SUPERVISORY CONTROL THEORY: MODELS AND METHODS

W.M. Wonham
Systems Control Group
ECE Department
University of Toronto
wonham@control.utoronto.ca

Abstract

In this informal talk we discuss by means of two examples some issues of DES modeling and approaches to supervisory control design. In particular we illustrate and compare the use of Petri nets, modular automata, and state tree structures.

Keywords: discrete-event systems, supervisory control, Petri nets, automata, state tree structures.

First of all, I'd like to thank the organizers, in particular Dr. Darondeau, for their invitation to give the keynote talk at this workshop.

Some time after rashly accepting, I began to wonder just what a "keynote talk" should be. Maybe one should strike a majestic chord and announce a sweeping agenda... . If so, this talk will fall drastically short. In fact my aim will only be to sketch some ideas and work in progress, perhaps of interest to others as well.

About supervisory control theory, or SCT, the first thing to say is that quite a lot has been accomplished already. We have the beginnings of a nice formal control theory, based on a few simple ideas, which is somewhat appealing esthetically, is amenable to computation, and lends itself to real industrial applications. Nevertheless, it still has quite some distance to go, both in respect to elegance and usefulness. These two features belong together, inasmuch as elegance implies conciseness and economy of concepts, and therefore efficiency of transmission to potential users.

My first theme, however, is that we could try to be more flexible about DES models. Some of us like to build systems up from simple components modelled as automata, and others like to build them out of Petri nets. In the literature these days there seems to be a sort of tacit competition, as the PN community adapts SCT concepts to their own methods, and makes much out of the claimed superior "power and compactness" of PN models. Alas, there is no free lunch. No matter what framework you choose, the basic interesting problems are exponentially hard, and the controversy seems to reduce to issues of cultural tradition and what computing tools happen to be available at any given time to the group you belong to.

Maybe, however, we could have the best of both worlds. Here's a small example, borrowed from Murat Uzam [Uza02]. Apart from its smallness, which is only for display purposes, it's fairly typical of examples in the PN literature nowadays. Much larger examples of this type have been worked out.

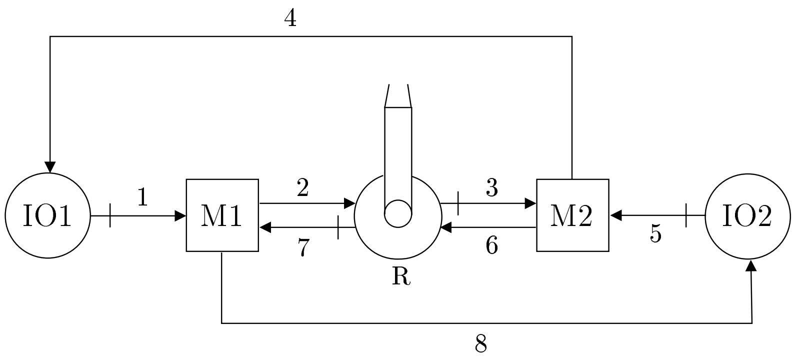

Fig. 1. FMS block diagram labelled with shared events [Uza02].

Part1 production cycle: 1 2 3 4
Part2 production cycle: 5 6 7 8

Deadlock sequence: 1 2 1 5

R holds a Part1 (expects 3)
M1 holds a Part1 (expects 2)
M2 holds a Part2 (expects 6)

FMS state size: 20
Number of noncoreachable states: 5
T(rimmed) FMS model state size: 15

The system of Fig. 1 is an FMS consisting of two machines and a robot, which can process 2 different part types. The number of concurrent activities which can take place for each part type is 3. Because the machines share use of the robot (resource), the system is prone to block in the absence of control, in fact to deadlock. The imposed control specification is, therefore, nonblocking with respect to the initial state (machines and robot idle, each input buffer full with 3 parts each). The computed SCT results exactly match Uzam's. FMS is the synchronous product of the machines, robot and buffer, and has 20 states (matching the size of Uzam's reachability graph of the PN model). Five of these states are noncoreachable, matching Uzam's Deadlock Zone (DZ). The controlled behavior with respect to the specification of nonblocking is exactly the structure of the trimmed plant TFMS (FMS with noncoreachable states removed). It has 15 states.

