

# DESIGN OF ADAPTIVE SUPERVISORS FOR DISCRETE EVENT SYSTEMS VIA LEARNING

Diana Gordon

Naval Research Laboratory, Code 5515  
Washington, D.C. 20375-5337  
Email: [gordon@aic.nrl.navy.mil](mailto:gordon@aic.nrl.navy.mil)

Kiriakos Kiriakidis

Department of Weapons and Systems Engineering  
United States Naval Academy  
Mail Stop 14a  
Annapolis, MD 21402  
Email: [kiriakid@novell.nadn.navy.mil](mailto:kiriakid@novell.nadn.navy.mil)

## ABSTRACT

In many practical applications, modeling using event-driven dynamics leads to interconnected discrete event systems. Often, these tend to be large-scale event-varying structures, which need to possess certain event-invariant properties. Although supervisory control theory offers some methods for the synthesis of such discrete event systems, adaptive supervision is, by and large, an open problem. This paper proposes an approach to the design of adaptive supervisors based on a systematic revision of the desirable language, via a learning mechanism, so that the system's properties are safe.

## INTRODUCTION

Interconnected Discrete Event Systems (IDES) provide for the modeling of practical systems in the fields of communications, flexible manufacturing, and traffic management. In applications such as these, it is often the case that the IDES must satisfy important behavioral constraints. Given the complexity of many IDES, assuring such constraints is a significant challenge. An appealing solution is the method of supervisory control of discrete event systems, the building blocks of IDES. Supervisory control addresses the problem of coordi-

nating IDES, albeit on the assumption that the system is fixed and known [1–5]. A particularly thorny issue that arises in practical IDES, however, is subjection to change. For example, at the highest level, an existing subsystem may fail or a new subsystem may be connected. At a lower level, the structure of a subsystem may change as well. In turn, such structural changes hamper the application of supervisory control theory to the class of IDES. At present, the literature offers but a few works on adaptive or robust supervisory control to tackle the problem of uncertainty in the IDES model [6–8].

The objective of this paper is the development of a design method for adaptive IDES control that can handle a particular kind of uncertainty, namely, subsystem failure. We propose the following approach. As a first step, a desired language in the form of a Finite State Automaton (FSA) is specified. This desired language, which specifies IDES coordination behavior, is required to satisfy behavioral constraints, called “properties.” An example of a property might be that two subsystems should never execute conflicting events (actions). In order to guarantee that the desired language satisfies the properties, it is advisable to check this using formal verification. FSA repair is done if the outcome of

verification is that properties are not satisfied. Next, a supervisor is automatically generated from the desired language. The supervisor enables and disables events in the IDES to ensure coordination as specified by the desired language.

Because the properties are important, they must be maintained regardless of structural changes. Suppose a subsystem fails. Then a learning mechanism deletes from the desired language events that pertain exclusively to the failed subsystem. It then patches the desired language FSA in accordance with an automaton repair algorithm. A new supervisor is then synthesized from the new desired language FSA to close loop around the IDES.

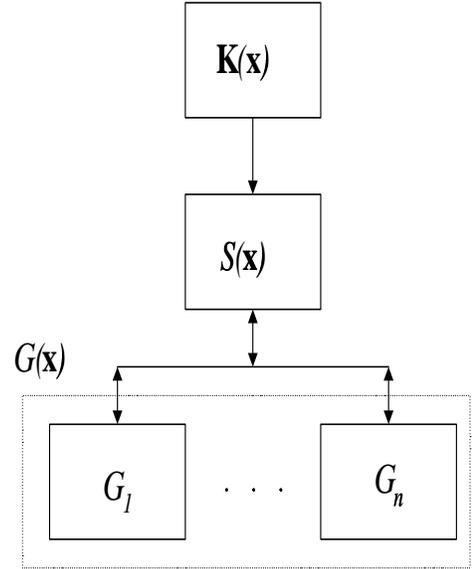
The organization of this paper is as follows. First, we formulate the problem and describe the proposed solution. Second, we present a repair algorithm. Third, we illustrate the advocated approach through a simulated example.

