Automated Addition of Fault Recovery to Cyber-physical Component-based Models[*]

Borzoo Bonakdarpour
School of Computer Science
University of Waterloo
200 University Avenue West
Waterloo N2L3G1, Canada
borzoo@cs.uwaterloo.ca

Yiyan Lin
Department of Computer Science and Engineering
Michigan State University
3115 Engineering Building
East Lansing, MI 48823, USA
linyiyan@cse.msu.edu

Sandeep S. Kulkarni
Department of Computer Science and Engineering
Michigan State University
3115 Engineering Building
East Lansing, MI 48823, USA
sandeep@cse.msu.edu

ABSTRACT

In this paper, we concentrate on automated synthesis of fault recovery mechanism for fault-intolerant component-based models that encompass a cyber-physical system. We define the notion of fault recovery for cyber-physical component-based models. We also present synthesis constraints that preserve the correctness and cyber-physical nature of a given fault-intolerant model under which recovery can be *added*. We show that the corresponding synthesis problem is NP-complete and consequently introduce symbolic heuristics to tackle the exponential complexity. Our experimental results validate effectiveness of our heuristics for relatively large models.

Categories and Subject Descriptors

D.4.5 [**Operating Systems**]: Reliability—*Fault-tolerance, Verification*; D.4.7 [**Operating Systems**]: Organization and Design—*Real-time and embedded systems*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*Logic of programs*

General Terms

Theory, Design, Languages, Reliability

Keywords

Component-based modeling, Cyber-physical systems, Transformation, Synthesis, Fault-tolerance, Recovery, Correctness-by-construction.

[*]This is an extended version of the paper appeared in ACM International Conference on Embedded Software (EMSOFT'11). This work is partially sponsored by Canada NSERC DG 357121-2008, ORF RE03-045, ORE RE-04-036, and ISOP IS09-06-037 grants, and, by USA AFOSR FA9550-10-1-0178 and NSF CNS 0914913 grants.

1. INTRODUCTION

Development of software applications often utilizes models and abstractions to simplify the design process as well as to promote communication among individuals and teams working on a system. It is desirable if these models are built using *component-based* design so as to permit reuse and reconfiguration. In this context, design and implementation of embedded applications in a component-based fashion is no exception and, in fact, more beneficial.

An orthogonal issue in design and implementation of embedded applications is their *correctness* and *dependability*. This is because these applications are often deployed in *safety-critical* systems, operating in hostile environments where different types of faults may occur. *Fault-tolerance* is the ability of a system to continue meeting its specification (possibly degraded) even in the presence of faults. Building fault-tolerant systems is a significantly challenging task, as it is not feasible to anticipate all possible faults at design time. This challenge occurs even more often in *cyber-physical* systems, where computational components are tightly coupled with physical processes in adverse environments. Thus, it is highly desirable if designers have access to techniques that automatically *add* fault-tolerance to fault-intolerant models with respect to a newly identified set of faults. Although automated addition of fault-tolerance to *monolithic* models has extensively been studied [7–11, 19] (see Section 2 for details), we currently lack methods that add fault-tolerance to component-based models that encompass cyber-physical systems.

With this motivation, in this paper, we focus on the problem of automated synthesis of fault *recovery* mechanism for component-based models subject to cyber-physical constraints. Fault recovery ensures that a system eventually resumes its normal operation after occurrence of faults. Our contributions in this paper are as follows. We first define a generic fault model and the notion of fault recovery for component-based models. Then, we identify two sets of constraints on addition of fault recovery to cyber-physical models: (1) constraints to guarantee that adding recovery mechanism does not interfere with the normal behavior of the model in the absence of faults (i.e., conditions on preserving the correctness of the original model in the absence of faults), and (2) constraints to ensure that cyber-physical characteristics of the model are respected during addition of recovery to the original model. One example of latter constraints is that the recovery mechanism is not allowed

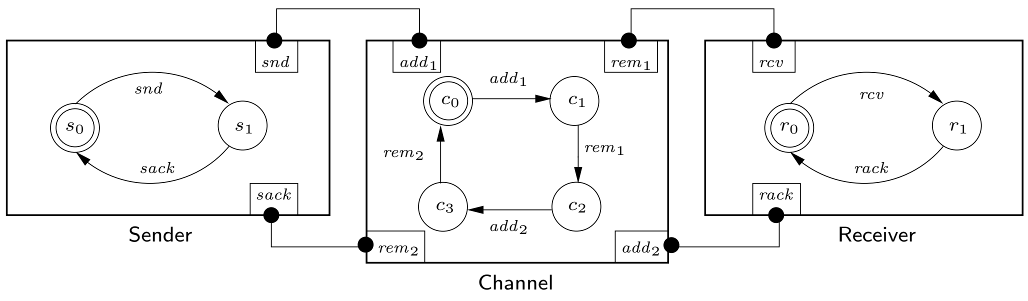

Figure 1: A communication protocol.

to alter the internal structure of physical components. In other words, recovery must be accomplished through collaboration amongst cyber components and possibly exploiting the existing structure of physical components.

We show that the corresponding synthesis problem is NP-complete in the size of the state space of the given model for adding fault recovery. Consequently, we propose a set of efficient heuristics to cope with the exponential complexity. Our heuristics preserve the normal behavior of a given model to add recovery in the absence of faults. Moreover, they preserve the structure of physical components and automatically add a recovery mechanism where only a minimal number of cyber components participate in achieving recovery when faults occur. Our heuristics are implemented using BDDs [12] and we present the results of experiments on two case studies in connection with adding fault recovery to cyber-physical systems subject to faults modelled in a component-based fashion. The experimental results validate the efficiency and effectiveness of our algorithms.

Organization. The rest of the paper is organized as follows. First, we discuss related work in Section 2. Then, in Section 3, we present the preliminary concepts. Section 4 is dedicated to our fault model and the notion of fault recovery. We discuss constraints on addition of recovery to cyber-physical component-based models in Section 5. Complexity analysis of addition of recovery is analyzed in Section 6. Then, we introduce our symbolic heuristics in Section 7. Experimental results are presented in Section 8. Finally, we make concluding remarks and discuss future work in Section 9.

2. RELATED WORK

Component-based analysis and design have been considered in numerous contexts. BIP (Behavior, Interaction, Priority) is a formal framework, where system's *behavior* is described in terms of a set of atomic components synchronized through a set of *interactions* [3, 16]. *Priorities* are used for scheduling purposes. Automated transformations for BIP have successfully been used to generate real-time [1] as well as distributed [5,6] code that is correct-by-construction. In [4], the authors address deadlock detection in BIP models, but the approach falls short on resolving deadlock states (e.g., created due to the occurrence of faults).

Automated addition of fault-tolerance to *monolithic* models is extensively studied in the literature. In [19], the authors introduce synthesis methods to add different levels of fault-tolerance to centralized and distributed models. In particular, they show that in the context of distributed models, the problem is NP-complete. The problem of adding different levels of fault-tolerance to real-time models is shown to be PSPACE-complete in [7]. Addition of multi-phase recovery, where each phase ensures satisfaction of a certain predicate during recovery, to real-time models is investigated in [9,11]. Since most related synthesis problems to add fault-tolerance to distributed and real-time models suffer from high-complexity, efficient symbolic heuristics and tools have been developed to tackle the problem [8, 10]. This line of research, however, does not deal with models expressed in a component-based fashion, or encompass cyber-physical constraints.

