DES Controller Synthesis and Fault Tolerant Control

A Survey of Recent Advances

Rune M. Jensen

IT University Technical Report Series

TR-2003-40

ISSN 1600–6100

December 2003

ISSN 1600–6100

ISBN 87-7949-054-9

Copies may be obtained by contacting:

IT University of Copenhagen
Glentevej 67
DK-2400 Copenhagen NV
Denmark

Telephone:	**+45 38 16 88 88**
Telefax:	**+45 38 16 88 99**
Web	**`www.itu.dk`**

DES Controller Synthesis and Fault Tolerant Control
A Survey of Recent Advances

Rune M. Jensen

Abstract

This report surveys recent advances in computer aided synthesis of controllers for discrete event systems. The report aims at providing an overview of the most successful symbolic synthesis approaches developed within the area of supervisory control. In addition, it describes approaches to fault tolerant control developed within control theory, diagnosis, and planning.

1 Introduction

Today a wide range of powerful tools exist for formal verification of discrete systems (e.g., [16, 22]). Verification problems are decision problems and thus - at a first glance - seem much easier to solve than synthesis problems. There are, however, reasons to believe that this is not always the case in practice:

1. A buggy design may lack the symmetry that makes symbolic approaches scale. For a synthesis problem, the structure of the underlying system is preserved given a correct system model and specification, and these are often fairly easy to obtain,

2. Verification involves reasoning about all the reachable states of the manual design. This set of states may be much larger than the set of states traversed by the synthesis algorithm.

Moreover, software protocols, controllers, and logic designs are tedious to construct and thus error prone. To the largest possible extend, computers should be employed to solve these tasks automatically leaving only the abstract system model and its specification to be manually defined.

So far, however, automated synthesis of controllers for Discrete Event Systems (DES) has received very little attention. In control theory, Ramadge and Wonham have introduced a theoretical framework based on language theory to investigate supervisory control of DES [46]. A considerable amount of work on decidability and controllability has been produced in recent years based on this framework. Only a few synthesis tools, however, have been developed. In automated planning, algorithms have been developed for generating discrete, untimed, and stateless controllers (non-deterministic plans). The focus has been on defining different solution classes and improving the performance of the planning algorithms by combining heuristic and symbolic approaches [31].

The first part of the survey introduces the RW framework and presents work on automated controller synthesis within control theory. The second part of the survey is a more detailed examination of previous work on fault tolerant control. We cover models of fault tolerance and synthesis techniques developed within control theory, planning and diagnosis.

Major information sources are papers and reports generated by the research groups at University of Toronto and Chalmers University of Technology centered around R. Murray Wonham and Martin Fabian. In addition, standard search tools including Google, INSPEC, Citeseer, CAMEO, IEEE Explore, and ACM Digital Library have been used.

The remainder of the survey is organized as follows. In Section 2, we introduce the supervisory control theory of Ramadge and Wonham and its major extensions. Section 2 gives an overview of work on symbolic controller synthesis and in particular describes the approach developed at Chalmers University of Technology. Section 4 describes previous work on fault tolerant control models and synthesis techniques developed in control theory, diagnosis and planning. Finally, we draw conclusions of the survey in Section 5.

2 DES Control Theory

This section introduces Ramadge and Wonham's supervisory control theory (RW theory,[46]) (mostly copied from [12]) and lists subsequent work described in Wonham's DES notes [62]. The motivation for the RW theory is a unified framework to describe DES control [46]:

> "While numerous practical examples are described in the literature on simulation, there is at present time apparently no unifying theory for the control of discrete event processes. Nor is it entirely clear what such a theory ought to encompass. Numerous approaches to the modeling of discrete event processes have appeared in the literature. A general sampling of these could include Boolean models [...]; Petri nets [...]; formal languages; temporal logic [...]; and port automata and networks [...]. All of this work is concerned, in one way or another, with the problem of how to achieve or verify the orderly flow of events; and to this end how to bring together ideas from logic, language and automaton theory. However, while control problems are implicit in much of the work just cited, control-theoretic ideas as such have found little application there. The variety of approaches reflect the diversity of areas in which discrete event processes play an important role. It also indicates that to date no dominant paradigm has emerged upon which a theory of control might be based."

Thus the purpose of the RW theory is to provide a general framework for DES control theory research.

2.1 Basic Concepts of the RW Theory

The plant to be controlled is modeled by an automaton $G = (\Sigma, Q, \delta, q_0, Q_m)$ where Σ is an alphabet of event labels, Q is a set of states, $q_0 \in Q$ is the initial state, $Q_m \subseteq Q$ is the set of *marker states*, and $\delta : \Sigma \times Q \to Q$, the transition function, is a partial function defined at each state in Q for a subset of Σ. The plant is assumed to be an automaton of the physical plant restricted to behavior satisfying its specification.

