

# Supervisory Control for Real-Time Systems Based on Conflict-Tolerant Controllers

Deepak D'Souza, Madhu Gopinathan, S. Ramesh and Prahladaradan Sampath

## Abstract

*This paper addresses the problem of detecting and resolving conflicts due to timing constraints imposed by features in real-time and hybrid systems. We consider systems composed of a base system with multiple **features** or **controllers**, each of which independently advise the system on how to react to input events so as to conform to their individual specifications. We propose a methodology for developing such systems in a modular manner based on the notion of **conflict-tolerant** features that are designed to continue offering advice even when their advice has been overridden in the past. We give a simple priority-based scheme for composing such features. This guarantees the **maximal** use of each feature. We provide a formal framework for specifying such features, and a compositional technique for verifying systems developed in this framework.*

## I. INTRODUCTION

We consider systems comprising a base system along with multiple *features* where each feature advises the base system on how to conform to the feature specification. Such systems are commonly encountered in various domains from telecom to dynamical control systems. One of the problems faced in integrating various features in such systems is that the system may reach a point of “conflict” between two (or more) features, where the features do not agree on a common action for the base system to perform. This situation is an instance of what is referred to in the literature as the *feature interaction* problem [9], [7], [6].

Consider a development model where individual features are specified by original equipment manufacturers

(OEMs) and implemented by third party vendors. The OEM must *verify* that feature implementations conform to their specifications. In addition, the OEM must *integrate* various features into a final product. Detecting and resolving conflicts between features during feature integration poses a significant challenge. Conflict between two features can, of course, be resolved by respecifying or redesigning one of the features. However this is often not possible in practice and there is no guarantee that the redesigned feature does not conflict with some other feature. Moreover, redesign of a feature for handling specific conflicts reduces the scope for reusing the feature in multiple contexts.

An alternative to redesign is to *suspend* the feature with lower priority so that the base system can continue with the advice of the higher priority feature. However the issue now is how and when to *resume* the suspended feature so as to maximize its use.

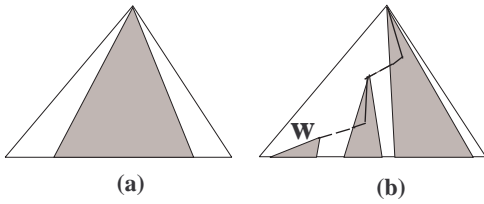
**Our Approach.** In this paper we propose a formal framework for developing feature based systems in a way that overcomes some of the problems outlined above. We work in the setting of real-time features so as to model more closely the time-dependent features that arise in the automotive domain. The framework is based on the novel notion of *conflict-tolerance*, which requires features to be *resilient* or *tolerant* with regard to violations of their advice. Thus, unlike the classical notion of a feature, a conflict-tolerant feature can observe that its advice has been over-ridden, takes into account the over-riding event, and proceeds to offer advice for *subsequent* behaviour of the system.

The starting point of this framework is the notion of a conflict-tolerant *specification* of a feature. A classical safety specification can be viewed as a prefix-closed language of finite words containing all the system behaviours which are considered *safe*. This can be pictured as a safety *cone* in the tree representing all possible behaviours, as shown in Fig. 1 (a). A conflict-tolerant specification on the other hand can be viewed as an *advice function* that specifies for *each* behaviour  $w$  of the base system, a safety

Deepak D'Souza and Madhu Gopinathan are with the Indian Institute of Science, Bangalore, India. {deepakd, gmadhu}@csa.iisc.ernet.in

S. Ramesh and Prahladaradan Sampath are with GM India Science Lab, Bangalore, India. {ramesh.s, p.sampath}@gm.com

cone comprising all future behaviours that are considered safe, *after* the system has exhibited behaviour  $w$  (Fig. 1 (b)).

