A Comparison of two Synthesis Methods for Timed Discrete-Event Systems*

M. Barbeau, F. Kabanza, and R. St-Denis
Département de mathématiques et d'informatique
Université de Sherbrooke
Sherbrooke, Québec CANADA J1K 2R1
Email: {barbeau, kabanza, stdenis}@dmi.usherb.ca

Abstract – **The control theory for discrete-event systems has become noteworthy due to its utility in automatically generating controllers. Recently two timed versions of this framework were developed to enable the specification of temporal properties and synthesis of controllers that supervise processes under time progression: one is operational, the other is axiomatic. The comparative study done in this paper shows their relative strengths and weaknesses.**

I. INTRODUCTION

In this paper, we compare two controller synthesis methods for *timed discrete-event systems* (TDES). These methods are in the line of the supervisory control theory introduced by Ramadge and Wonham [4]. Given an uncontrolled discrete-event system (hereafter called a *process*) and a control requirements specification, the goal in supervisory problems is to obtain a *nonblocking* and *maximally permissive* controller satisfying the requirements. Then, the process is controlled by combining its execution with that of the controller in a closed-loop as follows. At each step, the controller reacts to the same event as the process by determining those events that are prohibited at the next step among controllable ones. The framework of Ramadge and Wonham for deriving such a controller, by using synthesis algorithms, requires the following specifications: the description of unrestrained behavior of the process components; the identification of controllable events; and the expression of the control requirements as a set of legal event sequences. Generally, finite automata are used to specify the unrestrained and legal behavior of the process.

This framework has been specialized by Brandin and Wonham such that the process and control requirements specifications are both given in terms of TDES [3]. Hence, they are *operational*. This makes it possible to reason about time dependencies. Usually, the unrestrained behaviors of the different components of the process are specified as separate TDESs. The global TDES is then calculated as their *composition*. This approach is appealing because it allows manipulation of simpler (smaller) TDESs that are more readable. Similarly, the control requirements are specified using several TDESs (one per constraint). Then, the global control specification is obtained by taking their intersection. The operational character of the control specification hides the requirements in an executable structure. In real problems, however, computational complexity, readability, debugging and maintainability are limiting criteria which make the modeling, specification, and synthesizing tasks hard or even untractable with this approach [5]. In this case, an *axiomatic approach* can be a useful alternative.

Such an axiomatic approach has been recently developed by Barbeau, Kabanza, and St-Denis [2]. This method integrates concepts from control theory, concurrency, and AI planning. In this method, the control requirements are translated into a formal specification by using *Metric Temporal Logic* (MTL) formulas. These formulas are then checked over state sequences incrementally. During the synthesis process, a control-directed backtracking technique is used to prune the search space. Since MTL specifications are declarative and maintained as such during the synthesis process, we expect that debugging will become easier than with the operational approach. Moreover, since solutions are presented in a symbolic form, their traceability with regard to problem specifications is improved.

In this paper, we compare the two above methods. The goal of this comparative study is to determine their relative strengths and weaknesses. The comparison is made mainly along the convenience of specification languages by demonstrating how requirements stated in one formalism are expressed in the other.

II. DESCRIPTION OF THE PROBLEM

This section introduces a control problem on which our discussion and comparison is based. The process is specified by using an *activity transition graph* which is a graphical representation of a standard state machine. The control requirements are informally stated in natural language.

A. Process

The process consists of two components, numbered 1 and 2, that may be running in parallel. Every component is either idle, running or failed. Component i is started when event α_i occurs and then moves from state $IDLE_i$ to $RUNNING_i$. While running, a component may either stop (outgoing transition to state $IDLE_i$ labeled with event β_i) or fail (outgoing transition to state $FAILED_i$ labeled with event γ_i). Repair of a component is modeled as a transition from state $FAILED_i$ to state $IDLE_i$ labeled with event δ_i. Fig. 1 gives an activity transition graph which represents the global states and transitions of the process. The whole system is inactive until event α occurs

* The research described in this paper was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fonds pour la formation de chercheurs et l'aide à la recherche (FCAR).