Let Σ^* denote the set of all finite string over Σ including the null string ϵ. Then δ can be extended to Σ^* by defining $\delta(\epsilon, q) := q$ and $(\forall \sigma \in \Sigma, s \in \Sigma^*)\ \delta(s\sigma, q) := \delta(\sigma, \delta(s, q))$. Then G is characterized by two subsets of Σ^* called the *closed behavior* of G, written $L(G)$, and the *marked behavior* of G, written $L_m(G)$. The language $L(G)$ is defined as

$$L(G) := \{s | s \in \Sigma^* \text{ and } \delta(s, q_0) \text{ is defined}\}$$

and is interpreted to mean the set of all possible event sequences which the plant may generate. The language $L_m(G)$ is defined as

$$L_m(G) := \{s | s \in \Sigma^* \text{ and } \delta(s, q_0) \in Q_m\}$$

and is intended to distinguish some subset of possible plant behavior as representing completed tasks.

To impose supervision on the plant, we define some of its events as *controllable* and some as *uncontrollable*, thereby partitioning Σ into the disjoint sets Σ_c, the set of controllable events, and Σ_{uc}, the set of uncontrollable events. A supervisor is then an agent which observes a sequence of events as it is generated by G and enables or disables any of the controllable events at any point in time throughout its observation. By performing such a manipulation of controllable events, the supervisor ensures that only a subset of $L(G)$, called a *sublanguage*, is permitted to be generated. Formally, a *supervisor* $\mathcal{S}$ is a pair (S, ϕ) where S is an automaton which recognizes a language over the same event set as the plant G and ϕ called a *feedback map*, is a map from the event set and states of S to the set {1(enable),0(disable)}. If X denotes the set of states of S, then $\phi : \Sigma \times X \to \{1, 0\}$ satisfies $\phi(\sigma, x) = 1$ if $\sigma \in \Sigma_{uc}, x \in X$ and $\phi(\sigma, x) = \{1, 0\}$ if $\sigma \in \Sigma_c, x \in X$. The automaton S tracks and controls the behavior of G. It changes state according to the event generated by G and in turn, at each state x of S, the control rule $\phi(\sigma, x)$ dictates whether σ is to be enabled or disabled at the corresponding state of G. The behavior of the closed loop system, i.e., the sequence of events generated while the plant is under the control of $\mathcal{S} = (S, \phi)$, is represented by an automaton $\mathcal{S}/G$ whose closed behavior, denoted by $L(\mathcal{S}/G)$, permits a string to be generated if the string is in both G and S and if each event in the string is enabled by ϕ. The *marked behavior* of $L_m(\mathcal{S}/G)$ is,

$$L_m(\mathcal{S}/G) := L(\mathcal{S}/G) \cap L_m(G)$$

Thus the marked behavior of $\mathcal{S}/G$ consists exactly of the strings in $L_m(G)$ that "survive" under supervision of $\mathcal{S}$. Let $\overline{L}_m(\mathcal{S}/G)$ denote the prefix closure of $L_m(\mathcal{S}/G)$. We say that $\mathcal{S}$ is *non-blocking* (for G) if

$$\overline{L}_m(\mathcal{S}/G) = L(\mathcal{S}/G).$$

That is, each string that can be generated by the controlled system can be extended to a marked string. An approach for synthesizing least restrictive non-blocking supervisors is provided in the basic RW theory.

In [2] it is noted that the RW framework is somewhat awkward for real systems. The plant model proposed in supervisory control theory assumes that the plant "generates" events spontaneously. The control mechanism of the controller is to disable some of these events. However, real systems usually reacts to input commands generated by the controller. Thus, it is more natural to consider controllable events inputs to the plant and uncontrollable events outputs from the plant.

In the DES literature it is further unclear what the main purpose of the supervisor is: to ensure that the supervised system produce the marked language, *or* to make the supervised system reach the marked states. RW theory focus on least restrictive non-blocking supervisors, but these seems to be the least suited for reaching the marked states. Thus, in classical RW theory the object of the supervisor is the primary. However, the latter view is taken in several subsequent works (e.g., for generating optimal and fault tolerant controllers [56, 12, 13]).

2.2 Modular Synthesis

In this approach [47, 38, 49] the overall supervisory task is divided into two or more subtasks. Each of the latter is solved with the ordinary RW approach, and the resulting individual supervisors are run concurrently to implement a solution of the original problem. One of the problems studied is under which conditions the local supervisors form a global non-blocking supervisor.