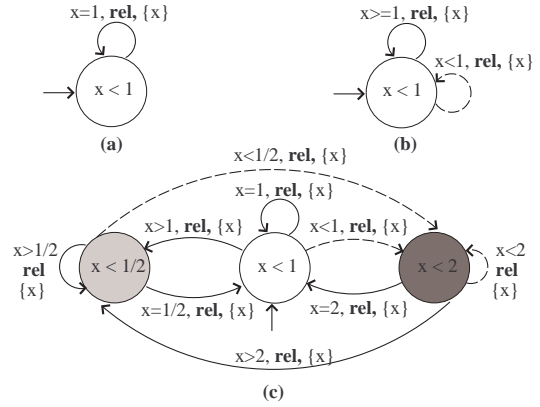


**Fig. 1. The conflict-tolerant specification on the right advises on how to extend  $w$  even though its advice has been overridden (dashed line) in the past when generating  $w$ .**

To illustrate how a conflict-tolerant specification can capture a specifier’s intent more richly than a classical specification, consider a feature that is required to release (denoted by the event “**rel**”) a single unit of lubrication every 1 second. A classical specification for this feature may be given by the timed transition system shown in Fig. 2(a). The systems we show use the timed automata model [1]. Thus they have continuous variables which grow with slope 1 (hence called “clocks”) and which can be reset to zero (the label “ $\{x\}$ ” indicates that clock  $x$  is reset along the transition). The state invariant “ $x < 1$ ” is to be interpreted as a “time-can-progress” condition: thus as long as the value of the clock  $x$  is less than 1, the specification recommends letting time elapse. However when  $x = 1$ , time elapse is no more recommended and instead the action **rel** is recommended “urgently”. Fig. 2(b) and (c) show annotated timed transition systems denoting conflict-tolerant specifications that induce the same classical specification shown in (a). These transition systems differ from the classical one in two ways. The dashed transitions are to be read as “not-advised”, and they enable the specification to keep track of events that occur in violation of its advice at a given state. Secondly, in a state the time-can-progress condition can be violated to let time elapse against the advice of the specification. In this case time elapses but control remains in the same state. Thus specification (b), advises **rel** urgently at all times after 1 second from the previous release, until its advice is taken. When  $x < 1$ , its advice is only to let time elapse; If the event **rel** is performed against its advice, it uses the dashed transition to keep track of this and resets its clock  $x$ . If its advice is not followed when  $x = 1$ , it continues to advise that **rel** be done urgently, i.e. time elapse is no more recommended

The specification (c) keeps track of whether the previous release was “on time” or not and changes its advice

accordingly. Thus it advises **rel** 0.5 seconds after the last release if the previous release was “late” (lightly shaded state) and advises **rel** 2 seconds after the last release if it was “early” (darkly shaded state).



**Fig. 2. A classical spec (a), and conflict-tolerant specs (b) and (c) that induce the same classical spec (a).**

A conflict-tolerant feature implementation can be viewed as a timed transition system with transitions annotated as *advised* and *not-advised*, similar to the conflict-tolerant specifications described above. A feature implementation is now said to satisfy a conflict-tolerant specification (with respect to a given base system), if after every possible behaviour  $w$  of the base system, the behaviours of the feature implementation are contained in the safety cone specified by the specification for  $w$ . We address the verification problem in this framework and give a decision procedure to solve this problem in the setting of Alur-Dill timed transition systems.

An important aspect of our framework is the fact that conflict-tolerant features admit a simple and effective composition scheme based on a prioritization of the features being composed. The composition scheme can also be viewed as a conflict resolution technique. The composition scheme ensures that the resulting system always satisfies the specification of the highest priority feature. Additionally, it follows the advice of all other features  $F$ , *except* at points where each action in the advice of  $F$  conflicts with the advice of a higher priority feature. It is in this sense that each feature is *maximally* utilized.

Our approach of viewing features as discrete event controllers [10] follows that of [12], [2]. In both these works, the main issue addressed is that of resuming the advice of a controller once it has been suspended due to conflict with a higher priority controller. In [2], specifications

are designed to anticipate conflict, by having two kinds of states, *in-spec* and *out-of-spec*. When a controller’s specification is violated it transitions to an out-of-spec state from where it passively observes the system behaviour, till it sees a specified event that brings it back to an in-spec state. Note that these controllers do not offer any useful advice in the out-of-spec states. In [8] a rule-based feature model and composition operators for resolving conflicts based on prioritization is presented. However, the notion of a conflict-tolerant *specification* (as against the feature implementation itself) is absent in their work.