## PROBLEM FORMULATION

Let us denote as  $G$  an IDES that comprises a finite number of subsystems, namely,  $G_1, \dots, G_n$ . To consider the uncertainty of the internal structure of  $G$ , we define the event-valued variable  $\xi$  such that  $\xi = \xi_i$  implies that the  $i$ -th subsystem is off-line. The event that all subsystems are on-line is  $\xi_0$ . With this in mind, we write  $G = G(\mathbf{x})$ , where  $\mathbf{x}$  is a sequence of events  $\xi_i$ . Hereafter, we assume there is a core of subsystems that are never off-line and hence remain part of the IDES after all events  $\xi_i$  have occurred.

Suppose that a supervisor  $S(\mathbf{x})$  exists so that the closed loop with the IDES,  $G(\mathbf{x})/S(\mathbf{x})$ , executes a desired lan-

guage,  $\mathbf{K}(\mathbf{x})$ ; see Figure 1. By dictating certain se-



**Figure 1:** The supervised IDES

quences of events, one embeds in the desired language important properties that are required to hold for the overall operation of the IDES. Because the properties need to hold in spite of structural changes that may occur to the IDES, it is necessary that the core of subsystems generates the prescribed sequences of events. For example, on the assumption that the second and fifth subsystems are part of the core, one considers the following property

*P:* If the fifth subsystem executes “pause”, the second will eventually execute “take action”

Suppose an event  $\xi$  occurs and, because of the structural changes this implies, the closed loop of the resulting IDES,  $G(\mathbf{x}\xi)$ , with the existing supervisor,  $S(\mathbf{x})$ , no longer executes  $\mathbf{K}(\mathbf{x})$ . In turn, the closed loop loses one or more of the aforementioned properties. Herein, we propose an adaptive scheme to obtain a new supervisor,  $S(\mathbf{x}\xi)$ , by taking into account the structural changes in

the system. By design, the new supervisor results in a closed loop,  $G(\mathbf{x}\xi)/S(\mathbf{x}\xi)$ , that possesses the desired properties. In particular, upon occurrence of the event  $\xi$ , a learning algorithm modifies the desired language by removing from it the events that pertain to the failed subsystem. Then, one synthesizes the new supervisor based on the resulting desired language.

### AUTOMATON REPAIR ALGORITHM

As stated above, adaptation occurs via a learning mechanism that deletes from the desired language FSA events pertaining to a subsystem has just been off-line. A problem that arises from this deletion of events is loss of continuity. In particular, deleting events generally implies deleting FSA transitions, which can fracture the FSA into isolated or unusable components or both. An example of an unusable component is a state from which no transition is possible. To restore continuity, the FSA needs to be repaired.

The repair algorithm presented here is guaranteed to preserve all properties with no re-verification required. This is because it only deletes, and never adds, FSA transitions. If FSA transitions are only deleted, then all properties are guaranteed to be preserved [9]. If the IDES must respond quickly, this algorithm saves time compared with the alternate repair algorithm that requires re-verification [10]. The trade-off is that it typically results in a smaller desired language than [10].

Figure 2 shows the algorithm for repairing the desired language FSA after learning occurs.  $\delta(s_i, \sigma) = s_j$  is the FSA transition function, where  $s_i$  and  $s_j$  are states and  $\sigma$  is an event. The subscripts “pre” and “post” denote pre- and post-learning.  $EVENTS_{pre}$  and  $EVENTS_{post}$  are the sets of all pre- and post-

learning events. STATES is the set of all FSA states.  $SUCC_{post}(s)$  is the set of all post-learning successors of state  $s$ . Prior to calling the algorithm of Figure 2,

```

procedure repair-method ( $s$ )
  visited( $s$ ) = true;
  for each  $\sigma \in EVENTS_{pre}$  do
    if ( $(\delta_{pre}(s, \sigma) \neq 0)$  and ( $visited(\delta_{pre}(s, \sigma)) == false$ )) then
      repair-method( $\delta_{pre}(s, \sigma)$ );
    fi
  od
  if ( $SUCC_{post}(s) == \emptyset$ ) then
    for each  $s' \in STATES$  do
      for each  $\sigma \in EVENTS_{post}$  do
        if ( $\delta_{post}(s', \sigma) == s$ ) then
           $\delta_{post}(s', \sigma) = 0$ ;
        fi
      od
    od
  fi
end