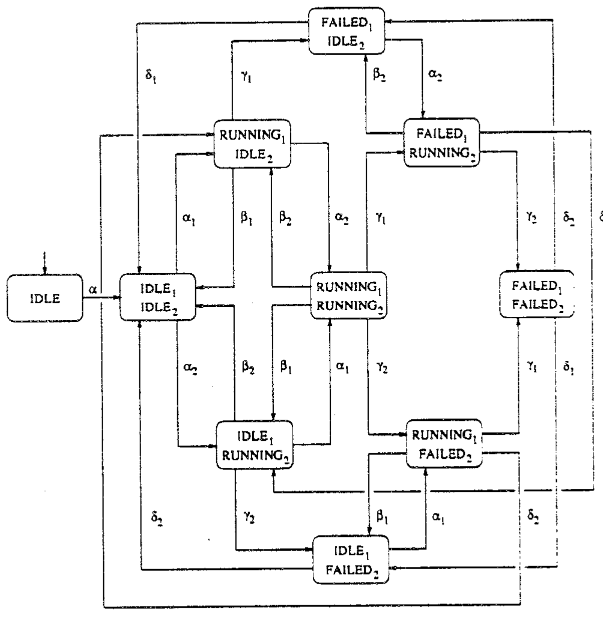

Fig. 1 Activity transition graph

from global state $IDLE$ to $\langle IDLE_1, IDLE_2\rangle$. For $i = 1, 2$, events α_i, β_i are controllable whereas γ_i and δ_i are uncontrollable. The occurrence of an event triggers an activity (or action) that has a duration. All activities last one time unit, except β_is which last two time units.

B. Control Requirements

There is a dependency from Component 1 to Component 2. The process startup must proceed as follows. Component 1 must be started first ($C1$). After a warmup period of 2 time units, Component 2 must be activated not later than 2 time units ($C2$). If Component 2 fails, then Component 1 may not be running more than 2 time units ($C3$). If Component 1 fails, Component 2 may remain running. For the sake of economy, however, Component 2 must be stopped within one time unit ($C4$).

III. THE OPERATIONAL APPROACH

In this approach, a *timed transition graph*, which is a graphical representation of a TDES, is used to formally specify the unrestrained behavior of the process. A TDES is like a finite state automaton in which there exists a special event, called *tick* and synchronized with a clock, that represents the progression of time. As usual, events are instantaneous, that is, an arbitrary number of events may occur between two *ticks*. The duration in time is associated to states. A timed transition graph is obtained from an activity transition graph and temporal properties associated to events. More specifically, each timed event σ has a *lower time bound* l_σ and an *upper time bound* u_σ. The set of events is partitionned into two subsets: the set of *prospective* events and the set of *remote* events. The events of the former have finite upper time bounds ($0 \leq l_\sigma \leq u_\sigma < \infty$), while the events of the latter have no upper time bounds ($0 \leq l_\sigma < u_\sigma = \infty$). An event may occur only within its time bounds relative to the times when it first becomes enabled.

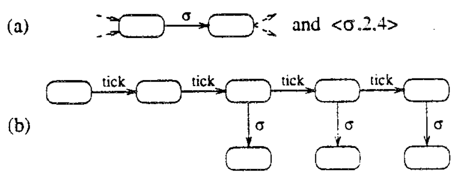

Fig. 2 Timed event

The concept of *timed event* has been introduced to study the controllability of TDESs and incorporate temporal attributes at the process level. Indeed, timed events are used to represent physical constraints that include delays (lower time bounds) and deadlines (upper time bounds). For example, the timed event $\langle\sigma, 2, 4\rangle$ is incorporated in a timed transition graph as shown in Fig. 2. The timed transition sub-graph of Fig. 2(b) actually corresponds to the transition labeled with σ in the activity transition graph of Fig. 2(a). Note that transitions labeled with *tick* must be inserted in such a way that all delay/deadline constraints are satisfied together. Furthermore, if we increase the delay and deadline for an event, the size of the corresponding timed transition sub-graph increases proportionately.