## II. EXAMPLE OF A CONFLICT-TOLERANT CONTROLLER

We illustrate some of these ideas with an example of a simplified door motor control system. Consider an electronic control unit (ECU) that controls the motors for locking and unlocking the doors of a car. The transition system for the motor (the base system) is shown in Fig. 3. We denote the environment events in italics and system events in bold. The set  $\Sigma_e$  comprises *lock\_req*, *unlock\_req* and *crash*, while  $\Sigma_s$  is  $\{\mathbf{lock}, \mathbf{unlock}\}$ .

The environment events *lock\_req* or *unlock\_req* occur respectively, when the lock or unlock button on a remote key is pressed. The environment event *crash* occurs when a car crash is sensed, and could be followed up by an the *ok* event signaling that it is ok to resume normal operation. The system event **lock** occurs when the door is locked and the system event **unlock** occurs when the door is unlocked.

The base system is typically run with several controllers including a “vanilla” controller which locks or unlocks the doors in response to passenger requests. Consider a feature called *crash safety* [3] which requires that if the doors are locked and a crash occurs, then the doors must be unlocked within 10 seconds. Fig. 4 shows a classical safety specification for this requirement. If the doors are locked and a crash occurs, then the clock  $z$  is reset and we are in the state labeled with the invariant  $z \leq 10$ . The delay advised in this state is at most 10 seconds as specified by the time-can-progress condition, and within 10 seconds we require that an **unlock** event occurs and the spec moves to state  $C$  where it does not allow any **lock** event till an *ok* event occurs.

Fig. 5 shows a possible conflict-tolerant specification for the crash-safety spec. Unlike a classical safety specification, this specification also specifies what needs to be done if  $z > 10$  (this could be the case if a feature with priority greater than crash safety requires more delay before the doors can be unlocked). The spec specifies that the doors must be unlocked as soon as possible when  $z \geq 10$ , i.e. “time elapse” is not advised when  $z \geq 10$ , and the only

advised events are *ok* and **unlock**. Further, in the state labeled  $C$ , if a **lock** event takes place (against its advice) the spec goes to a state labeled  $\perp$  where the tcp condition is “False”, and the advice is to do an **unlock** and go back to state  $C$ , or to go back to state  $L$  if an *ok* event is sensed. In this case, the given spec itself can be used as a conflict-tolerant controller that controls the given base system in a way that satisfies the given conflict-tolerant spec.

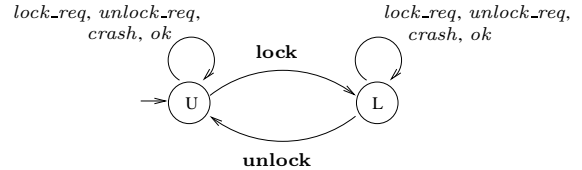


Fig. 3. Door motor base system.

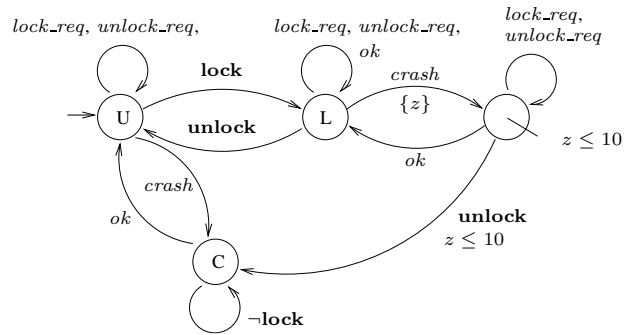


Fig. 4. Crash Safety Specification.