2.3 Hierarchical Supervision

The broader the temporal horizon of a control and its associated subtasks, or the deeper its logical dependency on other controls and subtasks, the higher it is said to reside in a control hierarchy. The question addressed is mainly *hierarchical consistency* e.g., under what circumstances timely and sufficiently detailed information can be send to the high-level supervisor for various critical low-level situations to be distinguished (e.g., to achieve non-blocking control) [64, 61].

2.4 Supervisory Control with Partial Observation

It is assumed that only a subset of the event labels generated by the plant is observable. It is still assumed, though, that the supervisor may disable unobservable events. The presented work, investigates the basic control concepts under this restriction [38].

3 Symbolic Controller Synthesis

Most control theory papers do not focus on developing efficient algorithms and data structures for synthesizing controllers. Algorithms are often provided that are polynomial in the size of the state space, but given the state explosion problem, such algorithms are not of much practical use. Wonham's group has developed a computer program called TCT based on explicit state manipulation. It was improved by a Masters student (SmartTCT, [63]) using Integer Decision Diagrams (IDDs). However, this work seems intellectually isolated. In particular, it has no references to related work in symbolic model checking and re-invents basic techniques such as automatic variable reordering and caching. Another synthesizer is Lafortune's UMDES, but it apparently also uses an explicit state representation.

Symbolic synthesis of supervisory controllers was probably first investigated by [27]. It has been shown in [46] that the least restrictive non-blocking supervisor can be computed as a greatest fixed point. In [27], this fixed point computation is performed using the VER package by David Dill, Andreas Dexler, and Alan Hi based on Brace, Rudell, and Bryant's BDD package [7]. The paper does not mention application of sophisticated techniques such as transition relation partitioning and variable reordering. Their approach is elaborated in [2] which is widely cited.

Model checkers have been used to verify controllers. A recent paper summarizes model checking approaches for verifying and synthesizing controllers for embedded systems [8]. SPIN [28] and UppAll [3] have been used to verify a timed batch plant controller as a part of the European Esprit project on verification of hybrid systems (VHS). Another work rooted in computer science is [1]. This paper provides a simple but general definition of controller synthesis covering DES including discrete games (via a *controllable predecessors* function). They then show the straight forward symbolic solution to the problem. However, no particular efficient symbolic model checking techniques are introduced. Verification of PLC code using SMV is described in [48].

3.1 High Performing Approaches

The only work we are aware of applying state-of-the-art symbolic approaches to reduce the complexity of supervisory control synthesis is by Arash Vahidi, Bengt Lennartson, Dennis Arkeryd, and Martin Fabian at the Control and Automation Laboratory of Chalmers University of Technology [58, 57].

They use the following algorithm to compute a non-blocking and controllable synchronous composition of the system and the controller (from [57]):

$$\begin{array}{rcl} Q_0^x & = & \text{given} \\ k & = & 1 \\ 1: & & \\ Q_k' & = & \mathrm{BR}'(Q_m, Q_{k-1}^x) \\ Q_k'' & = & \mathrm{BR}_u(Q - Q_k') \\ Q_k^x & = & Q_{k-1}^x \cup Q_k'' \\ \mathbf{If}(Q_{k-1}^x & \neq & Q_k^x)\ \mathbf{then}\ k = k+1;\ \mathbf{goto}(1) \\ Q_{sup} & = & Q - Q_k^x \end{array}$$

Q is the set of states in the synchronous composition of the plant and the controller. Q_0^x is the union of the forbidden states (stated in the problem) and set of uncontrollable states in Q found during the composition (i.e., states from which the plant can generate an uncontrollable event that the controller does not allow). As usual Q_m denotes the marked states. $\mathrm{BR}(S, F)$ denotes the set of states that are backward reachable from S (including S itself) without passing through one of the forbidden states F. $\mathrm{BR}_u(S)$ denotes the set of states reachable from S only via uncontrollable events. In each iteration, the algorithm extends the set of forbidden states Q_k^x by finding all states within the current controller that are backward reachable by states outside the controller via uncontrollable events. The resulting supervisor is stored in Q_{sup}. A small example from [57] shown in Figure 1. The synchronous composition of the

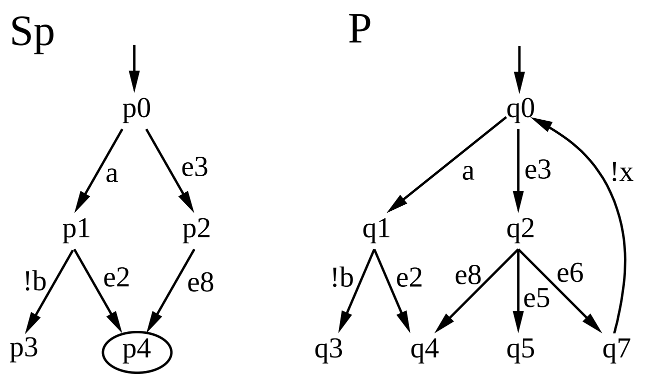

Figure 1: A specification *Sp* and plant *P*. The state p_4 is marked ($Q_m = \{p_4\}$). Events prefixed with an exclamation mark are uncontrollable. There are no forbidden states.