The role of timed events is limited because Brandin and Wonham have not defined any algebra for such events. For example, a timed event $\langle\sigma, -, -\rangle$ cannot capture temporal properties that depend not only on σ but also on events that occur before σ. For instance one cannot express directly the constraint that an event σ occurs after τ_1 with a deadline of two *ticks* or after τ_2 with a deadline of four *ticks*. This is indeed inconsistent with the definition of timed event, that is, each event must have only one upper bound. One can, however, express an equivalent constraint by using different event names to model different versions of event σ.

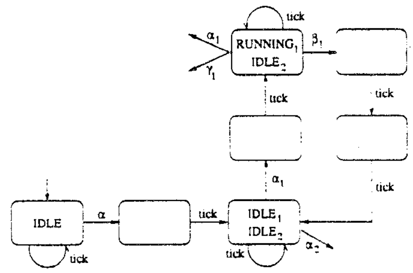

Fig. 3 Part of the timed transition graphs for the process

In this method only remote events can be controllable. Thus, it is impossible to introduce a deadline constraint on a controllable event at the process level. For the particular problem of Section II, this requires that the lower and upper bounds of all events are set to 0 and ∞ respectively even if α_2 has a deadline. The timed transition graph of the process is then obtained, firstly, by adding to each state of the activity transition graph of Fig. 1 a self-loop labeled with the event *tick*. Furthermore, durations must be considered since they represent physical properties of the process. Appropriate sequences of t *ticks* are thus added after each transition having an activity with

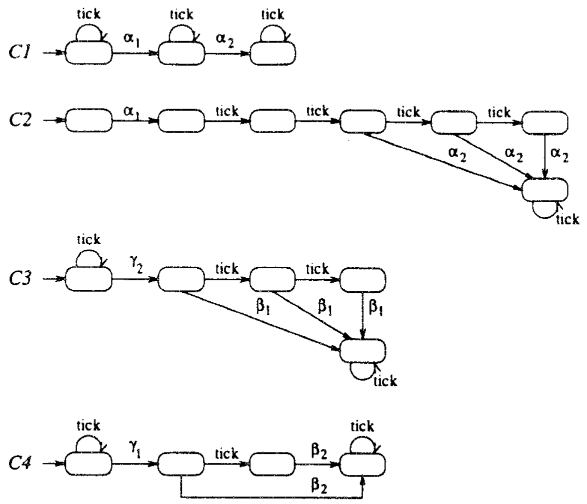

Fig. 4 Partiel timed transition graphs for the constraints

a duration t. Fig. 3 shows, in part, the resulting timed transition graph for the TDES G of the process. Note that for our problem, it is not true that an arbitrary number of events may occur between two *ticks*.

A. Modelling the Constraints

TDESs are used for modelling control requirements. Constraints $C1$ to $C4$ are formally described by the timed transition graphs of Fig. 4. Note that the timed transition graph for $C2$ includes also $C1$. Therefore, the first timed transition graph of Fig. 4 will not be further considered. One should note that these graphs are skeletons. Only self-loops labeled with *tick* have been added to these timed transition graphs. This first solution is intuitive. For example, to obtain a complete TDES for $C4$, we must reason on G by abstracting some details that are irrelevant for $C4$. The construction of such TDES represents a tedious task, particularly when the number of components or states of the process is large. Fig. 4 shows the complete solution for $C4$.

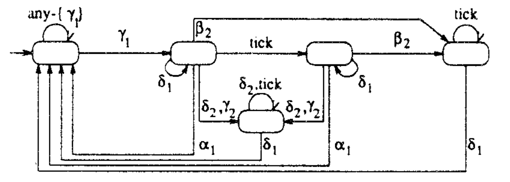

Fig. 5 Complete timed transition graph for $C4$

B. Deriving a Controller

The procedure used for computing a nonblocking and maximally permissive controller satisfying the requirements specification is algebraic and consists in the following three steps:

1. Combining the requirements specification languages by taking their intersection
$$C = \mathbf{meet}(\mathbf{meet}(C_2, C_3), C_4))$$
2. Deriving the automaton SC accepting the supremal controllable sublanguage included in $C \cap L_m(G)$
$$SC = \mathbf{supcon}(G, C)$$
3. Determining the feedback function
$$PHI = \mathbf{condat}(G, SC)$$

The operators **meet**, **supcon**, and **condat** are implemented in the TCT tool [6].[1] The operator **supcon** works on the product transition structure $\mathbf{meet}(G, C)$. It iteratively prunes states that do not comprise an outgoing transition on a possible but uncontrollable event in G.