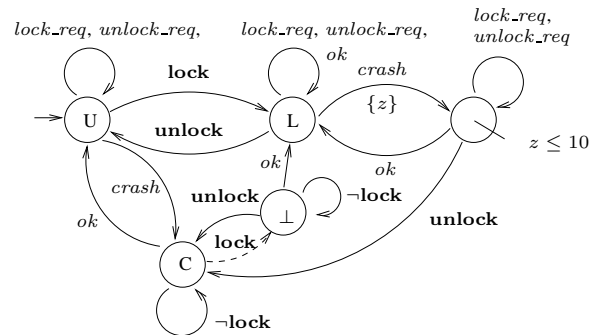


Fig. 5. Conflict-Tolerant Crash Safety Specification.

### III. VERIFICATION OF A TOLERANT CONTROLLER

In this section we address the verification problem in the conflict-tolerant framework.

Let  $\Sigma$  be an alphabet, partitioned into  $\Sigma_e$  and  $\Sigma_s$ . For a TTS  $\mathcal{T}$  over  $\Sigma$ , a state  $q \in Q$ , and an event  $a \in \Sigma$ , we denote by  $\text{cond}_{\mathcal{T}}(q, a)$ , the condition under which  $a$  is enabled in  $q$ , i.e.  $\text{cond}_{\mathcal{T}}(q, a) = \bigvee_{(q, g, a, X, q') \in \rightarrow} g$ , where  $\rightarrow$  is the transition relation of  $\mathcal{T}$ . For a conflict-tolerant TTS  $\mathcal{T}'$ ,  $\text{cond}_{\mathcal{T}'}(q, a)$  gives the condition under which  $a$  is advised from  $q$ , i.e.  $\text{cond}_{\mathcal{T}'}(q, a) = \bigvee_{(q, g, a, X, q') \in \rightarrow \setminus N} g$ .

*Theorem 1 (Verification):* Given a base system  $\mathcal{B}$  over  $\Sigma$ , a regular conflict-tolerant specification  $\mathcal{S}'$ , and a conflict-tolerant controller  $\mathcal{C}'$ , we can check whether  $\mathcal{C}'$  is a valid conflict-tolerant controller for  $\mathcal{B}$  satisfying  $\mathcal{S}'$ .

*Proof:* It is easy to see that a necessary and sufficient condition for  $\mathcal{C}'$  to be a valid controller for  $\mathcal{B}$  and satisfying  $\mathcal{S}'$ , is to check that in the synchronized product  $\mathcal{B} \parallel \mathcal{C}' \parallel \mathcal{S}'$ , there does *not* exist a configuration  $((b, p, q), v)$  which is reachable from the initial configuration  $((s_{\mathcal{B}}, s_{\mathcal{C}'}, s_{\mathcal{S}'}, \vec{0})$ , and satisfies one of the following conditions:

- 1) ( $\mathcal{C}'$  is restricting) there is an environment event  $e \in \Sigma_e$  allowed by  $\mathcal{B}$ , but not advised by  $\mathcal{C}'$ , i.e. there exists  $e \in \Sigma_e$  such that  $v \models \text{cond}_{\mathcal{B}}(b, e)$  and  $v \not\models \text{cond}_{\mathcal{C}'}(p, e)$ .
- 2) ( $\mathcal{C}'$  is blocking) there is no discrete or delay step that is enabled by  $\mathcal{B}$  and advised by  $\mathcal{C}'$ , i.e.  $v \not\models \text{tcp}(p)$  and  $v \not\models (\bigvee_{a \in \Sigma} \text{cond}_{\mathcal{B}}(b, a) \wedge \text{cond}_{\mathcal{C}'}(p, a))$ .
- 3) ( $\mathcal{C}'$  does not satisfy  $\mathcal{S}'$ ) there is a discrete or delay step that is both enabled by  $\mathcal{B}$  and advised by  $\mathcal{C}'$ , but not advised by  $\mathcal{S}'$ . That is, either for some  $a \in \Sigma$ ,  $v \models \text{cond}_{\mathcal{B}}(b, a) \wedge \text{cond}_{\mathcal{C}'}(q, a)$  but  $v \not\models \text{cond}_{\mathcal{S}'}(q, a)$ , or  $v \models \text{tcp}_{\mathcal{C}'}(p)$  and  $v \not\models \text{tcp}_{\mathcal{S}'}(q)$ .