specification and the plant is shown in Figure 2(a). State q_7p_2 is forbidden since the uncontrollable event x is blocked by the supervisor, while it is enabled in the plant. Figure 2(b) shows the states Q_1' backward reachable from the marked state in the composition q_4p_4 without passing through the forbidden state q_7p_2. The state q_1p_1, however, is backward reachable via the uncontrollable event b from the state q_3p_1 outside the supervisor. Thus, the resulting supervisor shown in Figure 2(c) is stripped for the branch leading to q_3p_1.

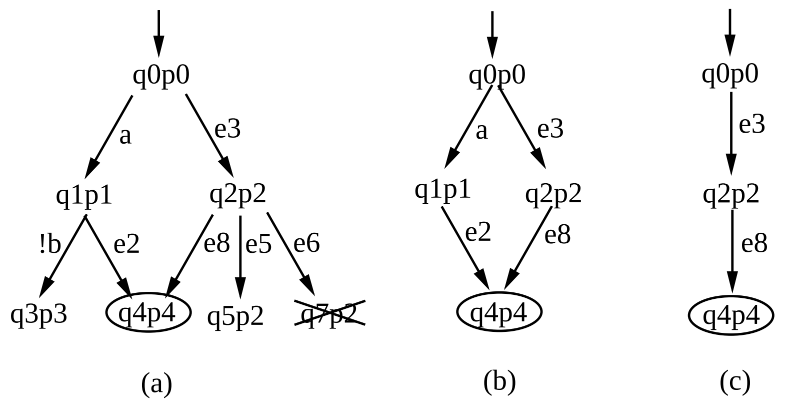

Figure 2: (a) The synchronous composition of the specification and the plant. (b) Q'_1. (c) The computed supervisor.

BDDs are used in [58, 57] to represent the transition relation $T(s, s')$ of the synchronous composition of the specification and the plant. Partitioning of the transition relation [10] and pruning of functional dependent variables [29] have been applied. In addition, a method called *smoothing* has been developed where only a subsystem of the transition relation is expanded in each iteration of a fixed point computation. In this way, each subsystem is "saturated" in turn such that the symmetry of the global system is preserved. This leads to smaller BDDs of frontier sets.

The approach has been applied to a class of resource allocation problems. These problems are often encountered in industrial automation systems where e.g. robots book physical locations before moving into them. The presented results are fairly similar to results obtained in planning. E.g., for a resource allocation version of the dinning philosophers the best result for 20 philosophers (monolithic transisiton relation) is 0.47 seconds on a 600 MHz Pentium III PC with a 850 MB RAM limit running Linux kernel 2.2, while 0.49 seconds were used for a planning version of the problem executed on a 500 MHz Pentium III PC with a 50 MB RAM limit running Linux kernel 2.2 [**?**].

4 Fault Tolerant Controller Synthesis

The INSPEC query "Fault Tolerant Control" gives 1300 records of which about 25 are related to fault tolerant controller synthesis and diagnosis for DES. We have used the references in these papers, the author search facility in IEEE Explore, and Google/CAMEO to find additional papers (We also went manually through the proceedings of the American Control Conference since 1996). As mentioned in the introduction, the primary goal has been to investigate if any previous work has considered symbolic synthesis of state feedback controllers with limited fault tolerance. This is apparently not the case, but the idea of limited fault tolerance (single fault tolerance) has been investigated to some extend.

4.1 Transition Based Fault Models

Most papers (e.g., [11, 13, 21]) regard *faults* as unexpected changes in a system that tends to degrade the overall system performance rather than causing a total breakdown. The term *failure* suggests a complete breakdown of a system component or function. It is essential [21] to differentiate between *temporary faults* and *permanent faults*, where permanent faults are caused by mal-functions that are triggered every time certain states are entered. It seems natural to model both temporary faults and permanent faults with a single class of fault transitions. Permanent faults would then be initiated by a fault transition changing a state variable that indicates permanent fault (e.g., *machine-1-broken*).

The usual way to model faults in the RW theory is to consider them uncontrollable events [2, 13].

The work in [44] specifies fault tolerance for mission critical systems. A *masking fault tolerant system* can recover from (mask) any fault. A *t-fault tolerant system* can mask up to t faults during its life time. A *gracefully degrading fault tolerant System* resumes operation during recovery, but less efficient. The paper models a system by an automaton with start states, but no marked states are considered. In addition, no algorithms or theory for controller synthesis are provided.