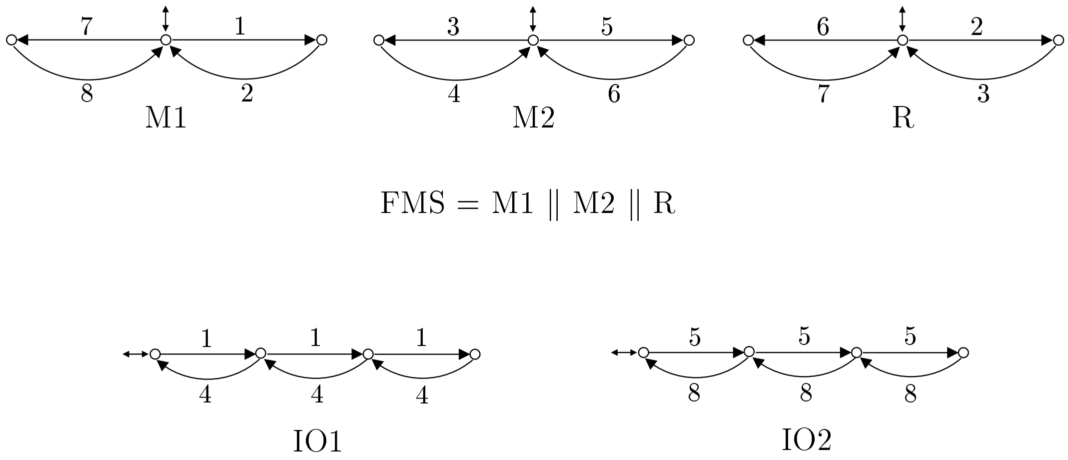

Fig. 2. FMS component state graphs.

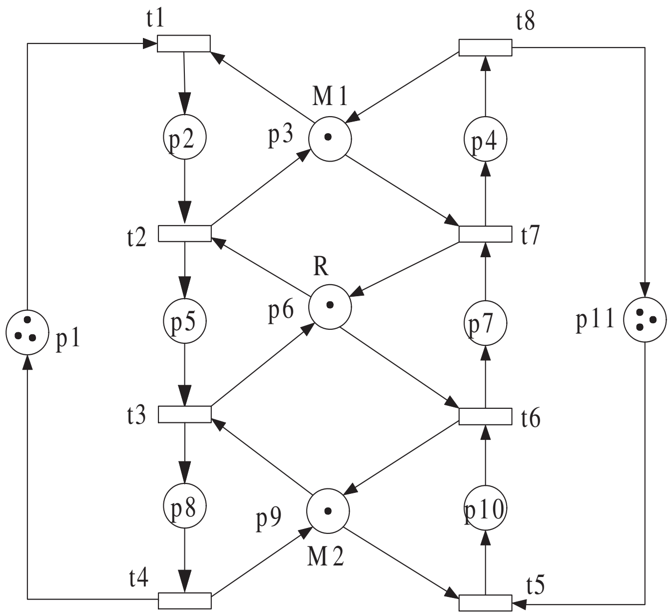

Fig. 3. FMS Petri net model [Uza02].

Figs. 2,3 display the component state graphs and Uzam's PN for the plant. There is not much to choose between them in terms of visual comprehensibility. What one more easily comprehends is not either of these graphs, but the layout of components which gave rise to them (Fig. 1 above).

Only the events *t1,t5* corresponding to the process inputs for part types 1 and 2 ever need to be disabled. The remaining controllable events *t3,t7* could have been declared uncontrollable without changing the control action.

While we could use the controlled behavior itself as a supervisor, it is often possible to simplify matters by reducing the controlled behavior using a "control congruence" [SW03]. In this example the simplified supervisor SIMSUP (Fig. 4) only has 4 states, reduced from 15, and this is the strict lower bound. If you couple SIMSUP to FMS, using synchronous product, you get exactly the desired optimal controlled behavior, or simply TFMS in this example.

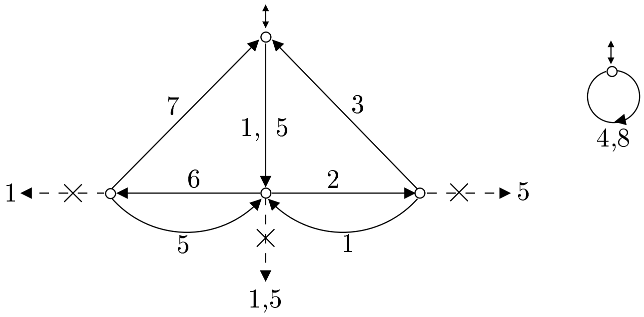

Fig. 4. Simplified supervisor SIMSUP for FMS.

Uzam's synthesis method provides 3 monitors, and is based on the "theory of regions", which is rather elaborate. This application requires, first, that you work out the reachability graph of the PN model (which is exactly what we computed directly as the synchronous product FMS with 20 states), and then that you solve 6 "event/state separation" problems, amounting to 6 systems of linear integer inequalities each of size 15 (the number of coreachable states, which have to be "separated" from the deadlock zone of 5 noncoreachable states). The 3 monitors and the resulting controlled PN model are displayed in Figs. 5,6.