In [14], the authors propose a formal component model that incorporates the notion of a *safety* interface. This work is fundamentally different from our work in that we focus on recovery which implies guaranteeing liveness in the presence of faults. Lui and Joseph [21, 22, 25] introduce a uniform framework for specifying, refining, and transforming programs that provide fault-tolerance and schedulability using the Temporal Logic of Actions [20]. A survey of similar methods on monolithic systems is presented in [15]. Other approaches in component-based design of fault-tolerant systems are limited to specific architectures and platforms (e.g., [17, 18, 24]). All these approaches study analysis issues using non-automated techniques and do not target embedded and/or cyber-physical applications.

3. BACKGROUND

In this section, we review the operational semantics of our component-based framework [3, 16].

Atomic Components We define *atomic components* as transition systems with a set of ports labeling individual transitions. These ports are used for communication between different components.

Definition 1 *An* atomic component *B is a labelled transition system represented by a tuple $(Q, P, \rightarrow, q^0)$ where*

- *Q is a finite set of* states*,*
- *P is a finite set of* communication ports*,*
- *$\rightarrow \subseteq Q \times P \cup \{\tau\} \times Q$ is a finite set of* transitions *including (1)* observable *transitions labelled by ports, and* unobservable *τ transitions, and*
- *$q^0 \in Q$ is the initial state.*

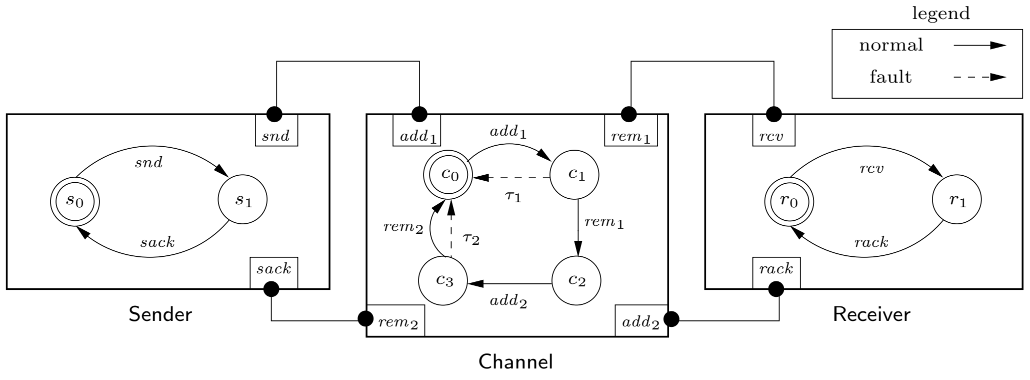

Figure 2: The communication protocol in presence of faults.

For any pair of states $q, q' \in Q$ and a port $p \in P \cup \{\tau\}$, we write $q \xrightarrow{p} q'$, iff $(q, p, q') \in \rightarrow$. When the label is irrelevant, we simply write $q \rightarrow q'$. Similarly, $q \xrightarrow{p}$ means that there exists $q' \in Q$, such that $q \xrightarrow{p} q'$. In this case, we say that p is *enabled* in state q. Figure 1 shows three atomic components Sender, Channel, and Receiver. For example, in atomic component Sender, we have $Q = \{s_0, s_1\}$, $q^0 = s_0$, $P = \{snd, sack\}$, and $\rightarrow = \{(s_0, snd, s_1), (s_1, sack, s_0)\}$.

In practice, atomic components are extended with variables. Each variable may be bound to a port and modified through interactions involving this port. We also associate a guard and an update function to each transition. A guard is a predicate on variables that must be true to allow the execution of the transition. An update function is a local computation triggered by the transition that modifies the variables. For simplicity and without loss of generality, we omit these details in this paper.

Definition 2 *A* computation *of a component $B = (Q, P, \rightarrow, q^0)$ is a finite or infinite sequence of states $q_0q_1q_2\cdots$, such that (1) $q_0 = q^0$ and (2) for all $j \geq 0$ (i) $q_j \in Q$, and (ii) $q_j \rightarrow q_{j+1}$.*

Reachable states. Let $B = (Q, P, \rightarrow, q^0)$ be a component and S be a state predicate in B; i.e., $S \subseteq Q$. We define state predicate $Reach_1(S) = S \cup \{q' \mid \exists q \in S : q \rightarrow q'\}$. That is, $Reach_1(S)$ can be computed by identifying states that are immediately forward reachable from the set of states S. Likewise, one can compute $Reach_2(S) = Reach_1(Reach_1(S))$. In a finite state model, it is straightforward to show that there exists $n \geq 1$, such that $Reach_n(S) = Reach_{n+1}(S)$. We call this the set of *reachable states* from S and denote it by $Reach(S)$. Thus, the set of reachable states of a component $B = (Q, P, \rightarrow, q^0)$ is $Reach(B) = Reach(\{q^0\})$. For example, in Figure 1, we have $Reach(\mathsf{Sender}) = \{s_0, s_1\}$.

Interaction. For a given system built from a set of m atomic components $\{B_i = (Q_i, P_i, \rightarrow_i, q_i^0)\}_{i=1}^m$, we assume that their respective sets of ports are pairwise disjoint; i.e., for any two $i \neq j$ from $\{1..m\}$, we have $P_i \cap P_j = \emptyset$. We can therefore define the set $P = \bigcup_{i=1}^m P_i$ of all ports in the system. An *interaction* is a set $a \subseteq P$ of ports. When we write $a = \{p_i\}_{i \in I}$, we suppose that for $i \in I$, $p_i \in P_i$, where $I \subseteq \{1..m\}$.

Definition 3 *A* composite component *(or simply* model*) is defined by a composition operator parametrized by a set of interactions $\gamma \subseteq 2^P$. $B = \gamma(B_1, \ldots, B_m)$, is a transition system $(Q, \gamma, \rightarrow, q^0)$, where $Q = \bigotimes_{i=1}^m Q_i$, $q^0 = (q_1^0, \ldots, q_m^0)$, and $\rightarrow$ is the least set of transitions satisfying the rule*

$$\frac{a = \{p_i\}_{i \in I} \in \gamma \qquad \forall i \in I:\ q_i \xrightarrow{p_i}_i q_i' \qquad \forall i \notin I:\ q_i = q_i'}{(q_1, \ldots, q_m) \xrightarrow{a} (q_1', \ldots, q_m')}$$

In a composite component, τ-transitions do not synchronize and execute in an interleaving fashion.

The inference rule in Definition 3 says that a composite component $B = \gamma(B_1, \ldots, B_m)$ can execute an interaction $a \in \gamma$, iff for each port $p_i \in a$, the corresponding atomic component B_i can execute a transition labelled with p_i; the states of components that do not participate in the interaction stay unchanged.

Figure 1 illustrates a composite component $CCP = \gamma(\mathsf{Sender}, \mathsf{Channel}, \mathsf{Receiver})$, where $\gamma = \{\{snd, add_1\}, \{rem_1, rcv\}, \{rack, add_2\}, \{rem_2, sack\}\}$. The behavior of the model is as follows. The component Sender sends a packet via port *snd* and receives the corresponding acknowledgement through port *sack*. Likewise, Receiver receives the sent packet through port *rcv* and sends an acknowledgement through port *rack*. By each transmission, component Channel adds an item to its single-space buffer (through ports add_1 and add_2) and by each delivery, the item is removed (via ports rem_1 and rem_2).

