract of the real controller of the door-filling-gap system can be seen as follows: if \((\text{sns\_o\_d} \lor \text{sns\_o\_fg})\) then \((\text{req\_c\_fg} = 0)\) else \((\text{req\_c\_fg} = 1)\); if \((\text{sns\_c\_d} \lor \text{sns\_c\_fg})\) then \((\text{req\_o\_d} = 0)\) else \((\text{req\_o\_d} = 1)\);

We assemble the result controller to the system model in order to construct the controlled system as shown in figure 7, the control part of the controlled system is surrounded by a dashed rectangle. In the fourth step we verify the liveness of the controlled system and the passiveness of the controller. To do so we verify the assembly guarantees \(A^{asm}\), and a passiveness properties
The fifth step is the simulation, to accomplish this step, the controller should be translated into VHDL language and assembled to the system executable model (in VHDL) then to be simulated using a simulation tool of hardware system like Xilinx CAD simulator.

Figure 7: The controlled system

7 Related work

The discrete controller synthesis technique has been used in [14] at a generation process for task level controllers, basing on the use of the formal tool SIGALI. In [1] the DCS technique was solicited to enforce a layer of properties over an assembly of components. The work proposed differs mainly of ours as the authors inject additional control inputs to the original design, this idea can barely be adopted in COTS-based design as the COTS design is already built and our goal is to preserve this originality to preserve the re-usability of the components. DSC has been also applied in [2] to construct reliable controllers for arbitrarily large discrete systems. An implementation of the controller generated by symbolic DCS has been proposed in [9] to solve two problems (1) the control non-determinism and (2) the structural incompatibility introduced by symbolic DCS. In [8] authors consider structured programs, as a composition of nodes, and apply DCS on particular nodes of the program, in order to reduce the complexity of the controller computation. An incremental approach for the discrete controller synthesis has been proposed in [17] in order to solve the problem of state space explosion. The idea of controlling concurrent processes (similar to ours) has taken place in [3] where a set of sufficient conditions under which the optimal control could be enforced by a finite automaton. An optimal control was enforced by a decentralized controller.
8 Conclusion

In this article, we present a compositional, semi-formal safe design framework over pre-built, reusable COTS components, it is an enhancement over our former work [11]. The method uses model checking in synergy with Discrete Controller Synthesis for automatically finding and automatically correcting design errors respectively. The method proposed is illustrated on a railway control system and the results obtained demonstrate the validity of the proposed method in producing correct by construction designs. Although the model checking and the discrete controller synthesis methods already exist in the literature, our method is considered the first cooperation between these two techniques, which takes advantages of each one and solves the problem of manual correction after the model checking and the problem of finding the controllable variables which is a basic input for the DCS technique. It is also the first application of the DCS technique on a real industrial system. Current investigations include possible performance to fully automatize the choice of controllable inputs of the DCS in order to reach a full-automatic safe design method starts by a system model and finishes by the implementation code.

References