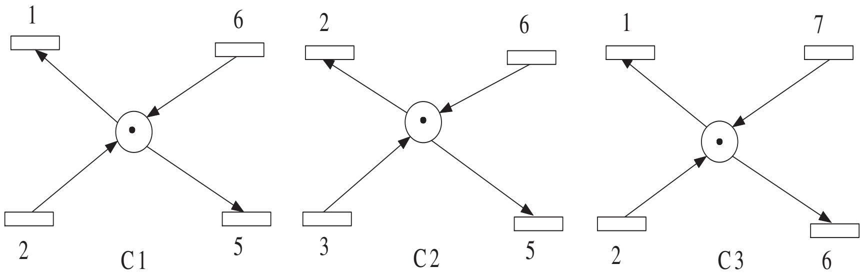

Fig. 5. Control places (monitors) for FMS PN model [Uza02].

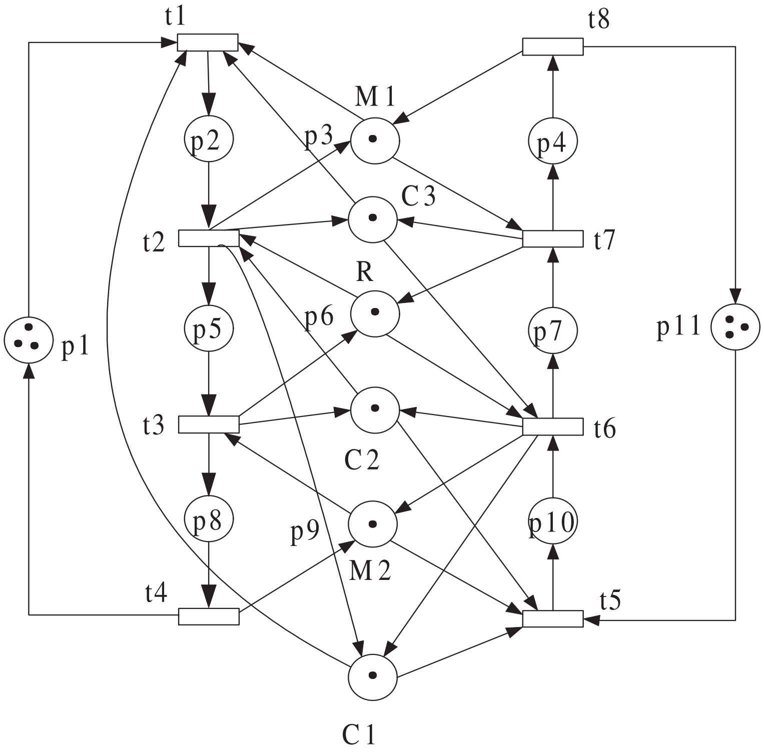

Fig. 6. FMS PN model controlled by monitors [Uza02].

Incidentally, because of its loop structure, our SIMSUP cannot be realized as a language-equivalent Petri net. This illustrates one of the limitations of the PN framework for use in SCT. An SCT plant or supervisor could be any finite automaton, whereas not all finite automata admit a PN representation (within closed behavioral language equivalence). While it seems not yet to be entirely clear just what finite automata do have PN realizations, it is easy to cook up examples which don't, like the one in Fig. 7.

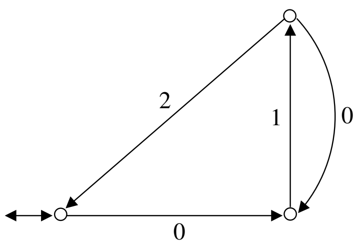

Fig. 7. Automaton having no PN realization.

For this example, even tricky selfloops in your PN won't help. Of course, you could in principle resort to a labelled PN, to take care of the presence in the automaton of transitions with the same label in different locations in the transition structure, but I haven't seen this rather cumbersome expedient exploited so far in the PN control literature. Anyhow, if you really prefer your PN plant model to the SCT automaton model, and you don't want to grapple with the theory of regions, then you can adopt a hybrid approach, again suggested by Uzam [Uza03]. Take your PN, assumed or confirmed to be bounded, work out its reachability graph (which you have to do anyway to take care of the global property of reversibility), then use SCT to compute the supervisor SUPER, simplify SUPER using a control congruence to get SIMSUP, then hook up SIMSUP to your original PN plant model using, if you want, the graphical PN convention of inhibitory arcs to display the disablements. This is entirely straightforward logically, graphically, and technologically. For our toy example, the result is shown in Fig. 8.