4.2 State Based Fault Models

Another approach is to separate the process state space into ranges of operation. In [36], three ranges are considered: *normal operation range*, *admissible error range*, and *non-admissible error range*. In this frame, the first and second range belongs to specified operation, whereas the non-admissible error range belongs to non-specified operation. In the non-admissible error range the plant will be damaged or produce damage to others. In [41, 40] the domain is divided into "good" and "bad" states.

In [41, 40] *Stability* is defined to visit the good states infinitely often. Thus, a controller is *stable* if it from any reachable bad state can force a trajectory that in a finite number of steps reaches the good states. *Stabilizability* is defined to choosing state feedback such that the closed loop system is stable.

A related approach [42] defines *Lyapunov stability* of a class of DES. Consider a set of states X_m that are *invariant* in the plant. That is, any execution starting in any state in X_m stays within X_m. X_m is *stable in the sense of Lyapunov* if for any $\epsilon > 0$ a max distance (given by some metric) $\delta > 0$ can be found such that any execution starting at a state within δ from X_m ends up in a state less than ϵ from X_m.

The two stability concepts of [41, 40] and [42] are shown to be to be exclusive of each other in [12] where a trivial example is given which is stable in the sense of error recovery but unstable in the sense of Lyapunov.

4.3 Symbolic Synthesis Approaches

In [2] a symbolic approach for synthesizing *non-blocking controllers* is described. Faults are modeled as uncontrollable events.

A symbolic approach for synthesizing *protective controllers* to prevent occurrence of forbidden states in chemical processes is presented in [36]. As described earlier, the state space is divided into three ranges: : *normal operation range*, *admissible error range*, and *non-admissible error range*. The purpose of the controller is to find a middle ground between the normal operation range and admissible error range such that no states in the non-admissible error range are visited. The plant model is an input/output extension of [46]. In particular, it is assumed that controllable events can be forced to happen (because they are considered inputs to the system). Synthesis algorithms are defined in Boolean Differential Calculus which, for more complex problems, could be implemented using BDDs and iterative squaring. An approach for this is, however, not presented.

In [21] a framework for analyzing and verifying fault tolerant behavior of state machines is presented. The state machine can be thought of as the closed loop of the plant and controller. Errors are modeled explicitly as separate entities (erroneous states) such that temporal and permanent faults can be modeled uniformly. Furthermore, the framework can describe fault-masking and t fault tolerance. It is based on a automaton where transitions are labeled with expected events. The automaton A is paired with a binary tolerance relation τ on the state space (A, τ). The tolerance relation defines acceptable unexpected state changes due to faults. An error of (A, τ) is a binary relation ϕ over the state space. Errors causes the events with similar "start state" to change behavior such that they lead to the errors "end state" instead of the end state given by the transition associated with the event. An error is *temporary* if it is applied only once, otherwise it is *permanent*. An error is *compatible*, if $\phi \subseteq \tau$. A fault masking automata has compatible and small errors (where *small errors* is given a reasonable meaning). (A, τ) is t fault tolerance if up to t faults can happen simultaneously in each transition in any execution from the initial state without the system leaving its tolerance bounds given by τ. An approach for stating the fault tolerance requirements in CTL is given, and it is suggested to use symbolic model checking to verify them. However, no empirical work in this direction is reported. The paper is also interesting due to its application of an algebra of binary relations.

4.4 Explicit Synthesis Approaches

Seong-Jin Park and Jong-Tae Lim from Department of Electrical Engineering at the Korea Advanced Institute of Science and Technology have contributed a large body of research on fault tolerant control of DES. After investigating stability concepts [12], they model faults as abnormal uncontrollable events in an ordinary RW DES model [15, 14, 13].

A frequently cited paper is [13], where they consider synthesizing supervisors for a *fault-tolerable* plant. A system is fault-tolerable if supervisor can be constructed that drives the state from the initial state to a marked state despite of any faults. Notice that the purpose of the controller is to reach the marked states. In [14], the work is extended to an uncertain plant model. In [15], a multiagent supervisory control approach for *antifault propagation* in serial production systems is studied. Basically, the problem is to find out in which cases an internal fault in one of the work cells can be fixed by subsequent work cells.

4.5 Diagnosis

Diagnosis of DES has mainly focused on analysing event sequences in order to determine if a fault has happened, and if so, which kind of fault [51, 52, 53, 55].