```

**Figure 2:** The repair algorithm.

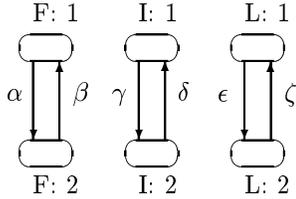
“visited” is initialized to false for every state in the FSA. Procedure “repair-method” is then called with the initial FSA state as parameter  $s$ . Procedure repair-method does a depth-first search through the set of all pre-learning states that are accessible from the initial state, and “visited” is set to avoid re-visitation. When the algorithm has visited all states, it pops the recursion stack to re-visit the states beginning with the last (i.e., it then visits the states in reverse order). For each state  $s$  visited in reverse order, the algorithm tests whether  $s$  has any post-learning successors. If not, it deletes all post-learning pointers to that state. Optionally, the agent may further simplify the FSA, e.g., by removing unused states.

Let us consider the worst-case time complexity of

the repair algorithm. In this analysis, we exclude the time to perform optional FSA simplifications. The worst-case time complexity of the algorithm is  $O((|STATES| * |EVENTS_{pre}|) + (|STATES|^2 * |EVENTS_{post}|))$ . This is because during the depth-first search the algorithm could try to visit every state with every possible pre-learning event. For each state visited on this search that has no post-learning successors, a pair of nested loops is executed that could entail all states and post-learning events.

### A SIMULATION EXAMPLE

Let us demonstrate the advocated adaptive supervisory control approach on an example scenario inspired by the Pathfinder mission to Mars. Figure 3 depicts a simplified model for the collection, the short-, and long-range transmission of packages of data. The sys-



**Figure 3:** The simplified model of the Mars mission

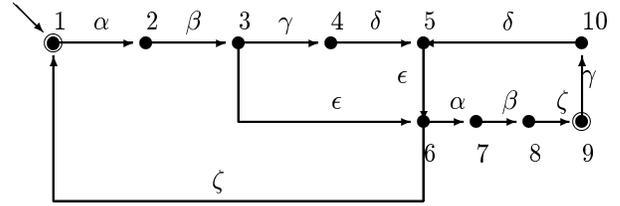
tem comprises the following subsystems: a far rover (F), an intermediary rover (I), and a lander (L). The events  $\alpha, \dots, \zeta$  are controllable. The states  $s_X$  of each automaton X are as follows

F: collecting ( $s_F = 1$ ), transmitting ( $s_F = 2$ )

I: receiving ( $s_I = 1$ ), transmitting ( $s_I = 2$ )

L: receiving ( $s_L = 1$ ), transmitting ( $s_L = 2$ )

The current desired language,  $\mathbf{K}(\xi_0)$ , is the language of the FSA  $D(\xi_0)$  in Figure 4. Alternatively, we represent such FSA in the form of Table 1. Clearly, the desired



**Figure 4:** The desired language FSA,  $D(\xi_0)$ , of the Mars mission

**Table 1:** The transition table of  $\mathbf{K}(\xi_0)$

		event					
		$\alpha$	$\beta$	$\gamma$	$\delta$	$\epsilon$	$\zeta$
state	1	2	0	0	0	0	0
	2	0	3	0	0	0	0
	3	0	0	4	0	6	0
	4	0	0	0	5	0	0
	5	0	0	0	0	6	0
	6	7	0	0	0	0	1
	7	0	8	0	0	0	0
	8	0	0	0	0	0	9
	9	0	0	10	0	0	0
	10	0	0	0	5	0	0

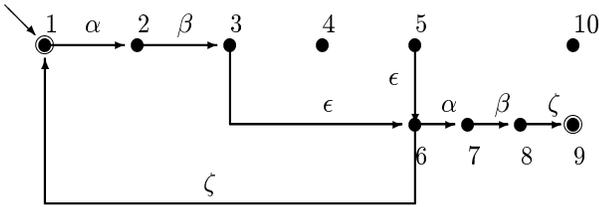
language  $\mathbf{K}(\xi_0)$  specifies that the supervisor will enable event  $\epsilon$  once  $\beta$  has occurred. In words, the FSA of  $\mathbf{K}(\xi_0)$  satisfies the following property:

*P:* If F transmits, L will eventually transmit

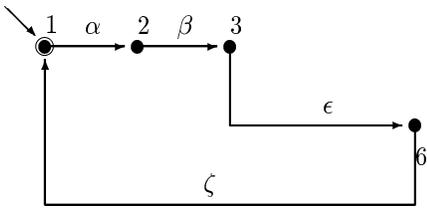
Because we assume such a property is essential for the system's operation, the design needs to guarantee that it continues to hold in spite of changes in the system.

Suppose the event  $\xi_I$  now occurs, i.e., the intermediary rover, I, suddenly is off-line. During the learning stage, the proposed adaptive scheme removes from the

desired language the events  $\gamma$  and  $\delta$ , as in Figure 5—these events affect the I subsystem only. In turn, it restores continuity to the desired language FSA using the repair algorithm. The result is the new language,  $\mathbf{K}(\xi_0\xi_I)$ , in Figure 6. Finally, a new supervisor is automatically generated from  $\mathbf{K}(\xi_0\xi_I)$  to enable or disable events in accordance with the FSA of Figure 6.



**Figure 5:** The desired language FSA after learning



**Figure 6:** The desired language FSA after repair method and simplification

## CONCLUSION

Herein, we treat an IDES as inherently uncertain, in the sense that some of its subsystems will eventually be off-line. To provide supervision in the presence of such uncertainty, we propose an approach to adapt the supervisor by modifying the desired language of the IDES via a learning mechanism. The algorithm that performs the modification guarantees that the desired language maintains certain system properties without

need for re-verification. As a building block, the network of a supervisor and a group of subsystems, where subsystems will be on-line or off-line, fits in an architecture that comprises a number of such dynamic groups of subsystems and hence more than one supervisor. The proposed approach seems capable of handling the design of IDES in this larger scale as well.

## ACKNOWLEDGMENTS

This work was supported by the Office of Naval Research N0001499WR20010, the Naval Academy Research Council, and the Office of Naval Research Grant N0001499WR20020.

## REFERENCES

- [1] P. J. Ramadge and W. M. Wonham, “Supervisory control of a class of discrete event processes,” *SIAM J. Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.
- [2] W. M. Wonham and P. J. Ramadge, “On the supremal controllable sublanguage of a given language,” *SIAM J. Control and Optimization*, vol. 25, no. 3, pp. 637–659, 1987.
- [3] Y. Du and S. H. Wang, “Control of discrete-event systems with minimal switching,” *International Journal of Control*, vol. 48, no. 3, pp. 981–991, 1988.
- [4] W. M. Wonham and P. J. Ramadge, “Modular supervisory control of discrete-event systems,” *Mathematics of Control, Signals, and Systems*, vol. 1, no. 1, pp. 13–30, 1988.
- [5] J. G. Thistle, “Supervisory control of discrete event systems,” *Mathematical and Computer Modelling*, vol. 23, no. 11, pp. 25–53, 1996.

- [6] F. Lin, "Robust and adaptive supervisory control of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 38, no. 12, pp. 1842–1852, 1993.
- [7] S. Young and V. K. Garg, "Model uncertainty in discrete event systems," *SIAM J. Control and Optimization*, vol. 33, no. 1, pp. 208–226, 1995.
- [8] Y.-L. Chen, S. Lafortune, and F. Lin, "How to reuse supervisors when discrete event systems evolve," in *Proceedings of the IEEE Conference on Decision and Control*, (San Diego, CA), pp. 1442–1448, Dec. 1997.
- [9] D. Gordon, "Asimovian adaptive agents," *Journal of Artificial Intelligence Research*, 1999 (in press). Also NCARAI Technical Report 97-016.
- [10] D. Gordon and K. Kiriakidis, "Adaptive supervisory control of interconnected discrete event systems," in *IEEE Conference on Control Applications*, (Anchorage, AK), Sept. 2000.