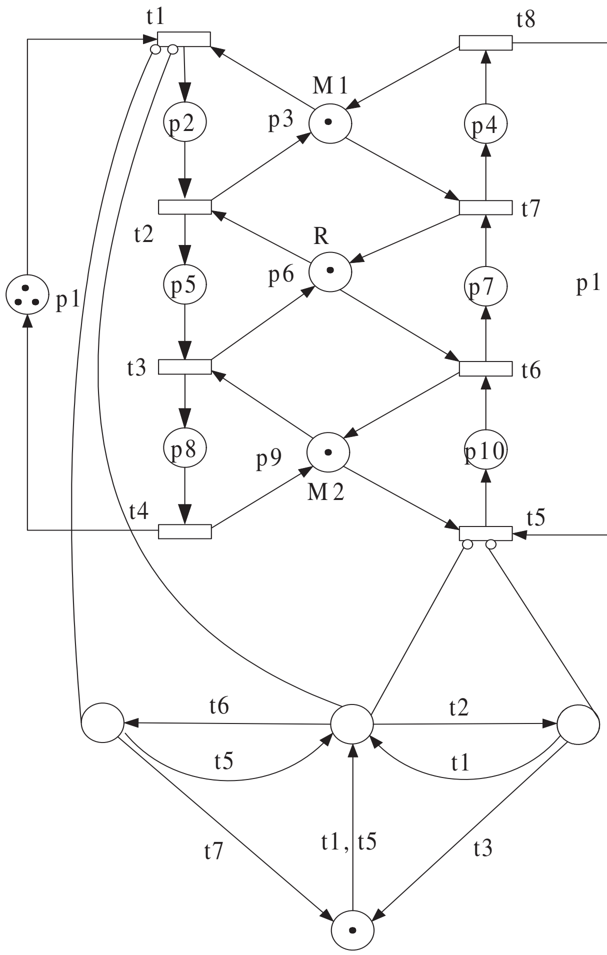

Fig. 8. SIMSUP coupled to FMS PN model with inhibitory arcs [Uza03].

Here the upper (plant PN) and lower (SIMSUP automaton) components are supposed to synchronize on shared events.

What is the moral of the story? Not that one approach is necessarily better than the other, in all or most cases, but rather that the two modelling frameworks can work together. What we are really doing is working in a state space of structure

$$Q_1 \times Q_2 \times \ldots \times Q_m \times \mathbb{N}^k \times \mathbb{B}^l$$

This is the cartesian product of m plain unstructured sets, for component automata, and k-dimensional natural-number space for those parts of our system that have a *natural* vector-additive structure. This will include things like buffers, and more generally any collection of similar objects where we are interested only in the numbers of objects resident in various states (like "idle", "working",...) and not in their individual identity. For good measure we've included a product of l booleans. The underlying set-theoretic control theory is nothing new; the challenge is to work out the algorithms. If this could be done, then in the general theory we could concentrate on the basic (set-theoretic) issues, and not get hung up on issues of, and the limitations of, specific styles of representation (notably automata vs. PN). This would be a highly desirable separation of issues.

Just before leaving our toy FMS, let's note that neither of our proposed "simple" controllers, either by PN monitors or by SIMSUP, is the most humanly comprehensible. Revisiting the block diagram (Fig. 1) you'll probably guess rather quickly that the optimal control is simply this: don't start a Part1 cycle as long as there's a Part2 somewhere in progress, and vice-versa for a Part2 cycle. This control is easily implemented by 2 modular controllers based on I01 and I02 (Fig. 9), and the verification of optimality is immediate. This illustrates a second moral, well known, that even the simplest answers produced by formal synthesis needn't be the most appealing to human engineers. But how to incorporate user-friendly "intelligence" into our synthesis methods remains a mystery.

Fig. 9. Modular supervisor for FMS.

Moving on in this vein, we suggest that doing the computations for systems of industrial size may not be much of a problem nowadays. The computing technology of boolean and integer decision diagrams (BDDs, IDDs) has moved the notion of "large" from where it was, say at around a million states 15 years ago, to where it is now, say at 10^{20} states, or almost wherever you need it. So the issue in modelling is not raw state size, but much more the issue of comprehensibility. No one is going to fly in something with a million states (let alone 10^{20}) unless they can understand how it was put together. So the real challenge nowadays is complex system representation, or in other words, architecture.