4.6 Planning

A *planning domain* is a tuple $(S, A, \rightarrow)$ where S is a finite set of states, A is finite set of actions, and $\rightarrow$: $S \times A \times S$ is a deterministic transition relation. A *planning problem* is a tuple (D, s_0, G) where D is a planning domain, s_0 an initial state, and G is a set of goal states. A *planning language* (e.g., STRIPS, ADL, $\mathcal{AR}$ [24, 43, 17]) is used to define planning domains and planning problems. In most planning languages an action is described by a precondition expression that must hold in the state, the action is applied, and a effect expression that describes how the action deterministically changes the state.

A *plan* is a sequence of actions leading from the initial state to a goal state. Since all actions are assumed to be deterministic and controllable, there is no explicit way to model non-determinism caused by faults. The focus in classical planning has been on developing efficient and optimal (in terms of the number of actions in the produced plan) algorithms for generating plans (e.g., by abstraction [23], learning search control rules [59], using search heuristics [6, 5], and symbolic representation [34, 17]).

Non-deterministic domains have only received minor attention in planning. The traditional approaches are *conditional planning* and *re-planning* [50]. In conditional planning, the truth-value of a set of state predicates is assumed to be unknown. They can become known during plan execution by applying sensing actions. The conditional plan branches out into two separate plans after a sensing action according to its two possible outcomes. Conditional planning can be used to synthesize fault tolerant controllers, where fault checks are carried out by sensing actions. A major problem, however, is the lack of efficient algorithms for synthesizing conditional plans. To date only explicit state algorithms have been developed.

As a response to the high complexity of conditional planning, re-planning and *reactive planning* have been considered [37, 25]. The re-planning approach observes plan execution. If the plan fails, a new plan is generated. Thus, it is an indirect way of handling non-determinism. The PRS reactive planner [26] mixes execution and planning. Reactive planning has been applied to fault recovery. The remote agent experiment on the Deep Space 1 probe was a fault tolerant reactive planner [60, 30, 35]. However, a general problem with all reactive planning approaches is that they are incomplete. In addition, for re-planning, response latency is a problem.

Universal planning [54], in its original form, is another indirect way of handling non-determinism. Given a deterministic domain model, a plan for each state is generated. During execution the state is observed to pick a plan to apply. Schoppers represented plans by decision trees. A more recent and interesting approach uses BDDs [9] and techniques from symbolic model checking [39] to generate and represent universal plans [17, 19, 20]. In this work, non-determinism is explicitly modeled. A universal plan U is a map from states to relevant actions to execute $U : S \rightarrow 2^A$. a universal plan corresponds to state feedback controller. For a non-deterministic domain D ($\rightarrow$ is now a non-deterministic transition relation) the closed-loop system of U and D behaves like D where each state s is pruned for transitions of actions not in $U(s)$. An execution is a possible state sequence produced by the closed-loop system starting in s_0. A universal planning domain summarizes the effect of uncontrollable events as non-determinism of controllable actions. Three classes of universal plans are introduced in [19, 20]: strong, strong cyclic, and weak. Their executions respectively satisfy the CTL formulas: AFG, AGEFG, and EFG.

The allowed behavior of a non-blocking supervisor is equal to a strong cyclic universal plan. However, a non-blocking supervisor is more general. Its control states are separated from the plant states. The control states of a universal plan on the other hand is the plant states.

Subsequent work has extended the domain model with uncontrollable actions [32] and considered adversarial environments [33]. Universal planning has further been extended to conformant planning [18] where the state is assumed to be unobservable, universal planning for CTL goals [45], and universal planning with partial observable states [4].

5 Conclusion

The impression from the literature is that the RW theory still is young. Wonham's group reports work mainly on the existence of non-blocking controllers in different classical situations (modular and hierarchical systems). Non-blocking controllers are not particularly interesting if the objective for the controller is to drive the plant into the marked states (as several subsequent application oriented papers assume). It is surprising that the DES literature apparently contains so little work on classifying the "strength" of controllers with respect to a class of reasonable control objectives. Aggressive work on reducing the computational complexity of controller synthesis of supervisory controllers seems to be limited to [58, 57], and this work does not go beyond standard techniques developed in symbolic model checking.

As far as we know, work on fault tolerant control within the RW framework has only been studied recently. No unified framework has emerged. Limited fault tolerance has been suggested [21, 44], but not in the form considered in n fault tolerant planning. The solution in [13] can be regarded as a ∞-fault tolerant plan. Planning has not specifically considered fault tolerance, except the DS1 remote agent experiment and related work centered around Brian Williams at MIT.

Acknowledgments

The former part of the report was mainly carried out at the IT University of Copenhagen in the fall 2003. The latter part was written in the fall 2002 at Carnegie Mellon University. Thanks to Bruce Krogh and Randal E. Bryant at Carnegie Mellon University and Henrik Reif Andersen at the IT University of Copenhagen for pointing out relevant references. This work was supported in part by the Danish Research Agency and the United States Air Force under Grants Nos F30602-00-2-0549 and F30602-98-2-0135. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force, or the US Government.