Similar to atomic components, one can trivially express the notions of computations and reachable states in the context of composite components as well. For example, the set of reachable states of the model in Figure 1 is the following $Reach(CCP) = \{s_0c_0r_0, s_1c_1r_0, s_1c_2r_1, s_1c_3r_0\}$.

4. FAULT MODEL AND RECOVERY

In this section, we describe our fault model and the concept of fault recovery in the context of the component-based framework described in Section 3.

4.1 Fault Model

Let $B = (Q, P, \rightarrow, q^0)$ be an atomic component. In order to specify the faulty behavior of component B, denoted B^f, we extend the component by introducing new ports P^f. A

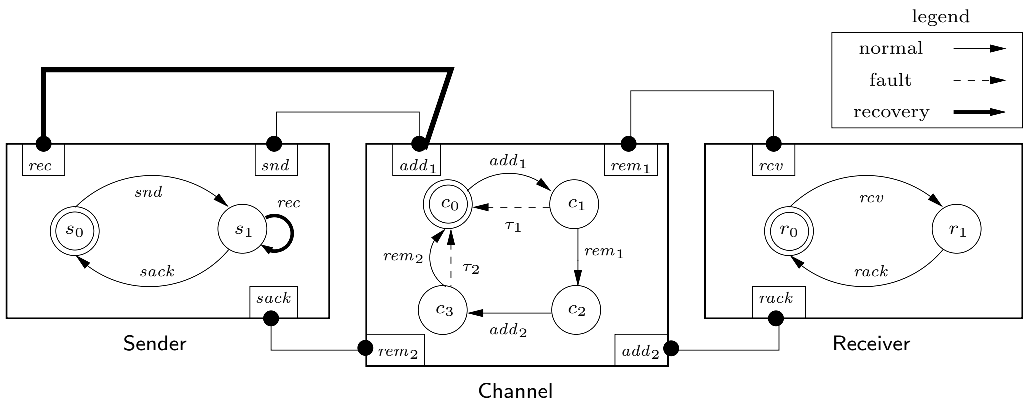

Figure 3: The communication protocol with fault recovery.

fault transition in B^f is of the form $t = (q, p, q')$, such that (1) $t \not\in \rightarrow$, (2) $q, q' \in Q$, and (3) $p \in P^f \cup \{\tau\}$. If $p \in P^f$, we say that fault transition t is *observable*. Otherwise, (i.e., $p = \tau$), we say that fault transition t is *internal*. We call transitions in $\rightarrow$ *normal* transitions. Now, let $\rightarrow^f$ be the set of fault transitions in component B^f. Thus, we obtain component $B^f = (Q, P \cup P^f, \rightarrow \cup \rightarrow^f, q^0)$ and we call it *component B in the presence of faults*.

We emphasize that such representation is possible notwithstanding the type of the faults (be they stuck-at, crash, fail-stop, timing, performance, Byzantine, message loss, etc.), the nature of the faults (be they permanent, transient, or intermittent), or the ability of the program to observe the effects of the faults (be they detectable or undetectable). In fact, representation of faults in transition systems has been explored extensively.

We also note that since our focus is on model-based synthesis of fault recovery, in our framework the set of faults needs to be provided. Having said that, one can model the effect of *unanticipated* faults by specifying the set of faults that start from any state of a component and can reach any state of the component. This is in fact the core idea in self-stabilizing systems [13], where faults can perturb the system to any arbitrary state. Modeling self-stabilizing systems and unanticipated faults in BIP have been studied in [2]. All results in this paper hold regardless of the set of faults in a model.

Likewise, we define a composite component in the presence of faults. Let $B = \gamma(B_1, \ldots, B_m)$ be a composite component and $B^f = (\gamma \cup \gamma^f)(B_1^f, \ldots, B_m^f)$ be the composite component in the presence of faults. Obviously each interaction in γ only consists of ports associated with normal transitions (called *normal interactions*). However, an interaction in γ^f (called *fault interaction*) consists of at least one port in P_j^f, where $1 \leq j \leq m$.

The concept of reachable states in a model in the presence of faults is similar to the one presented in Section 3, by considering internal, normal, and fault transitions as well as fault and normal interactions. Likewise, the notion of computations can be trivially extended by considering the union of normal and fault interactions and internal transitions.

Continuing our communication protocol example (see Figure 2), let us consider the case where the channel is lossy and faults cause loss of the sent packet (i.e., internal transition τ_1) or the acknowledgement (i.e., internal fault transition τ_2). We denote this model by $CCP^f = (\gamma \cup \gamma^f)(\mathsf{Sender}^f, \mathsf{Channel}^f, \mathsf{Receiver}^f)$. Components Sender and Receiver have no faulty behavior (i.e., $P^f = \rightarrow^f = \emptyset$). In other words, $\mathsf{Sender}^f = \mathsf{Sender}$ and $\mathsf{Receiver}^f = \mathsf{Receiver}$. On the contrary, in Channel, we have $P^f = \emptyset$, and, $\rightarrow^f = \{c_1 \longrightarrow c_0, c_3 \longrightarrow c_0\}$. Since both fault transitions are internal, we have $\gamma^f = \emptyset$.

Introducing faults to a model may result in obtaining undesirable behaviors. For instance, the occurrence of a fault transition in Channel leads CCP^f to reach the state $s_1c_0r_0$ that is not reachable in the absence of faults. This state is a *deadlock* state, as no interaction or internal transition is enabled in $s_1c_0r_0$.

Definition 4 *Let $B = (Q, \gamma, \rightarrow, q^0)$ be a model. We say that a state $q \in Q$ is a* deadlock *state, if and only if there does not exist $a \in \gamma$ such that $q \xrightarrow{a}$.*

4.2 Fault Recovery

In general, introducing faults to a model may result in two undesirable situations:

- Introducing deadlock states.
- Introducing cycles outside the set of reachable states (i.e., livelocks).

In order to tackle these pitfalls, a common practice in fault-tolerant system is to provide the system with a *recovery mechanism*. Roughly speaking, given a model B^f, fault recovery ensures that if faults cause a system to reach a state in $\neg Reach(B)$, the system is able to reach a state in $Reach(B)$ within a finite number of steps[1].

Definition 5 *Let B^f be a model in the presence of faults and $q_0q_1q_2q_3\cdots$ be a computation of B^f. We say that B^f provides* fault recovery *iff when there exists $i \geq 1$ such that $q_i \not\in Reach(B)$, then there exists $j \geq i+1$, where $q_j \in Reach(B)$.*

[1]We emphasize that in this paper, we are not concerned with *safety* issues. In other words, safety can be temporarily violated during recovery. In fault-tolerant systems, it is normally assumed that the system works correctly in the absence of faults when it reaches a normal state (in our context, a state in $Reach(B)$).

Considering our communication protocol, one way to resolve the deadlock state $s_1c_0r_0$ is to add the recovery mechanism, where a packet is re-transmitted when the packet or its acknowledgement is lost in the previous attempt. This solution results in obtaining the model $CCP' = \gamma'(\mathsf{Sender}', \mathsf{Channel}, \mathsf{Receiver})$ in Figure 3, where:

- Sender' includes an additional port *rec* and transition $s_1 \overset{rec}{\to} s_1$.
- γ' includes an additional recovery interaction $\{rec, add_1\}$.

The recovery interaction $\{rec, add_1\}$ is enabled when CCP'^f is in the state $s_1c_0r_0$. From this state, executing interaction $\{rec, add_1\}$ results in re-transmitting the last packet, which in turn leads the model to state $s_1c_1r_0$. Since this state is in $Reach(CCP')$, we are guaranteed that the model recovers and can resume its normal operation in the absence of faults.