This condition can be checked in linear time using the region automaton [1] for the synchronised product of  $\mathcal{B}$ ,  $\mathcal{C}'$  and  $\mathcal{S}'$ . ■

### IV. COMPOSITION OF CONFLICT-TOLERANT CONTROLLERS

We now give a way of composing conflict-tolerant controllers – or equivalently building a supervisory controller – based on a given prioritization of the controllers. The composition guarantees that the advice of each controller is used in the “best possible” way.

Let  $\mathcal{B} = \langle B, C_0, s_0, \rightarrow_{\mathcal{B}} \rangle$ , be a base system over  $\Sigma$ , and let  $\mathcal{C}'_1 = \langle Q_1, C_1, s_1, \rightarrow_1, N_1, \text{tcp}_1 \rangle$  and  $\mathcal{C}'_2 = \langle Q_2, C_2, s_2, \rightarrow_2, N_2, \text{tcp}_2 \rangle$  be valid conflict-tolerant controllers for  $\mathcal{B}$ . We assume that the set of clocks  $C_0, C_1$  and  $C_2$  are disjoint. Let  $P$  be a priority ordering between

$\mathcal{C}'_1$  and  $\mathcal{C}'_2$ , and say  $P$  assigns a higher priority to  $\mathcal{C}'_1$ . We denote this by  $\mathcal{C}'_1 >_P \mathcal{C}'_2$ .

For  $a \in \Sigma$ , and  $p \in Q$ ,  $q_1 \in Q_1$ , and  $q_2 \in Q_2$ , let  $g_{11}^a(p, q_1, q_2)$  stand for the disjunction of constraints of the form  $g \wedge g_1 \wedge g_2$  such that there exist transitions  $t = (p, a, g, X, p')$ ,  $t_1 = (q_1, a, g_1, X_1, q'_1)$  and  $t_2 = (q_2, a, g_2, X_2, q'_2)$  in  $\mathcal{B}$ ,  $\mathcal{C}'_1$  and  $\mathcal{C}'_2$  respectively, with  $t_1 \notin N_1$  and  $t_2 \notin N_2$ . Let  $g_{11}(p, q_1, q_2)$  stand for the constraint  $\bigvee_{a \in \Sigma} g_{11}^a(p, q_1, q_2)$ . Also, let  $\text{tcp}_{11}(p, q_1, q_2)$  denote the constraint  $\text{tcp}_1(q_1) \wedge \text{tcp}_2(q_2)$ . Then:

*Definition 1 (Prioritized Composition of Controllers):*

The  $P$ -prioritized composition of the controllers  $\mathcal{C}'_1$  and  $\mathcal{C}'_2$ , with respect to the base system  $\mathcal{B}$ , is denoted  $\parallel_{P, \mathcal{B}}(\mathcal{C}'_1, \mathcal{C}'_2)$ , and defined to be the timed transition system  $\mathcal{C} = \langle B \times Q_1 \times Q_2, C_0 \cup C_1 \cup C_2, (s_0, s_1, s_2), \Rightarrow, \text{tcp} \rangle$ , where  $\Rightarrow$  and  $\text{tcp}$  are defined as follows. Let  $a \in \Sigma$ , and  $t = (p, a, g, X, p')$ ,  $t_1 = (q_1, a, g_1, X_1, q'_1)$  and  $t_2 = (q_2, a, g_2, X_2, q'_2)$  be transitions in  $\mathcal{B}$ ,  $\mathcal{C}'_1$  and  $\mathcal{C}'_2$  respectively. Then we have transitions in  $\mathcal{C}$  according to the following rules:

- 1) If  $t_1 \notin N_1$  and  $t_2 \notin N_2$  (that is they are both advised transitions) then add the transition

$$((p, q_1, q_2), a, g \wedge g_1 \wedge g_2, X \cup X_1 \cup X_2, (p', q'_1, q'_2)).$$