From an algebraic point of view, developing an architecture means picking a small number of constructs and operators, and being clear about how they interrelate. In model checking this is known as "compositional reasoning" [BCC98], but in any case it is quite standard algebraic procedure. In SCT a typical instance is the following. Say you want to construct a large system out of two smaller systems, for concreteness the synchronous product G of two DES $G1$ and $G2$: $G = G1 \parallel G2$. You would like to know, for instance, when the large system G inherits the property of nonblocking from the smaller systems. In terms of marked and closed behaviors, you want to know if

$$\overline{L_m(G)} =?\ L(G) \qquad \text{[overbar means prefix-closure]}$$

namely

$$\overline{L_m(G)} := \overline{L_m(G1) \parallel L_m(G2)} =?\ \overline{L_m(G1)} \parallel \overline{L_m(G2)} = L(G1) \parallel L(G2) =: L(G)$$

Here we've assumed that $G1$ and $G2$ are independently nonblocking by themselves. So we want to know when prefix-closure distributes across synchronous product. This question can be answered in several ways [Won02], but the point here is that it's just one of a number of basic algebraic "laws" that won't hold in general but will hold under certain structural restrictions. These restrictions will provide an advance guarantee that a nice property of the large system will flow from the same nice property of the smaller components, provided the latter's interaction is suitably controlled. In SCT there are a number of basic ingredients of this sort, notably the construct synchronous product, the operator prefix-closure, the supremal controllable sublanguage $\sup_G \mathcal{C}(\cdot)$ operator, and of course the abstraction operator, say by causal projection θ, that moves languages from a low-level alphabet to their abstract images higher up. Working out the details leads to a better understanding of the basic commutative diagram of hierarchical control, as in Fig. 10, and its extension to "heterarchical control" where you have synchronous products at both top and bottom.

$$\begin{tikzcd}
L_{\mathrm{hi}} & E_{\mathrm{hi}} \arrow[r, "\sup\, \mathcal{C}_{\mathrm{hi}}(\cdot)"] \arrow[d, "\theta^{-1}"'] & K_{\mathrm{hi}} \\
L_{\mathrm{lo}} \arrow[u, "\theta"] & E_{\mathrm{lo}} \arrow[r, "\sup\, \mathcal{C}_{\mathrm{lo}}(\cdot)"'] & K_{\mathrm{lo}} \arrow[u, "\theta"']
\end{tikzcd}$$

Fig. 10. Commutative diagram of hierarchical control.

Another way of looking at this notion of architecture is to start by asking how a large state set might be constructed from smaller ones. The basic constructs for sets are cartesian product and disjoint union. This idea was exploited by David Harel [Har87] many years ago, and led to the idea of state charts, again a way of "understanding" a complex system at a given level by suppressing details lower down. I like to think of these things as "state trees" pictured as follows: A global system at the top, with possibly many concurrent high-level components, is represented as a product, in which each component may be split into disjoint pieces, and the decomposition continues recursively top down, until you get to something sufficiently primitive at the bottom, as shown in Fig. 11. The product factors (resp. disjoint union summands) are called "AND" (resp. "OR") components in this literature.

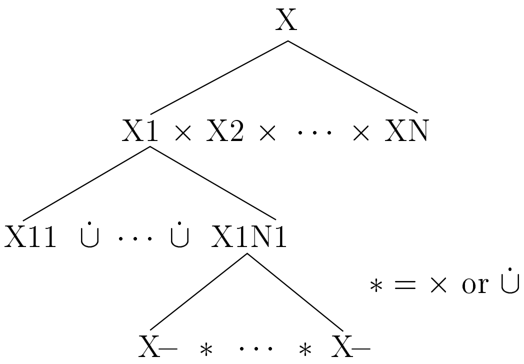

Fig. 11. State tree as a model for structured state space.

The next stage is to supply the dynamics, which can be formalized using an idea akin to the "holons" of Arthur Koestler [Koe89] as adapted to manufacturing systems by Peter Luh and others [GLK97]. We call these "superstates". This program has been carried out for DES, where boolean decision diagrams (BDDs) [McM93] have come to the rescue in actually implementing the computations.

Here's an application to a decently large system called the AIP testbed, which resides in Grenoble, and which was first successfully treated as an SCT problem by Bertil Brandin and François Charbonnier in 1994 [BC94]. Recently Asma Ghaffari has published a presentation in Petri nets [GRX03]. The physical setup, shown in Fig. 12, consists of 3 assembly stations (AS1,AS2,AS3), each with its external conveyor loop and transfer unit (TU1,TU2,TU3) to move pallets from and to a central conveyor loop; a transfer unit TU4 links the central loop to a fourth external loop and I/O station. Two part types can be processed, through different sequences of visits to AS1,AS2, while AS3 acts as a backup station in case AS1 or AS2 breaks down.