Our goal in this paper is to investigate automated approaches to synthesize a component-based model that provides fault recovery, such as CCP' in Figure 3 from a given component-based model in the presence of faults, such as CCP^f in Figure 2. To this end, we first present a set of constraints under which we devise our synthesis decision procedure.

5. CONSTRAINTS FOR ADDING RECOVERY

As discussed in Section 4, developing a recovery mechanism for a model involves augmenting the model with computations that ensure reaching normal operation of the model when faults occur. For instance, a naive approach to achieve such recovery is to reset all components in the model (e.g., to force all components to start working from their initial states). Now, a natural question is whether such a solution is acceptable for all models in all domains.

In order to automatically synthesize a recovery mechanism for a model in the presence of faults that does not satisfy the recovery condition as stated in Definition 5, one has to first identify the constraints of such synthesis depending upon correctness criteria and application domain:

- **Correctness.** We require that addition of recovery should not interfere with the normal behavior of the model in the absence of faults; i.e., adding recovery does not result in changing the behavior of the original model in the absence of faults. This is discussed in Subsection 5.1.
- **Application domain.** The second type of the constraints deals with the case where the initial model encompasses a *cyber-physical system*. For instance, in such systems, it is not reasonable (or sometimes possible) to change the behavior of physical processes in order to add a recovery mechanism. Thus, our naive reset solution is impractical as it is often impossible to reset physical processes during system execution. These constraints are discussed in Subsection 5.2.

5.1 Non-interference Constraints

The first set of constraints ensure that adding recovery to a model in the presence of faults does not change the behavior of the model in the absence of faults. Formally, let $B = \gamma(B_1, \ldots, B_m)$ be a model and B^f be B in the presence of faults. Suppose that $B' = \gamma'(B'_1, \ldots, B'_m)$ is synthesized from B by adding some recovery mechanism; i.e., if B'^f reaches a state in $\neg Reach(B')$, then it eventually reaches a state in $Reach(B')$. Recall that γ' may include interactions that do not exist in γ. In order to guarantee that such synthesis preserves the normal behavior of the given model, we require that when B'^f recovers (i.e., it reaches a state in $Reach(B')$), it behaves the same as B in the absence of faults. More specifically, we require that the set of computations of B and B' in the absence of faults are equivalent. Furthermore, we require that all atomic components in B' behave the same as their corresponding atomic components in B. To this end, we first introduce the notion of *projection*.

Definition 6 *Let $B = (Q, P, \to, q^0)$ be a component. The* projection *of a set of transitions $T \subseteq \to$ on a state predicate $S \subseteq Q$ is the set of transitions:*

$$T \mid S = \{q \to q' \in T \mid (q \in S) \ \wedge \ (q' \in S)\};$$

i.e., the set of transitions that start in S and end in S.

We now formalize our correctness constraints as follows:

$(C1)$ For all $1 \leq i \leq m$, we have $Q_i = Q'_i$, and

$(C2)$ $\gamma' \mid Reach(B') \subseteq \gamma \mid Reach(B')$

The first constraint is concerned with atomic components in B'. In particular, constraint $C1$ requires that the state space of each atomic component is kept unchanged during synthesis. Constraint $C2$ ensures that no new interactions are added to B' in the absence of faults. This constraint along with Constraint $C1$ ensure that the set of computations of B' is equal to the set of computations of B in the absence of faults.

5.2 Constraints on Cyber-physical Interactions

In this section, we describe the constraints that have to be met during addition of recovery to component-based models of cyber-physical systems. In particular, our focus is on different types of interactions in such systems and their effect during synthesis. For simplicity, we first consider scenarios where an interaction is only between two components; interactions with three or more components can be handled in a similar fashion as discussed at the end of this Subsection.

Let $B = \gamma(B_1, \ldots, B_m)$ be a model. We partition the atomic components in B into classes B_C (cyber components) and B_P (physical components). Such partitioning is normally specified by a model designer. For instance, in our communication protocol $B_C = \{\mathsf{Sender}, \mathsf{Receiver}\}$ and $B_P = \{\mathsf{Channel}\}$. Based on this classification, we also partition interactions into four sets: cyber-to-cyber (CC), cyber-to-physical (CP), physical-to-cyber (PC), and physical-to-physical (PP). While CC and PP interactions can be trivially identified, we distinguish between CP and PC interactions based on whether the interaction is "*initiated*" by a cyber component or a physical component. We expect that the initiator of an interaction is identified by the designer, as it depends upon the semantics of the given action. For example, in our communication protocol example, it is expected that Sender initiates the message transmission and, hence, interaction $\{snd, add_1\}$ is considered to be a CP interaction. Although the issue of symmetric interactions, where the initiator cannot be determined is outside the scope of this paper, we can consider it to be a set of interactions; for example, an interaction between components A and B would be viewed as two interactions: one where A is the initiator and one where B is the initiator.

Before we describe the cyber-physical constraints, notice that among all interaction types, CC is the most simple interaction. It involves two computational components. Hence, during the synthesis process, it would be possible to revise either of the two components involved in the interaction by adding and/or removing transitions inside a component, adding and/or removing ports inside a component, or adding and/or removing interactions among components. For example, in Figure 2, if the receiver interacted with another cyber component that processed the messages, then during synthesis, it would be possible to modify the receiver component as well as the component that processed these messages. Thus, we impose no restrictions over CC interactions.

The cyber-physical constraints are as follows:

($C3$) Unlike CC interactions, CP interactions add constraints during synthesis. Consider the interaction $\{snd, add_1\}$ between Sender and Channel in Figure 2. This interaction is initiated by Sender and Channel participates in it. Since Channel is a physical process, it is not possible to modify this component during synthesis. Hence, any synthesis algorithm must keep original Channel transitions unchanged. It cannot add new transitions (e.g., from state c_1 to c_3) or remove existing transitions. Likewise, the synthesis algorithm cannot add or remove existing ports. However, it may be possible to utilize existing ports (e.g., add_1 port in Channel) to add new interactions (e.g., recovery interaction $\{rec, add_1\}$ in Figure 3 for re-transmission by Sender).

($C4$) In PC interactions, we have an issue similar to the CP interactions; i.e., the physical component cannot be modified to add/remove transitions and/or ports. Additionally, PC interactions also prevent removal of certain interactions. To illustrate this, consider the interaction $\{rem_1, rcv\}$ in Figure 2. It is expected that when Channel delivers the message, the receiver is obligated to accept it by performing its own transition $r_0 \xrightarrow{rcv} r_1$. Thus, in addition to the constraints imposed by physical components, a PC interaction requires one to deal with *forced interactions*, where an action in one component must be associated with an action in another component. In other words, in the synthesized model, we cannot have transitions where the physical component executes its transitions but the corresponding cyber component does not execute its transitions.

($C5$) The effect of PP interactions can be understood by the constraints in synthesizing physical components. Specifically, it is not possible to add/remove any such interactions and/or ports. Moreover, it is not permissible to use existing ports to create new interactions. (Note that this was possible in the CP interactions.)

Finally, for interactions that involve three or more components, these restrictions can be extended in a similar manner by first identifying the *initiator* component and then extending above restrictions accordingly. For example, a CP interaction (initiated by a cyber component with two physical components), it would not be possible to modify the physical components. However, the ports inside the physical components can be used to create new interactions with the cyber component.