- 2) If  $t_1 \notin N_1$  and  $t_2 \in N_2$  (that is  $t_1$  is advised but  $t_2$  is not) then add the transition

$$((p, q_1, q_2), a, g'', X \cup X_1 \cup X_2, (p', q'_1, q'_2))$$

$$\text{where } g'' = g \wedge g_1 \wedge g_2 \wedge \neg g_{11}(p, q_1, q_2) \wedge \neg \text{tcp}_{11}(p, q_1, q_2).$$

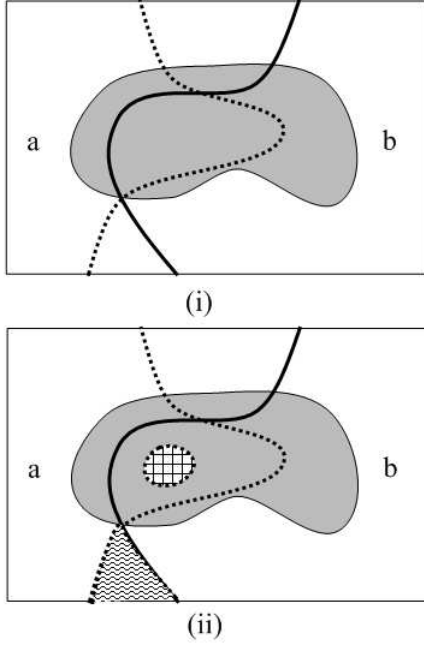
The time-can-progress condition  $\text{tcp}$  is given by  $\text{tcp}(p, q_1, q_2) = \text{tcp}_{11}(p, q_1, q_2) \vee \text{tcp}_{10}(p, q_1, q_2)$ , where  $\text{tcp}_{10}(p, q_1, q_2) = \text{tcp}_1(q_1) \wedge \neg \text{tcp}_2(q_2) \wedge \neg g_{11}(p, q_1, q_2)$ .

*Lemma 1:* The controller  $\mathcal{C} = \parallel_{P, \mathcal{B}}(\mathcal{C}'_1, \mathcal{C}'_2)$  defined above is a **valid** classical controller for  $\mathcal{B}$ . The immediate advice function  $f_{\mathcal{C}}^i$  it induces is given as follows. For each  $\tau \in L(\mathcal{B})$ :

$$f_{\mathcal{C}}^i(\tau) = \begin{cases} f_{\mathcal{B}}^i(\tau) \cap f_{\mathcal{C}'_1}^i(\tau) \cap f_{\mathcal{C}'_2}^i(\tau) & \text{if } f_{\mathcal{B}}^i(\tau) \cap f_{\mathcal{C}'_1}^i(\tau) \cap f_{\mathcal{C}'_2}^i(\tau) \neq \emptyset \\ f_{\mathcal{B}}^i(\tau) \cap f_{\mathcal{C}'_1}^i(\tau) & \text{otherwise.} \end{cases}$$

*Proof:* [Proof Sketch] Let  $\mathcal{C}'_1, \mathcal{C}'_2$  be valid conflict-tolerant controllers for a base system  $\mathcal{B}$ . Rectangle (i) in Fig. 6 shows a partitioning of the configurations of a conflict-tolerant controller  $\mathcal{C}'_1$ . In the region to the left of the dark line,  $a$  is advised. In the region to the right of the dotted line,  $b$  is advised and in the shaded region, delay  $\delta$  is advised.