References

[1] E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. In *Hybrid Systems*, pages 1–20, 1994.

[2] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin. Supervisory control of a rapid thermal multiprocessor. *IEEE Trans. on Automatic Control*, 38(7), 1993.

[3] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W Yi. UPPAAL - A tool suite for automatic verification of real-time systems. In *Hybrid Systems*, pages 232–243, 1995.

[4] P. Bertoli, A. Cimatti, and M. Roveri. Conditional planning under partial observability as heuristic-symbolic search in belief space. In *Pre-Proceedings of the 6th European Conference on Planning (ECP-01)*, pages 379–384, 2001.

[5] A. Blum and M. L. Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1636–1642, 1995.

[6] B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Proceedings of the 5th European Conference on Planning (ECP-99)*, pages 360–372. Springer, 1999.

[7] K. Brace, R. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 40–45, 1990.

[8] E. Brinksma and A. Mader. Model checking in embedded system design. In *Proceedings of WODES-02*, pages 151–158, 2002.

[9] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 8:677–691, 1986.

[10] J.R. Burch, E.M. Clarke, and D.E. Long. Symbolic model checking with partitioned transition relations. In *International Conference on Very Large Scale Integration*, pages 49–58. North-Holland, 1991.

[11] J. Chen and R. J. Patton. *Robust Model-Based Fault Diagnosis for Dynamic Systems*. Kluwer Academic Publishers, 1999.

[12] K.-H. Cho and J.-T. Lim. A study on stability analysis of discrete event dynamic. *IEICE Trans. Inf. & Syst.*, E80-D(12):1149–1154, 1997.

[13] K.-H. Cho and J.-T. Lim. Synthesis of fault tolerant supervisor for automated manufacturing systems: A case study on photolithographic process. *IEEE Trans. on Robotics and Automation*, pages 348–351, 1998.

[14] K.-H. Cho and J.-T. Lim. Fault-tolerant robust supervisor for discrete event systems with model uncertainty and its applications to a workcell. *IEEE Trans. on Robotics and Automation*, 15(2):386–490, 1999.

[15] K.-H. Cho and J.-T. Lim. Multiagent supervisory control for antifault propagation in serial production system. *IEEE Trans. on Industrial Electronics*, 48(2):460–466, 2001.

[16] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *Proceeding of the International Conference on Computer-Aided Verification (CAV 2002)*, 2002.

[17] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via model checking: A decision procedure for $\mathcal{AR}$. In *Proceedings of the 4th European Conference on Planning (ECP'97)*, pages 130–142. Springer, 1997.

[18] A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.

[19] A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98)*, pages 875–881. AAAI Press, 1998.

[20] A. Cimatti, M. Roveri, and P. Traverso. Strong planning in non-deterministic domains via model checking. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning System (AIPS'98)*, pages 36–43. AAAI Press, 1998.

[21] M. D. Cin. Verifying fault-tolerant behavior of state machines. In *Proceedings of the Second IEEE High-Assurance Systems Engineering Workshop HASE 97*, pages 97–99, 1997.

[22] A. David, G. Behrmann, K. G. Larsen, and W. Yi. A tool architecture for the next generation of uppaal. Technical report, Uppsala University, 2003.

[23] K. Erol, J. Hendler, and D. S. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 1123–1128, 1994.

[24] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[25] E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI'92)*, pages 809–815. AAAI Press, 1992.

[26] M. Georgeff and A. L. Lansky. Procedural knowledge. *Proceedings of IEEE*, 74(10):1383–1398, 1986.

[27] G. Hoffmann and H. Wong-Toi. Symbolic synthesis of supervisory controllers. In *Proceedings of 1992 American Control Conference*, pages 2789–2793, 1992.

[28] G. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[29] A. J. Hu. *Techniques for Efficient Formal Verification Using Binary Decision Diagrams*. PhD thesis, Department of Computer Science of Stanford University, 1995.

[30] M. Ingham, R. Ragno, and B. C. Williams. A reactive model-based programming language for robotic space explores. In *Proceedings of ISAIRAS-01*, 2001.

[31] R. M. Jensen, R. E. Bryant, and M. M. Veloso. SetA*: An efficient BDD-based heuristic search algorithm. In *Proceedings of 18th National Conference on Artificial Intelligence (AAAI'02)*, pages 668–673, 2002.

[32] R. M. Jensen and M. M. Veloso. OBDD-based universal planning for synchronized agents in non-deterministic domains. *Journal of Artificial Intelligence Research*, 13:189–226, 2000.