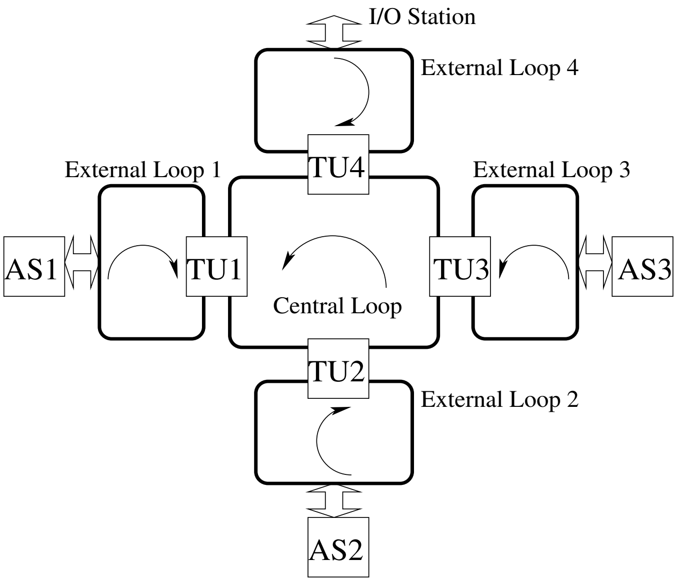

Fig. 12. AIP manufacturing testbed - main component layout.

A state tree version has recently been worked out in his doctoral thesis by Ma Chuan. The representation starts at the root AIP and develops the first-level product structure shown in Fig. 13. Here L1,L2 are buffers that model pallet storage in external loops 1,2.

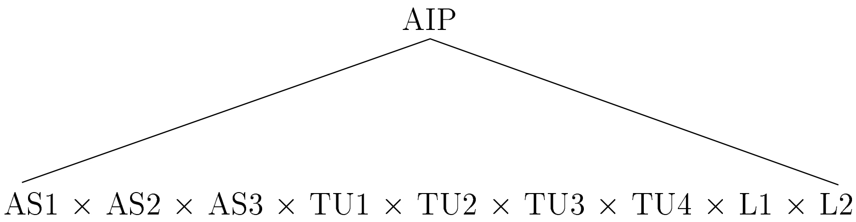

Fig. 13. First level state tree for AIP testbed.

Each of these AND components will be further developed top-down. For instance, a product development of AS1 into three further AND components is displayed in Fig. 14. Physically these are AS1's robot, pallet handler, and workpiece processor, as shown.

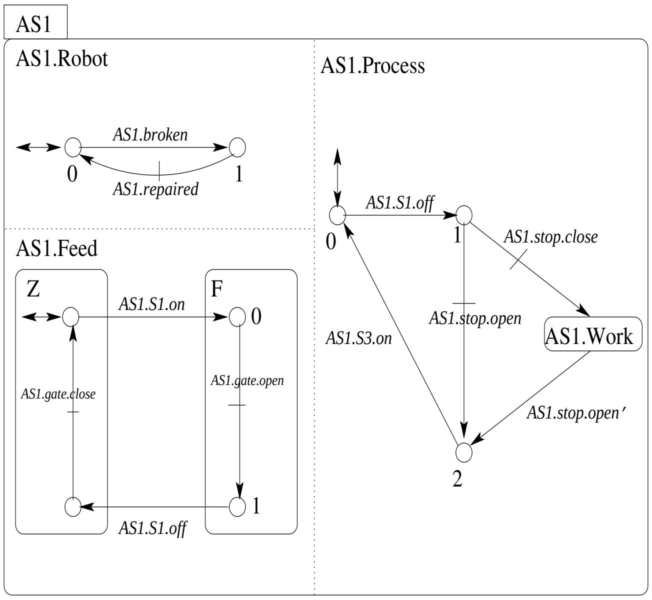

Fig. 14. AND decomposition of AS1.

In turn, AS1's processor has the detailed OR breakdown in Fig. 15, which includes 4 "extractor" superstates ET shown in Fig. 16.

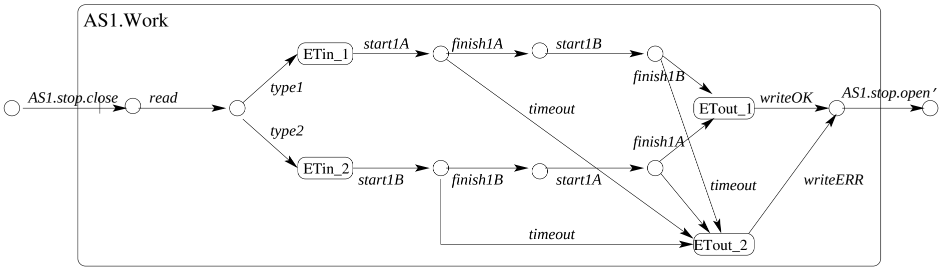

Fig. 15. OR decomposition of AS1.Work (in AS1.Process).