6. THE SYNTHESIS PROBLEM AND ITS COMPLEXITY

In this section, we focus on complexity analysis of the problem of adding fault recovery to component-based models according to Constraints $C1 \cdots C5$ identified in Section 5. We consider an additional constraint that identifies an efficiency requirement. Specifically, we can expect that the components that interact in the original model can continue to interact efficiently in the synthesized model. However, new interactions among components that do not interact in the original model may be inefficient:

($C6$) Let $B = \gamma(B_1, \dots, B_n)$ be a model and $B' = \gamma'(B'_1, \dots, B'_n)$ be a synthesized model by adding fault recovery to B. We require that if there exists an interaction $a \in \gamma' \backslash \gamma$ that involves components $\{B_i\}_{i \in I}$, where $I \subseteq \{1..n\}$, then there must exist an interaction $a' \in \gamma$, such that a' also involves components $\{B_i\}_{i \in I}$. In other words, a new recovery interaction can only involve components that interact in the original model.

Instance. A model $B = \gamma(B_1, \dots, B_n)$ where $B^f = (\gamma \cup \gamma^f)(B_1^f, \dots, B_n^f)$ is B in the presence of faults.

Component-based CPS synthesis decision problem (CBCPS). Does there exist a model $B' = \gamma'(B'_1, \dots, B'_n)$, such that $B'^f = (\gamma' \cup \gamma'^f)(B_1'^f, \dots, B_n'^f)$ provides fault recovery and meets Constraints $C1 \cdots C6$?

Theorem 1 *CBCPS is NP-complete.*

PROOF. Given a certificate to the above decision problem, it is straightforward to verify whether the certificate solves the problem in polynomial time. Thus, the CBCPS belongs to the class NP.

We now show that the problem is NP-hard. To this end, we reduce the problem of adding *masking* fault-tolerance to *distributed* programs (denoted MFTDP) [19] to our decision problem. The MFTDP problem is as follows. Let $V = \{v_0, v_1, \dots, v_n\}$ be a finite set of variables with finite domains $D_{v_0}, D_{v_1}, \dots, D_{v_n}$, respectively. A state is determined by mapping each variable v in V to a value in D_v. The set of all possible states obtained by variables in V is called the state space. A transition is a pair of states of the form (s_0, s_1) in the state space. A process π is defined by a set of guarded commands of the form:

$$l :: g \quad \longrightarrow \quad st;$$

where l is a label, guard g is a Boolean expression (i.e., a predicate) defined over variables in V and st is a statement that describes how the process's state is updated. Without loss of generality, we assume that guards only involve

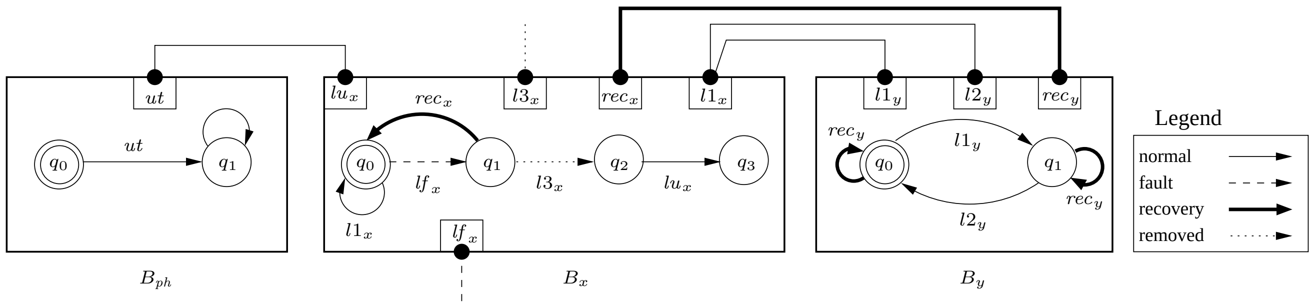

Figure 4: NP-hardness mapping.

a conjunctions of equalities (a guard with other arithmetic and Boolean operators can be trivially transformed to a set of guarded commands with only conjunctions of equalities). Thus, an action $g \longrightarrow st$ denotes the transition predicate $\{(s_0, s_1) \mid s_0 \Rightarrow g$ and s_1 is obtained by changing s_0 as prescribed by $st\}$. Consequently, a process π can be trivially transformed into a transition system. Moreover, a process π is constrained by a set of variables $\pi_r \subseteq V$ that it is allowed to read and a set $\pi_w \subseteq \pi_r$ that it is allowed to write. A program Π is a set of processes defined over a common set of variables.

To illustrate our NP-hardness proof, we utilize the following program. Let $V = \{x, y\}$, where $D_x = \{0, 1, 2, 3\}$ and $D_y = \{0, 1\}$. Process π is defined by:

$$\begin{array}{lll} l1 :: (x = 0) \wedge (y = 0) & \longrightarrow & y := 1; \\ l2 :: (x = 0) \wedge (y = 1) & \longrightarrow & y := 0; \end{array}$$

where $\pi_r = \{x, y\}$ and $\pi_w = \{y\}$. Process π' is defined by:

$$l3 :: (x = 1) \longrightarrow x := 2;$$

where $\pi'_r = \pi_w = \{x\}$.

Fault actions are also specified by a set of guarded commands. In our example, the following guarded command represents a fault action:

$$lf :: (x = 0) \longrightarrow x := 1;$$

Moreover, there exists a set of guarded commands that encode unsafe transitions that should not be executed during the execution of the program. We assume that such transitions are not reachable in the absence of faults. In our example, let the following guarded command encode an unsafe transition:

$$lu :: (x = 2) \longrightarrow x := 3;$$

The MFTDP decision problem is as follows. Let $\Pi = \{\pi_1, \dots, \pi_k\}$ (i.e., a set of processes with read/write restrictions) be a program, f be a set of faults, and ut be a set of unsafe transitions. Decide whether there exists a program Π' such that (1) in the absence of faults, the set of computations of Π' is a subset of the set of computations of Π, (2) Π' respects read/write restrictions of Π, and (3) in the presence of faults, (i) Π' never executes a transition in ut, and (ii) if Π' reaches a state in $\neg Reach(\Pi')$, then it eventually reaches a state in $Reach(\Pi')$. This problem is known to be NP-complete in the size of the state space of Π [19].