IV. THE AXIOMATIC APPROACH

In the axiomatic approach, only the process is described by a TDES. The legal language is described by an MTL formula. TDESs, however, differ from those in the operational approach by the fact that transitions are labeled with timed activities rather than instantaneous events. That is, time progression is captured by transitions rather than states. In particular, a *tick* activity lasts at least one time unit. For instance, the TDES for the problem in Section II is described by a similar graph as in Fig. 1, except that each transition has the duration corresponding to that given in the problem description, and each state has a self-loop labeled with *tick*.

A. Modelling the Constraints

The requirements are expressed by the MTL formula $f_0 \equiv \Box_{\geq 0}(g_1 \wedge g_2 \wedge g_3 \wedge g_4)$, with g_i described as follows:

- g_1 expresses condition $C1$:
$$g_1 \equiv ((idle_1 \vee failed_1) \wedge idle_2) \rightarrow \Box_{\leq 1} \neg running_2$$
- g_2 expresses condition $C2$:
$$g_2 \equiv idle_1 \rightarrow \Box_{\leq 1}(running_1 \rightarrow (\Diamond_{\leq 4}(running_1 \rightarrow running_2) \wedge \Box_{\leq 2} \neg running_2))$$
- g_3 expresses constraint $C3$:
$$g_3 \equiv failed_2 \rightarrow \Diamond_{\leq 2} \neg(\neg running_2 \wedge running_1)$$
- g_4 expresses constraint $C4$:
$$g_4 \equiv failed_1 \rightarrow \Diamond_{\leq 1} \neg(\neg running_1 \wedge running_2)$$

B. Deriving a Controller

The manner in which a controller is synthesized is essentially *incremental model-checking* using a technique for *progressing formulas* along state sequences. That way, one prunes those transitions violating the specification, taking into account the controllability problem. To be more specific, using formula progression, one generates a *search graph* from which the controller is extracted. In this graph, states are identified with MTL formulas. Fig. 6 shows a *part* of such a graph for the aforementioned problem.

In this figure, w_i denotes a state of the process and f_i an MTL formula. For process states, we have $w_0 \equiv idle$, $w_1 \equiv idle_1 \wedge idle_2$, $w_2 \equiv running_1 \wedge idle_2$, $w_3 \equiv idle_1 \wedge running_2$, $w_4 \equiv running_1 \wedge running_2$, and $w_5 \equiv failed_1 \wedge idle_2$. The

[1] We presume that **supcon** handles the *forcible* events, which is not the case with the current version of TCT.

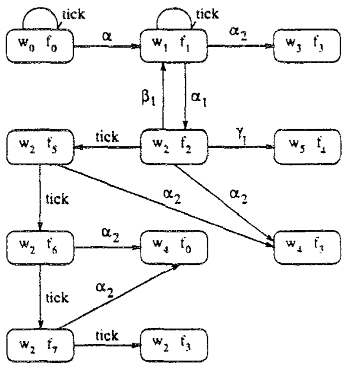

Fig. 6 Progression of the MTL specification through the process TDES

MTL formula f_0 is the input. The other formulas are obtained using the formula progression technique. Each of these formulas is a combination of sub-formulas from the original one, f_0, but possibly with different time subscripts. In each state, the formula expresses the requirements that must be satisfied by all the sequences rooted on it, according to the original formula f_0, and the history of paths from the initial state. Specifically:

$$\begin{aligned}
f_1 &\equiv f_0 \wedge \Box_{=0}(running_1 \rightarrow (\Diamond_{\leq 4}(running_1 \rightarrow running_2) \\
&\qquad \wedge \Box_{\leq 2} \neg running_2)) \wedge \Box_{=0} \neg running_2 \\
f_2 &\equiv f_0 \wedge \Diamond_{\leq 3}(running_1 \rightarrow running_2) \wedge \Box_{\leq 1} \neg running_2 \\
f_3 &\equiv false \\
f_4 &\equiv f_0 \wedge \Box_{=0} \neg running_2 \\
f_5 &\equiv f_0 \wedge \Diamond_{\leq 2}(running_1 \rightarrow running_2) \wedge \Box_{=0} \neg running_2 \\
f_6 &\equiv f_0 \wedge \Diamond_{\leq 1}(running_1 \rightarrow running_2) \\
f_7 &\equiv f_0 \wedge \Diamond_{=0}(running_1 \rightarrow running_2)
\end{aligned}$$

Note that Formula f_3 is equivalent to *false*. This means that states labeled with f_3 are *sinks*, so the controller must inhibit events leading to them.

MTL specifications have many advantages. For instance, as illustrated above, MTL formulas can be more natural than TDES. Their intuitive meaning is easily captured using the natural language interpretation for logical connectors (e.g., $\Diamond_{\leq t} f$ is read as "eventually f within t time units" and $f_1 \rightarrow f_2$ as "f_1 implies f_2"). We can express other styles of requirements more concisely and straightforwardly. For example, the requirement that each free-failure cycle on the $idle_1 \wedge ilde_2$ state lasts at most 20 time units is expressed by the formula

$$\Box_{\geq 0} \Diamond_{\leq 20}(\neg(failed_1 \vee failed_2) \rightarrow (idle_1 \wedge idle_2))$$

It should also be noted that one can change the deadline requirements by modifying only the subscripts of MTL modal operators. If one were using a TDES, such a change would lead to a more complex automaton. We are planning to handle even more succinct specifications using quantified MTL formulas.

Furthermore, the fact that states in the *search graph* are characterized by MTL formulas facilitates understanding of the impact of specification requirements on the operations of the process. For example, we can immediately understand the reason why a state is a sink by observing that an eventuality deadline is expired (e.g., its time script has decreased to zero before the eventuality is satisfied), or an always condition is made false. In the end, this helps debugging of specifications.

V. CONCLUSION

In this paper, two timed frameworks for the specification of temporal properties and synthesis of controllers have been discussed. These two frameworks have been compared with respect to three aspects, namely, the time model, specification formalism, and synthesis procedure.

In the time model of Brandin and Wonham time progresses in states, whereas in that of Barbeau, Kabanza, and St-Denis it progresses in transitions. Both methods use TDES to describe the process. In the method of Brandin and Wonham, the requirements are expressed operationally, that is, with automata, whereas in that of Barbeau et al. they are expressed declaratively, that is, with MTL formulas which we think are more readable than automata. The synthesis procedure used by Brandin and Wonham consists of a state space comparison between the TDES of the requirements and the TDES of the process (implemented in **supcon**). Barbeau et al. promote a procedure in which synthesis is performed as a state space search during which MTL formulas are checked incrementally. We think that this approach improves traceability because of its symbolic nature. For efficiency purposes, the state space is pruned using control-directed backtracking.

The two frameworks are thus substantially different. Notice that it could be possible to generate an automaton consisting of states satisfying MTL formulas [1]. One can then use the **supcon** operator to generate the controller. One can, however, reasonably expect that the formula progression technique performs better on average because sink states are avoided by incremental model-checking.

VI. REFERENCES

[1] R. Alur and T. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993.

[2] M. Barbeau, F. Kabanza, and R. St-Denis. Synthesizing plant controllers using real-time goals. To appear in the Proceedings of the International Joint Conference on Artificial Intelligence, 1995.

[3] B. A. Brandin and W. M. Wonham. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 39(2):329–342, 1994.

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

[5] K. Rudie, N. Shimkin, and S. D. O'Young. Timed discrete-event systems: A manufacturing application. In *Proceedings of the 1994 Conference on Information Science and Systems*, pages 374–381, Princeton, 1994.

[6] W. M. Wonham. Notes on control of discrete-event systems. Technical report, University of Toronto, 1994. 255 pages.