Rectangle (ii) in Fig. 6 shows the configurations of  $\mathcal{C}'_1$  where there is a conflict with  $\mathcal{C}'_2$ . In the region shaded with squares,  $\mathcal{C}'_1$  advises only  $\delta$  whereas  $\mathcal{C}'_2$  does not advise  $\delta$ . Time can progress in this region as  $\mathcal{C}'_1$  advises  $\delta$  and there is no other advice in common with  $\mathcal{C}'_2$ . In the region shaded



**Fig. 6. Rectangle (i) shows a partitioning of the configurations of  $C'_1$ . Rectangle (ii) marks regions where the advice of  $C'_1$  overrides that of  $C'_2$  due to conflict.**

with a wavy pattern,  $C'_1$ 's advice is  $\{a, b\}$  but  $C'_2$  advises only  $\delta$ . By the second rule in Definition 1, the resulting advice is  $\{a, b\}$  as  $C'_1$  does not have any other advice in common with  $C'_2$ . ■

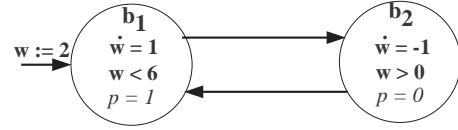
*Theorem 2:* The timed transition system  $C = \parallel_{P, \mathcal{B}}(C'_1, C'_2)$  is a valid controller for  $\mathcal{B}$ . Furthermore,  $C$  satisfies the specifications  $S'_1, S'_2$  in the following “maximal” sense. For every  $\sigma \in L(\mathcal{B} \parallel C)$ :

- 1)  $\sigma \in L_{\varepsilon}^c(S'_1)$ .
- 2) Let  $I_0, I_1, \dots, I_k$  be the sequence of adjacent intervals induced by  $f_{C'_2}$  on  $\sigma$  such that throughout the even intervals (WLOG)  $\sigma$  is according to the advice of  $C'_2$ , and throughout the odd intervals, the advice of  $C'_2$  is in conflict with that of  $C'_1$ , in that for each prefix  $\tau$  of  $\sigma$  in these intervals, for each  $c \in f_{C'_2}^i(\tau)$ ,  $c \notin f_{\mathcal{B}}^i(\tau) \cap f_{C'_1}^i(\tau)$ . □

We can generalize these results to the prioritized composition of any number of controllers.

## V. CONFLICT-TOLERANT CONTROLLERS FOR HYBRID SYSTEMS

In ongoing work, we are formulating the notion of a conflict-tolerant specification for systems modeled as



**Fig. 7. A Water Tank (plant).**

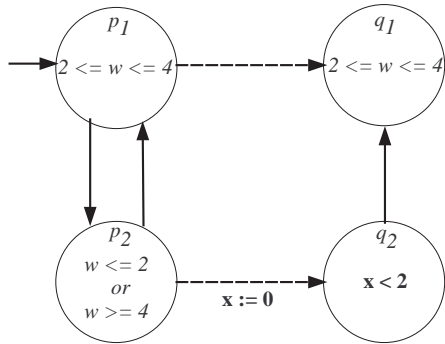
dynamical or hybrid systems. Our set up is similar to that of [11]. Here we assume the base system or plant is a modeled as a hybrid automaton over continuous variables  $V$  which is partitioned into system variables  $X$  and input variables  $U$ . The system defines the evolution of the  $X$  variables, based on the values of  $X$  and  $U$ . A controller in turn is a hybrid system over  $V$ , but which defines only the variables in  $U$ , possibly in terms of the values of the  $X$  variables. The controlled plant can thus be seen to be describing behaviours (taken here to be “signals” over  $V$ ). A specification is a hybrid automaton that defines signals over  $V$ , possibly using extra continuous variables to do this. A *conflict-tolerant* specification is one which describes for any plant behaviour  $\sigma$  a set of advised future behaviours  $\mathcal{S}(\sigma)$ . A conflict-tolerant controller is no different from a classical controller, but satisfies the tolerant spec  $S$  iff after every possible uncontrolled plant behaviour  $\sigma$ , the plant behaviour controlled according to the controllers advice, is contained in  $\mathcal{S}(\sigma)$ .

Fig 7 shows a plant model of a water tank where  $w$  represents the level of water in the tank, and  $p$  is a controllable switched input which turns a pump on ( $p = 1$ ) or off ( $p = 0$ ) to pump in water into the tank.