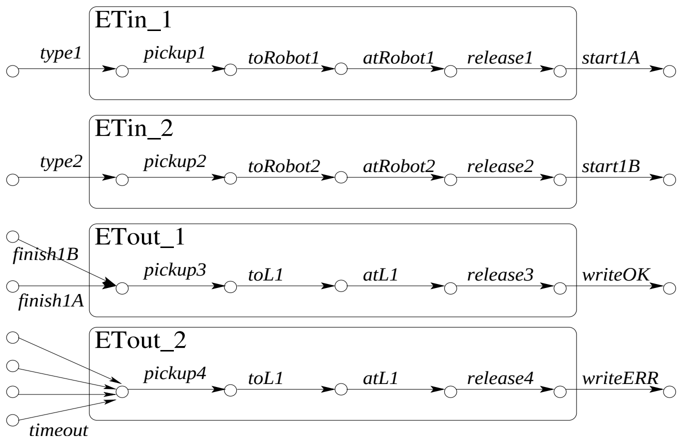

Fig. 16. OR decomposition of superstates ET (in AS1.Work)

The other AND components in Fig. 14 are handled similarly.

The problem is to design a controller, first of all to satisfy various "safety" specifications. These will enforce correct sequencing for the two part types, avoidance of overflow on the conveyors and assembly units, and the logic of redirection in case of breakdown. The modelling of these is based on the standard method of state predicates encoded by BDDs [McM93, MW03].

Much more of a challenge is to enforce nonblocking, so the overall process is guaranteed (if the model is correct!) never to wind up in some impasse, always a risk when you have shared resources, as we already know. As I stressed earlier, nonblocking is a global problem, and so we have to rely on structural features in the modelling if we are to avoid having to treat the whole system in one fell, exponentially large, swoop [Sha23]. The state tree structure itself being recursive, the control synthesis algorithm (for both safety and nonblocking) is recursive as well. Essentially, the computation iterates depth first between a superstate X and its descendants in the state tree, ensuring that all descendants of X are processed before returning from X.

This program was able to compute the controller in about 45 minutes, using a fixed memory allocation (as required by our BDD package), on a PC with 1 GHz Athlon CPU and 256 MB RAM. Specialized tricks were exploited, like "clusters" to divide a large set of AND components into smaller sets according to the pattern of shared events. While the raw state size (of an equivalent "flat" model) would be on the order of 10^{24}, the state tree controller was encoded into a BDD with only about 7×10^4 nodes. It was possible to control the BDD size of intermediate predicates in the computation to less than 3 times that number, as displayed in Fig. 17. A proposed plant/controller loop is shown in Fig. 18. Here the controller's internal plant model AIP-Tracker feeds the current plant state b to BDD-encoded event-enabling functions f_{σ_i}, which could be quickly evaluated online.

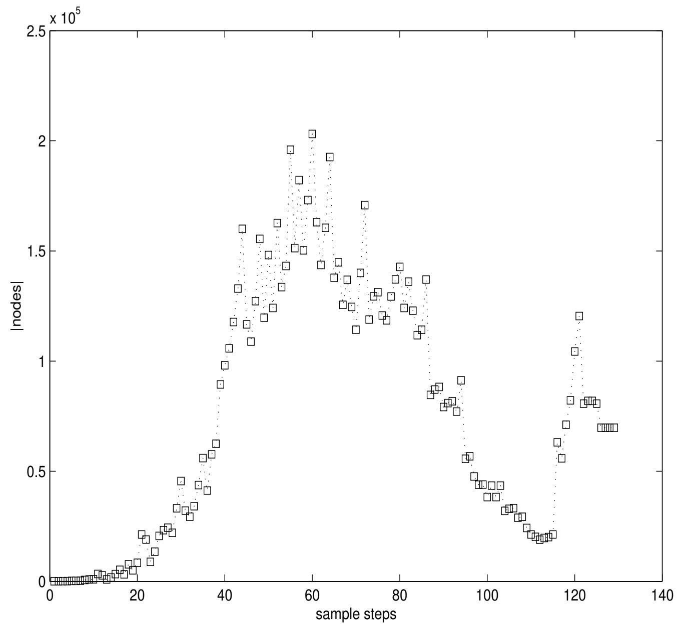

Fig. 17. Size of BDD in computation of AIP controller.