Mapping. We now map an arbitrary instance of MFTDP to an instance of CBCPS as follows. Let V be a set of variables and Π be a program defined over V in MFTDP[2]:

- (*Cyber Components*) We map each variable v in V to an atomic component $B_v = (Q_v, P_v, \rightarrow_v, q_v^0)$ in our instance of CBCPS. For instance, in Figure 4, components B_x and B_y correspond to variables x and y respectively in the above guarded commands example. For each element d in D_v (i.e., the domain of variable v), we include a state q_d in the set Q_v. For instance, $Q_x = \{q_0, q_1, q_2, q_3\}$, as $D_x = \{0, 1, 2, 3\}$. For each transition (s_0, s_1) encoded in guarded commands of Π that changes the value of variable v from d_0 to d_1, we include a transition $q_{d_0} \overset{l_v}{\rightarrow}_v q_{d_1}$, where l is the label of the guarded command in Π that encodes transition (s_0, s_1). We also include port l_v in P_v[3]. For instance, in our example (see Figure 4), guarded commands $l1$ and $l2$ are mapped to transitions $q_0 \overset{l1_y}{\rightarrow} q_1$ and $q_1 \overset{l2_y}{\rightarrow} q_0$ in component B_y, as they change the value of y. Likewise, guarded command $l3$ results in transition $q_1 \overset{l3_x}{\rightarrow} q_2$ in component B_x. We assume that all components B_v, where $v \in V$, are cyber components.
- (*Physical Component*) We also include a special physical component $B_{ph} = (Q_{ph}, P_{ph}, \rightarrow_{ph}, q_{ph}^0)$, where $Q_{ph} = \{q_0, q_1\}$, $P_{ph} = \{ut\}$, $\rightarrow_{ph} = \{q_0 \overset{ut}{\rightarrow} q_1, q_1 \rightarrow q_1\}$, and $q_{ph}^0 = q_0$ (see Figure 4). Addition of this component is regardless of the structure of the instance of MFTDP.
- (*Interactions*) In order to compose the atomic components and build a model $B_\Pi = \gamma_\Pi(B_{v_1}, \dots, B_{v_n}, B_{ph})$, for each guarded command l in a process π, we include an interaction a_l. First, this interaction does not involve a component B_v, where $v \notin \pi_r$. The interaction involves a component B_v if v appears in the guard or statement of l. If the guard of l includes a constraint of the form $v = d$, and l does not change the value of v,

[2]We note that our mapping is polynomial-time in the size of the state space of Π. Thus, our proof shows that CBCPS is NP-complete in the size of global reachable states of a component-based model.

[3]In case a guarded command encodes multiple transitions within an atomic component, a simple naming convention must be used to distinguish different ports.

then we add a self-loop $q_d \xrightarrow{l_v} q_d$ to component B_v. We also add port l_v to component B_v. For instance, mapping constraint $(x = 0)$ in $l1$ results in adding self-loop $q_0 \xrightarrow{l1_x} q_0$ and port $l1_x$ in component B_x (see Figure 4). Now, for guarded command l, we construct interaction a_l using all ports l_v, where v is a variable that participates in guarded command l; i.e, $a_l = \bigcup_{v \in V} l_v$. Observe that such an interaction synchronizes transitions in atomic components that are encoded in the corresponding guarded command l. For example, guarded command $l1$ is mapped to interaction $\{l1_x, l1_y\}$ and guarded command $l3$ is mapped to singleton interaction $\{l3_x\}$.

- (*Fault and unsafe actions*) Fault actions are mapped to transitions and interactions in a similar manner. For unsafe actions, we add corresponding transitions and interactions as well. However, we include port ut of physical component B_{ph} in all unsafe interactions as well. For example, unsafe action lu is mapped to interaction $\{ut, lu_x\}$.

Reduction. We now show that the instance of MFTDP has a solution if and only if the answer to the corresponding instance of CBCPS is affirmative:

- ($\Rightarrow$) Let the answer to MFTDP be a program Π'. We construct model $B_{\Pi'}$ from program Π' in the same way that we mapped an arbitrary instance of MFTDP to an instance of CBCPS. We now show that this model is *sound*; i.e., $B_{\Pi'}$ and B_Π satisfy the constraints identified in Section 5 and $C6$:
  - (*Constraints* $C1$ *and* $C2$) Since the set of computations of Π' is a subset of the set of computations of Π, we are guaranteed that the set of states of each atomic component in $B_{\Pi'}$ is equal to the set of state of the same component in B_Π. Also, $B_{\Pi'}$ and B_Π must have the same set of atomic components, as MFTDP cannot employ new variables to obtain Π' from Π. Moreover, the set of transitions of Π' that start and end in $\mathit{Reach}(\Pi')$ has to be a subset of the set of transitions of Π that start and end in $\mathit{Reach}(\Pi')$. Thus, in the absence of faults, we have $\gamma' \mid \mathit{Reach}(B') \subseteq \gamma \mid \mathit{Reach}(B')$, where γ and γ' are the sets of interactions of B_Π and $B_{\Pi'}$. For example, in Figure 4, the behaviors of $B_{\Pi'}$ and B_Π are identical in the absence of faults.
  - (*Fault recovery*) In the presence of faults, if Π' reaches a state in $\neg \mathit{Reach}(\Pi')$, then it eventually reaches a state in $\mathit{Reach}(\Pi')$. Thus, by adding masking fault-tolerance to Π, Π' is augmented with a set of transitions that guarantee reaching $\mathit{Reach}(\Pi')$ from a state in $\neg \mathit{Reach}(\Pi')$. These transitions in Π' form a set of transitions and interactions in $B_{\Pi'}$. By construction, these interactions guarantee that if $B_{\Pi'}$ reaches a state in $\neg \mathit{Reach}(B_{\Pi'})$ in the presence of faults, then it reaches a state in $\mathit{Reach}(B_{\Pi'})$, as by abstracting component B_{ph}, one can obtain an one-to-one correspondence between state space of Π' and global states of $B_{\Pi'}$. For example, in Figure 4, interaction $\{rec_x, rec_y\}$ is a recovery interaction.
  - (*Constraints* $C3 \ldots C5$) Since Π' never executes an unsafe transition even in the presence of faults, it implies that interactions that involve component B_{ph} never get enabled. Thus, there is no need to modify component B_{ph}, respecting constraints $C3 \ldots C5$. For example, Π' cannot include transition $((x = 1), (x = 2))$. Thus, in Figure 4, transition $q_1 \xrightarrow{l3_x} q_2$ and, hence, interaction $\{l3_x\}$ are not included in component B_x when we construct $B_{\Pi'}$ from Π', disabling interaction $\{ut, lu_x\}$ in all computations of $B_{\Pi'}$.
  - (*Constraint* $C6$) Since Π' respects read/write restrictions of Π, constructing $B_{\Pi'}$ does not include an interaction whose set of components does not match another interaction in B_Π. For example, in Figure 4, recovery interaction $\{rec_x, rec_y\}$ is added between component B_x and B_y that already interact in the original model.
- ($\Leftarrow$) Let B' be an answer to CBCPS (i.e., obtained from B_Π, respecting Constraints $C1 \cdots C6$ and fault recovery). One can construct Π' by simply computing the transition system of B'. Observe that Π' never executes a transition in ut. Otherwise, there exists a computation of B' that executes an interaction that involves component B_{ph}. Such an interaction cannot occur, because it leads the model to a deadlock situation. Also, it is not possible to remove these interactions, as B' violates Constraint $C3$ or $C4$. Moreover, Π' does not violate read/write restrictions of Π. Otherwise, B' would violate Constraint $C6$. Furthermore, it is straightforward to see that computations of Π' is a subset of computations of Π in the absence of faults and Π' augments a fault recovery mechanism (proof is similar to the first two points of the forward direction). $\Box$

7. HEURISTICS

During the addition of fault-tolerance to component-based models, we need to add recovery transitions and/or interactions to ensure that the model recovers to states that are reachable from initial states in the absence of faults. As shown in Section 6, the problem of synthesizing fault-tolerant CPS models is NP-complete when we want to preserve the efficiency of the original model. Hence, we need to identify heuristics that permit efficient implementation while ensuring that we can find the desired fault-tolerant component-based model in many examples. Moreover, in cases where preserving efficiency is not possible, we would like to minimize the number of interactions that do not satisfy constraint $C6$. Based on this discussion, we present five heuristics, next. Of these, the first two preserve $C6$. The remaining three provide a tradeoff between the feasibility to synthesize the fault-tolerant model and its efficiency.