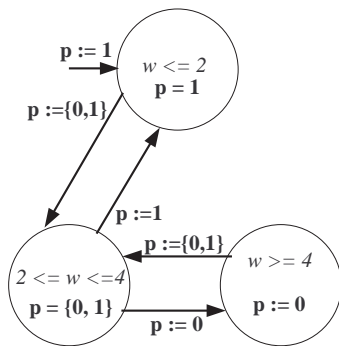
Fig 8 shows a conflict-tolerant specification which requires the water level to be maintained between 2 and 4. When the level goes outside this range, its advice is to restore the level within 2 seconds. The states  $p_1$  and  $p_2$  are used by the specification to track a given plant behaviour  $\sigma$  to reach a given configuration  $(q, \mathbf{v})$ . The states to the right ( $q_1, q_2$ ) denote the behaviours that are advised after  $\sigma$ , and these behaviours are obtained by taking the dashed transition starting in the configuration  $(q, \mathbf{v})$ , and then generating the allowed behaviours.

Finally Fig 9 shows a controller for the plant which controls the input  $p$  to the plant, and which meets the given tolerant specification.

In summary, we have proposed a novel framework for supervisory control which has several advantages. Here one begins with a modular conflict-tolerant specification for each controller which is *independent* of other features envisaged for the system. A vendor must now design a controller which meets the given tolerant specification. Each individual controller can be validated against its tolerant specification, sometimes in an automated fashion,



**Fig. 8. A conflict-tolerant specification.**



**Fig. 9. A conflict-tolerant controller.**

using some of the techniques we have described. Finally, a product containing several features, with a given prioritization of these features, can be synthesized easily from the collection of controllers. The product thus constructed is guaranteed to be “correct” by construction.

Most of the work described here appears in a couple of earlier papers by the authors in [4], [5], and some of it is ongoing work whose details will be published elsewhere.

## References

- [1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
- [2] Y. L. Chen, S. Lafortune, and F. Lin. Modular supervisory control with priorities for discrete event systems. In *Conf. on Decision and Control*, pages 409–415. IEEE, 1995.
- [3] W. Damm and M. Cohen. Advanced validation techniques meet complexity challenge in embedded software development. In *Embedded Systems Journal*, 2001.
- [4] Deepak D’Souza and Madhu Gopinathan. Conflict-tolerant features. In Aarti Gupta and Sharad Malik, editors, *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 227–239. Springer, 2008.
- [5] Deepak D’Souza, Madhu Gopinathan, S. Ramesh, and Prahладavaran Sampath. Conflict-tolerant real-time features. In *QEST*, pages 274–283. IEEE Computer Society, 2008.

- [6] Stefan Ferber, Jürgen Haag, and Juha Savolainen. Feature interaction and dependencies: Modeling features for reengineering a legacy product line. In *SPLC*, pages 235–256, 2002.
- [7] Robert J. Hall. Feature interactions in electronic mail. In *FIW*, pages 67–82, 2000.
- [8] Jonathan D. Hay and Joanne M. Atlee. Composing features and resolving interactions. In *SIGSOFT Found. of Softw. Engg.*, pages 110–119, 2000.
- [9] Dirk O. Keck and Paul J. Kühn. The feature and service interaction problem in telecommunications systems. a survey. *IEEE Trans. Software Eng.*, 24(10):779–796, 1998.
- [10] Peter J. G. Ramadge and W. Murray Wonham. The control of discrete event systems. In *Proc. of the IEEE*, volume 77, pages 81–98, 1989.
- [11] Claire Tomlin, John Lygeros, and Shankar Sastry. Synthesizing controllers for nonlinear hybrid systems. In Thomas A. Henzinger and Shankar Sastry, editors, *HSCC*, volume 1386 of *Lecture Notes in Computer Science*, pages 360–373. Springer, 1998.
- [12] K. C. Wong, J. G. Thistle, H. H. Hoang, and R. P. Malhamé. Supervisory control of distributed systems: Conflict resolution. In *Conf. on Decision and Control*, pages 416–421. IEEE, 1995.