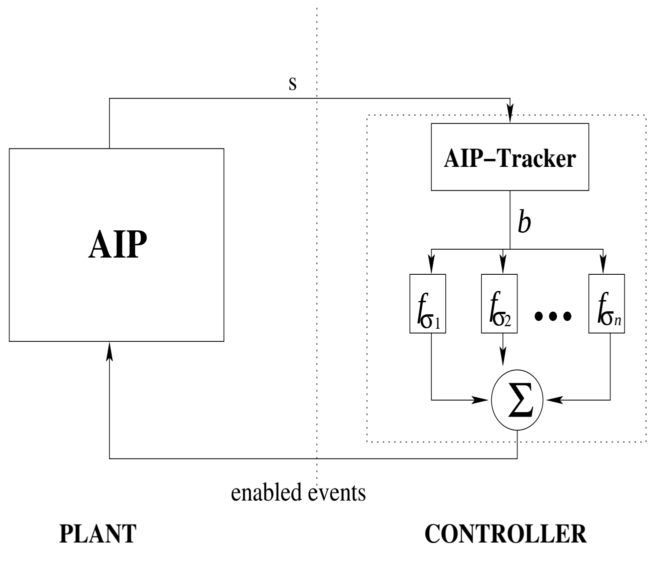

Fig. 18. Plant/Controller loop.

There are many other topics we might explore under the title heading of “models and methods”. One of the more interesting is concurrency and causality, especially lattice-theoretic notions in the spirit of Nielsen [NPW81], Pratt [Pra86], Valmari [Val92] and others. This is work of long standing and, I suggest, well worth revisiting by control specialists, now that we are looking at large systems distributed with respect to space and function.

Acknowledgements

In preparing this talk I've been helped by Chen Shulin, Asma Ghaffari, Li Zhiwu, Ma Chuan, Murat Uzam and Wu Weimin. The opinions expressed, however, are my own.

References

[BC94] B.A. Brandin, F.E. Charbonnier. The supervisory control of the automated manufacturing system of the AIP. Proc. of the Fourth Intnl. Conf. on Computer Integrated Manufacturing and Automation Technology. IEEE Computer Society Press, New York, October 1994, pp. 319-324.

[BCC98] S. Berezin, S. Campos, E.M. Clarke. Compositional reasoning in model checking. Lecture Notes on Computer Science #1536, pp. 81-102. Springer, 1998.

[GLK97] L. Gou, P.B. Luh, Y. Kyoya. Holonic manufacturing scheduling: architecture, cooperation mechanism, and implementation. 1997 IEEE ASME International Conference on Advanced Intelligent Mechatronics (AIM97), Tokyo, Japan 1997; 6 pp.

[GRX03] A. Ghaffari, N. Rezg, X. Xie. Feedback control logic for forbidden-state problems of marked graphs: Application to a real manufacturing system. IEEE Trans. on Automatic Control 48 (1) January 2003, pp. 18-29.

[Har87] D. Harel. Statecharts: a visual formalism for complex systems. Science of Computer Programming 8 (3) 1987, pp. 231-274. North-Holland.

[Koe89] Arthur Koestler. *The Ghost in the Machine*. Hutchinson 1967, Arkana 1989.

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer 1993.

[MW03] C. Ma, W.M. Wonham. Control of state tree structures. 2003 Mediterranean Control Conference, Rhodes, Greece, June 2003.

[NPW81] M. Nielsen, G. Plotkin, G. Winskel. Petri nets, event structures and domains, Part I. Theoretical Computer Science 13, 1981, pp. 85-108.

[Pra86] V. Pratt. Modeling concurrency with partial orders. International J. of Parallel Programming 15 (1) 1986, pp. 33-71.

[Sha23] W. Shakespeare. *Macbeth* IV, iii, 216-217. First Folio, London, 1623, p. 147.

[SW03] R. Su, W.M. Wonham. On supervisor reduction in discrete event systems. Systems Control Group, ECE Dept., U. of Toronto, January 2003 [http://www.control.utoronto.ca/DES].

[Uza02] M. Uzam. An optimal deadlock prevention policy for flexible manufacturing systems using Petri net models with resources and the theory of regions. Int. J. Adv. Manuf. Technology 19, 2002, pp. 192-208.

[Uza03] M. Uzam. Personal communication, February 2003.

[Val92] A. Valmari. A stubborn attack on state explosion. Formal Methods in System Design 1 (4) 1992, pp. 297-322.

[Won02] W.M. Wonham. Notes on Control of Discrete-Event Systems. Systems Control Group, ECE Dept., U. of Toronto, 1 July 2002 [http://www.control.utoronto.ca/DES].