Before we describe our heuristics, we observe the *execution cost* of any added interaction. An interaction among several components suffers from a high execution cost since it requires synchronization among several components. Also, if an interaction is added among components that were not interacting before, then the resulting execution cost may be high. Based on these observations, we identify our heuristics, next.

Heuristic 1: ***Original Interactions.*** If two components interact in the original (input) model then we can hypothesize that it is possible to implement the interaction among those two components efficiently. For this reason, we first identify any recovery interactions that can be added by only focusing on components that interacted in the original model. As an illustration, in the example in Figure 1, we can add an interaction between the pair (Sender, Channel) and (Channel, Receiver). Of course, this might include additional recovery transitions added inside one component. Also, it may include new interactions that utilize existing transitions and/or the newly added transitions. However, the choice of new transitions and/or interactions is limited by the constraints of the cyber-physical system. For example, in adding a new recovery interaction for Sender and Channel component, we cannot add new transitions or ports to Channel. However, existing ports could be composed with new ports introduced in Sender.

A limitation of the first heuristic is that there are scenarios where the original model involves interaction between several components although the synthesized model needs to include interactions that only include a subset of those components. As an illustration, consider the extension of the example in Figure 1 where there are two channels and two (corresponding) receivers. In this case, the original model will contain a (broadcast) interaction where the sender and both channels participate. However, the synthesized model may also contain additional (unicast) interactions where sender and only one channel participate. Based on this observation, we introduce the following heuristic.

Heuristic 2: ***Limited Original Interactions.*** If the original model contains an interaction that includes components B_1, $B_2, \ldots, B_n$ then during recovery, we consider interactions consisting of a subset of components in $\{B_1$, B_2, ..., $B_n\}$. Although this may result in a worst case situation where we need to consider an exponential number of possibilities, we note that this is exponential in the number of components (and not in state space). Furthermore, this could be optimized further by only considering interactions where we only consider a small subset of components (e.g., with two or three components) and their complements (i.e., all components except the chosen components).

The above heuristics only permit interactions among components that were interacting in the original model. In some scenarios, it may be necessary to add further interactions among components that did not interact in the original model. For this reason, we introduce the following heuristic:

Heuristic 3: ***Transitive Interaction.*** If no recovery could be added by applying heuristics 1 and 2, we consider new interactions that are obtained by transitive closure of interactions in the original model. Specifically, if original model contains an interaction between components B_1 and B_2 and another interaction between components B_2 and B_3, then we consider possible recovery interactions among components B_1, B_2, and B_3. As an illustration, in the example in Figure 1, we can consider an interaction between the Sender, Channel, and Receiver.

Heuristic 4: ***Limited Transitive Interaction.*** For the case where transitive interaction also fails to add the required recovery, we consider limited transitive interactions.

Number of channels	Reachable States	Time for Addition of Fault Recovery (ms)
5	10^3	48
10	10^6	58
50	$10^{3}0$	149
100	$10^{6}0$	489
200	10^{120}	2278
500	10^{300}	11935
1000	10^{600}	58763
2000	10^{1200}	256166

Table 1: Experimental results for automated addition of fault recovery to broadcast channel.

For instance, in the example considered in the previous paragraph, this would result in consideration of an interaction between components B_1 and B_3.

Finally, for the case where all these heuristics fail, we consider new interactions that are developed from scratch. Specifically, in this case, the new recovery interactions may not be related to the interactions in the original model.

Heuristic 5: ***Hail Mary.*** Since our goal is to identify interactions with small number of components, we first consider interactions among any set of two components, then any set of three components and so on.

8. EXPERIMENTAL RESULTS

In this section, we discuss the experimental results of applying the heuristics from Section 7 in the context of the example in Figure 1 as well as another example. All experiments in this section are run on a PC with AMD Athlon II X4 2.8 MHz processor with 6GB RAM. All heuristics are implemented using the Glu/CUDD package [23].

8.1 Broadcast Channel

Based on the heuristics in Section 7, we first consider possible new interactions that can be added to the model in Figure 1. For the case where there is only one channel and one receiver, according to heuristic 1, we consider possible interactions between (Sender, Channel) and (Channel, Receiver). While we consider all possible interactions that can be added, we ensure that no new transitions can be added to Channel component and no existing transitions from Channel component are removed. Based on these restrictions, we add the interaction that consists of transition (s_1, s_1) in Sender and the transition (c_0, c_1) in the Channel. Applying this heuristic results in obtaining the model in Figure 3.

For the case where there are multiple channels, heuristic 1 adds the interaction to deal with the case where all receivers lose the message. Subsequently, heuristic 2 deals with the case where a subset of receivers lose the message. First, heuristic 2 considers interactions between a subset of two components. Thus, it considers an interaction between (1) a pair of channels and (2) between the sender and one channel. Of these, due to the restriction imposed on the channel component, we cannot add any interactions between a pair of channels. And, it adds an interaction between the sender and one channel. Since added interactions suffice to provide recovery for this example, an exponential blowup in the number of possible component combinations is avoided.

The results for the time required for synthesis for a differ-

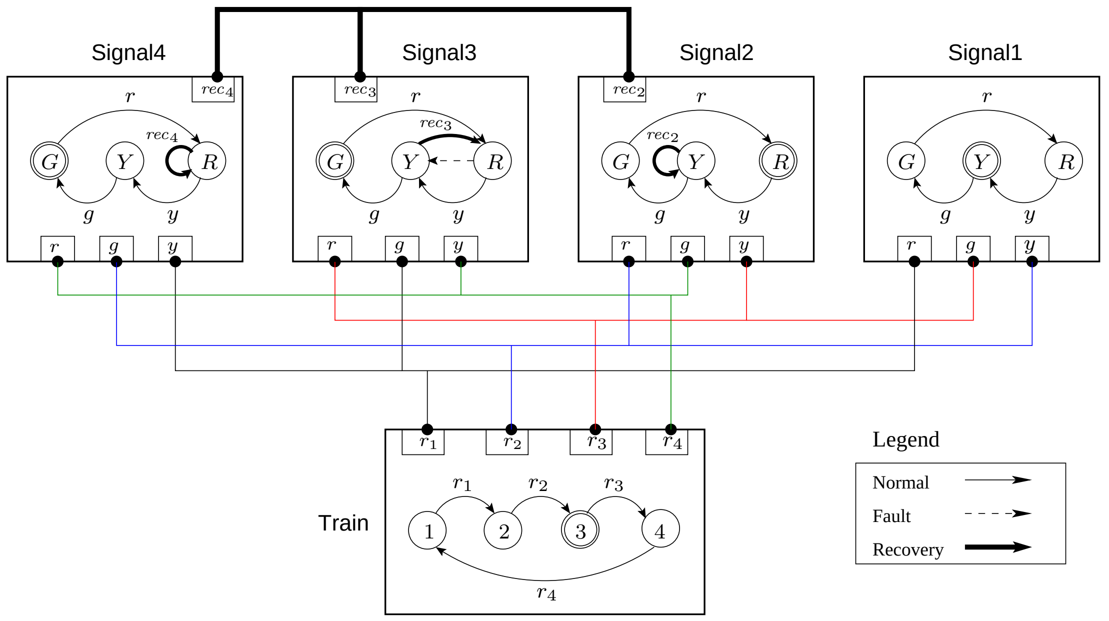

Figure 5: A train signal controller.

ent number of receivers is as shown in Table 1. Furthermore, based on the discussion above, the time required for obtaining the fault-tolerant program is at most quadratic in the number of components. Furthermore, the time needed for addition of fault recovery is small. For example, even with 1000 receivers, the time for addition of fault recovery is less than one minute.