[33] R. M. Jensen, M. M. Veloso, and M. Bowling. Optimistic and strong cyclic adversarial planning. In *Pre-proceedings of the 6th European Conference on Planning (ECP'01)*, pages 265–276, 2001.

[34] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI'96)*, volume 2, pages 1194–1201. AAAI Press, 1996.

[35] P. Kim, B. C. Williams, and M. Abramson. Executing model-based programs through graph-based temporal planning. In *Proceedings of IJCAI-01*, pages 487–493, 2001.

[36] E. Klein and H. Wehlan. Systematic design of a protective controller in process industries by means of the boolean differential calculus. In *Proceedings of WODES-96*, 1996.

[37] S. Koenig and R. G. Simmons. Real-time search in non-deterministic domains. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1660–1667. Morgan Kaufmann, 1995.

[38] F. Lin. *On Controllability and Observability of DES*. PhD thesis, Dept. Electl. Engrg., Univ. of Toronto, 1987.

[39] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publ., 1993.

[40] C. M. Özveren. Output stabilization of discrete-event dynamic systems. *IEEE Trans. on Automatic Control*, pages 925–935, 1991.

[41] C. M. Özveren and A. S. Willsky. Stability and stabilizability of discrete event dynamic systems. *Journal of ACM*, pages 730–752, 1991.

[42] K. M. Passino. Lyapunov stability of a class of discrete event systems. *IEEE Trans. on Automatic Control*, pages 269–279, 1994.

[43] E. P. D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the 1'st International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 324–332. Morgan Kaufmann, 1989.

[44] T. S. Perraju, S. P. Rana, and S. P. Sarkar. Specifying fault tolerance in mission critical systems. In *Proceedings of High-Assurance Systems Engineering Workshop, 1996*, pages 24–31. IEEE, 1997.

[45] M. Pistore, R. Bettin, and P. Traverso. Symbolic techniques for planning with extended goals in non-deterministic domains. In *Pre-Proceedings of the 6th European Conference on Planning (ECP-01)*, pages 253–264, 2001.

[46] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1):206–230, 1987.

[47] P.J. Ramadge. *Control and Supervision of DES*. PhD thesis, Dept. Electl. Engrg., Univ. of Toronto, 1983.

[48] M. Rausch and B. H. Krogh. Formal verification of PLC programs. In *Proceedings of the American Control Conference*, pages 234–238, 1998.

[49] K. Rudie. *Decentralized Control of DES*. PhD thesis, Dept. Electl. Engrg., Univ. of Toronto, 1992.

[50] S. Russell and P. Norvig. *Artificial Intelligence: a Modern Approach*. Prentice-Hall, 1995.

[51] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Trans. on Automatic Control*, 40(9):1555–1575, 1995.

[52] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Trans. on Control Systems Technology*, 4(2):105–123, 1996.

[53] M. Sampath, R. Sengupta, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete-event systems. *IEEE Trans. on Automatic Control*, 43(7):908–929, 1998.

[54] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 1039–1046. Morgan Kaufmann, 1987.

[55] R. Su. Decentralized fault diagnosis for discrete-event systems. Master's thesis, Dept. Electl. Engrg., Univ. of Toronto, 2001.

[56] E. Tronci. Automatic synthesis of controllers from formal specifications. In *Proceedings of the 2nd IEEE Conference on Formal Engineering Methods*, 1998.

[57] A. Vahidi. A study of symbolic tools in automatic control. Master's thesis, Control and Automation Laboratory of Chalmers University of Technology, 2002. Technical report 434L.

[58] A. Vahidi, B. Lennartson, D. Arkeryd, and M. Fabian. Efficient application of symbolic tools for resource booking problems. In *Proceedings of the American Control Conference*, 2001.

[59] M. Veloso, J. Carbonell, A. Pérez, D. Borrajo, E. Fink, and J. Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1):81–120, 1995.

[60] B. C. Williams and P. Nayak. A reactive planner for a model-based executive. In *Proceedings of IJCAI-97*, pages 1178–1185, 1997.

[61] K. C. Wong and W. M. Wonham. Hierarchical control of DES. *Discrete Event Dynamic Systems*, 6(3):241–273, 1996.

[62] W. M. Wonham. Notes on control of discrete-event systems. Technical report, Dept. of Electrical and Computer Engineering, University of Toronto, 1999. ECE 1636F/1637S.

[63] Zhonghua Zhang. Smart tct: An efficient algorithm for supervisory control design. Master's thesis, Dept. Electl. Engrg., Univ. of Toronto, 2001.

[64] H. Zhong. On control of DES: Decentralized and hierarchical control. Master's thesis, Dept. Electl. Engrg., Univ. of Toronto, 1987.