8.2 Train Signals Controller

We also applied our heuristics to a train signal controller example. In this example (see Figure 5), a train (physical component Train) travels on a circular railway track controlled by a sequence of signals (cyber components Signal1 $\cdots$ Signal4). The train can pass a signal only if it is green. Each signal operates as follows. When Train passes the signal (say Signal3), it changes phase from green to red. This action is synchronized with the two preceding signals (i.e., Signal2 and Signal1), so that the previous signal turns yellow (i.e., Signal2) and the signal before that (i.e., Signal1) turns green. Moreover, this action is synchronized with the location of Train as well (i.e., moving from location 3 to 4). Thus, by starting from initial state $YRGG3$, where the first 4 letters denote the state of the signals and the last number represents the state of Train, the composite component exhibits a computation of the following form:

$$YRGG3, GYRG4, GGYR1, RGGY2, \cdots$$

Faults can arbitrarily cause change of phase from red to yellow in each component. We show this in Figure 5 in component Signal3 only for simplicity. The occurrence of such a fault causes the model to reach the state $GYYG4$ which is a deadlock state. Applying the heuristics introduced in Section 7 adds the recovery interaction $\{rec_4, rec_3, rec_2\}$ and the corresponding transitions in component Signal4, Signal3, and Signal2 as shown in Figure 5.

The time for synthesis for the train example is as shown in Table 2. As we can see, when the number of trains is

	2 trains	3 trains	4 trains	5 trains
5 signals	50ms			
8 signals	52ms	53ms		
10 signals	107ms	65ms	62ms	
12 signals	119ms	582ms	54ms	
15 signals	765ms	11649ms	105ms	138ms

Table 2: Experimental results for automated addition of fault recovery to train signals controller.

increased initially, the reachable state space increases. This causes an increase in the time for addition of fault recovery. However, after a certain threshold on the number of trains, the level of concurrency decreases. In other words, only a small subset of trains can move at a given instance. Hence, after this threshold, the time needed for addition of fault recovery decreases.

9. CONCLUSION

In this paper, we focused on the problem of automated addition of fault recovery to component-based models that encompass cyber-physical systems. We used the BIP framework [3, 16] to specify component-based models and proposed a set of constraints to capture cyber-physical features. We showed that automated addition of fault recovery to cyber-physical component-based models augmented with faulty behavior is NP-complete. Consequently, we introduced a set of heuristics to cope with exponential complexity. Our method is fully implemented using BDDs [12] for which we presented very encouraging experimental results. Although our focus is on BIP, all results in this paper can be applied to any model specified in terms of a set of components synchronized by broadcast and rendezvous interactions.

There are still several open complexity questions. For example, the complexity of the recovery addition problem where non-interacting components can participate in recovery (i.e., eliminating Constraint $C6$) is unknown. Another open problem is the complexity of the synthesis problem where recovery interactions must involve a minimum number of components. We are currently working on the same problem in the context of timed component-based cyber-physical models. Another interesting research direction is to exploit invariant generation techniques to generate recovery mechanisms for component-based models.

10. REFERENCES

[1] T. Abdellatif, J. Combaz, and J. Sifakis. Model-based implementation of real-time applications. In *ACM International Conference on Embedded Software (EMSOFT)*, pages 229–238, 2010.

[2] A. Basu, B. Bonakdarpour, M. Bozga, and J. Sifakis. Systematic correct construction of self-stabilizing systems: A case study. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 4–18, 2010.

[3] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in BIP. In *Software Engineering and Formal Methods (SEFM)*, pages 3–12, 2006.

[4] S. Bensalem, M. Bozga, J. Sifakis, and T.-H. Nguyen. Compositional verification for component-based systems and application. In *Automated Technology for Verification and Analysis (ATVA)*, pages 64–79, 2008.

[5] B. Bonakdarpour, M. Bozga, M. Jaber, J. Quilbeuf, and J. Sifakis. Automated conflict-free distributed implementation of component-based models. In *IEEE Symposium on Industrial Embedded Systems (SIES)*, pages 108 – 117, 2010.

[6] B. Bonakdarpour, M. Bozga, M. Jaber, J. Quilbeuf, and J. Sifakis. From high-level component-based models to distributed implementations. In *ACM International Conference on Embedded Software (EMSOFT)*, pages 209–218, 2010.

[7] B. Bonakdarpour and S. S. Kulkarni. Incremental synthesis of fault-tolerant real-time programs. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, LNCS 4280, pages 122–136, 2006.

[8] B. Bonakdarpour and S. S. Kulkarni. Exploiting symbolic techniques in automated synthesis of distributed programs with large state space. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 3–10, 2007.

[9] B. Bonakdarpour and S. S. Kulkarni. Masking faults while providing bounded-time phased recovery. In *International Symposium on Formal Methods (FM)*, pages 374–389, 2008.

[10] B. Bonakdarpour and S. S. Kulkarni. SYCRAFT: A tool for synthesizing fault-tolerant distributed programs. In *Concurrency Theory (CONCUR)*, pages 167–171, 2008.

[11] B. Bonakdarpour and S. S. Kulkarni. On the complexity of relaxed and graceful bounded-time 2-phase recovery. In *International Symposium on Formal Methods (FM)*, pages 660–675, 2009.

[12] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[13] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[14] J. Elmqvist, S. Nadjm-tehrani, and M. Minea. Safety interfaces for component-based systems. In *Computer Safety, Reliability, and Security (SAFECOMP)*, pages 246–260, 2005.

[15] F. C. Gärtner. Transformational approaches to the specification and verification of fault-tolerant systems: Formal background and classification. *Journal of Universal Computer Science*, 5(10):668–692, 1999.

[16] G. Gössler and J. Sifakis. Composition for component-based modeling. *Science of Computer Programming*, 55(1-3):161–183, 2005.

[17] P. A. C. Guerra, C. M. F. Rubira, and R. Lemos. An idealized fault-tolerant architectural component, 2002.

[18] B. Hamid, A. Radermacher, A. Lanusse, C. Jouvray, S. Gérard, and F. Terrier. Designing fault-tolerant component based applications with a model driven approach. In *Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS)*, pages 9–20, 2008.

[19] S. S. Kulkarni and A. Arora. Automating the addition of fault-tolerance. In *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, pages 82–93, 2000.

[20] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16:872–923, May 1994.

[21] Z. Liu and M. Joseph. Transformation of programs for fault-tolerance. *Formal Aspects of Computing*, 4(5):442–469, 1992.

[22] Zhiming Liu and Mathai Joseph. Specification and verification of fault-tolerance, timing, and scheduling. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 21(1):46–89, 1999.

[23] F. Somenzi. CUDD: Colorado University Decision Diagram Package. `http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html`.

[24] J. Xu, B. Randell, A. B. Romanovsky, C. M. F. Rubira, R. J. Stroud, and Z. Wu. Fault tolerance in concurrent object-oriented software through coordinated error recovery. In *Symosium on Fault-Tolerant Computing (FTCS)*, pages 499–508, 1995.

[25] Liu Z. and Joseph M. Specification and verification of recovery in asynchronous communicating systems. In *Formal techniques in real-time and fault-tolerant systems (FTRTFT)*, pages 137–163, 1993.