Modeling Decentralized Real-Time Control by State Space Partition of Timed Automata

Thanikesavan Sivanthi, Srivas Chennu*, Lothar Kreft
Department of Communication Networks
Hamburg University of Technology
Schwarzenberg Strasse 95, BA 4d, D - 21073, Germany
{thanikesavan.sivanthi, srivas.chennu, kreft}@tu-harburg.de

Abstract

Timed automata provide useful state machine based representations for the validation and verification of real-time control systems. This paper introduces an algorithmic methodology to translate the state space visualization of a centralized real-time control system to a decentralized one. Given a set of timed automata representing a centralized real-time control system, the algorithm partitions them into a collection of interacting submachines. Importantly, this methodology allows for model-checking of the derived decentralized system against the same set of verifications as that specified for the centralized system. The complexity analysis of the algorithm is presented as a function of the number of tasks and nodes comprising the decentralized system.

1. Introduction

Current trends in the development of microprocessor hardware, a need for lesser maintenance overheads, better scalability, and robustness are driving modern real-time control systems from centralized to decentralized architectures. Such a decentralized architecture consists of a set of loosely coupled nodes, which communicate with each other by means of a broadcast bus. Different approaches for decentralization of a centralized real-time control system have been discussed in literature [4, 6, 8, 10]. An important requirement on the process of decentralization is that the decentralized system should satisfy the same verification criteria as the centralized system. This ensures that the decentralized system provides the same functionality from the designer's perspective. The correctness of the system can be verified by specifying the *reachability*, *safety* and *liveness* properties [2] that should always hold if the system has to meet its specifications.

Timed automata provide a useful state machine based representation for the validation and verification of a real-time system. The verification tools like UPPAAL [2, 3] accept such timed automata as input and perform model-checking based on a given set of verifiable properties specified by the real-time system requirements. In this paper, we detail an algorithmic methodology to perform such a *verifiably correct* decentralization of a centralized real-time control system. The centralized real-time control system, consisting of interdependent tasks, is modeled using timed automata and the decentralization is achieved by distributing the states of the timed automata over the processing nodes comprising the decentralized system. To our knowledge, this methodology is novel in its presentation of a consistent procedure for moving from centralized to decentralized real-time control and we expect that it will serve as a component in the process of modeling large-scale distributed real-time systems.

The rest of the paper is organized as follows. Section 2 describes the representation of a centralized real-time control system as a collection of timed automata. Section 3 describes the state space partition model of a decentralized real-time control system. Section 4 derives complexity measures for the algorithm. The paper concludes with Section 5.

2. Timed automata model of a centralized real-time control system

A centralized real-time control system consists of a set of interdependent tasks that execute concurrently, under deadline constraints, managed by a central scheduler. Each task consists of a collection of modules, which execute in a particular sequence and represent the elementary computing functions performed within the task. The flow of

*Srivas Chennu is a graduate student at the Hamburg University of Technology.

Proceedings of the 2005 Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'05)
0-7695-2462-1/05 $20.00 © 2005 IEEE

control in such a system can be suitably represented by a task dependency graph, which is a directed acyclic graph $G = (V, E)$. The vertices V of the graph represent the modules $\{M_k : k \in 1..N\}$ comprising the tasks, where N is the total number of modules. The edges E represent execution dependencies between these modules. Figure 1, depicts a task dependency graph for two tasks $\{T_i : i = 1, 2\}$ each consisting of a set of modules. The task T_1 is comprised of five modules $\{M_1, M_2, M_3, M_4, M_5\}$ and the task T_2 is comprised of three modules $\{M_6, M_7, M_8\}$. For every task T_i, there is an associated deadline d_i that specifies the time starting from the release time of the task within which all the modules in T_i must complete their executions. The edges of the graph represent the following possible types of dependencies between the modules:

Order dependencies are identified by solid edges, and indicate that the execution of a module depends on the completion of, or on the data from, one or more modules. In Figure 1, the module M_2 is order dependent on the module M_1.

Temporal dependencies are identified by dashed edges, and indicate that a module must finish its execution within a certain time from the completion of another module. In Figure 1, the modules M_6 and M_8 have a temporal dependency $\Delta = 4$, specifying that M_8 must complete within four time units of the completion of M_6.

Control dependencies are identified by dotted edges that denote mutually exclusive execution paths. The modules in an execution path can depend on other modules, except those in another mutually exclusive execution path. During runtime one of these execution paths is chosen, taking care of the dependencies along that path. In Figure 1, after M_2 completes its execution either the path with the module M_3 or the path with the module M_4 is chosen. The module M_5 is triggered after the completion of M_3 or M_4, depending on the execution path chosen during runtime.

The whole execution system represented above, consisting of the modules and the associated control flow, can be conveniently modeled by a set of *timed automata*. The theoretical basis for such automata has been previously developed [1] and extended [3] as a suitable means of modeling real-time systems. We base our construction and semantics of timed automata on that defined and used in UPPAAL [2], in which a timed automaton is a 6-tuple $\langle L, L^0, C, A, E, I \rangle$ where

- L is a set of locations,
- $L^0 \in L$ is the initial location,

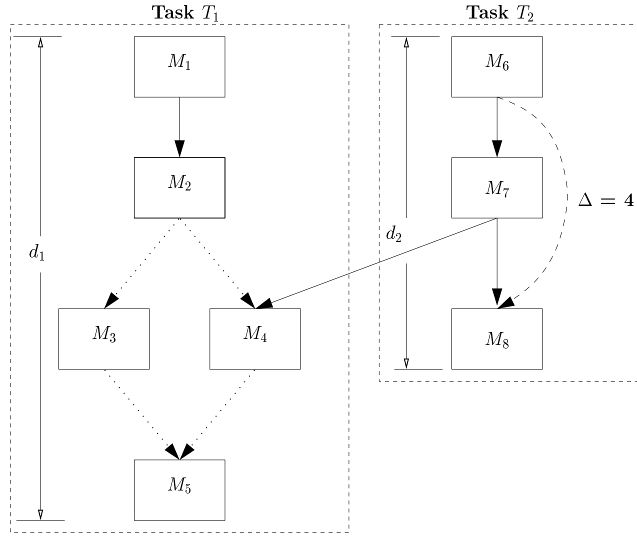

Figure 1. A simple task dependency graph

- C is a set of clocks,
- A is a set of actions, and
- $E \subseteq L \times A \times B(C) \times 2^C \times L$ is a set of transitions. An edge $\langle l, a, \delta, \lambda, l' \rangle$ in E represents a transition from a location l to a location l' with an action a. The set $\lambda \subseteq C$ is the clocks to be reset with this transition, and δ is a clock constraint.
- $I : L \rightarrow B(C)$ is a function that maps the locations to the clock invariants.

Based on this definition, we construct a timed automaton (TA) denoted by $\mathcal{A}_i$, for each task T_i in a real-time system by introducing the following abstractions:

- In a centralized real-time control system, a scheduler allocates resources and schedules the modules at times based on a scheduling mechanism. In our model, we assume a timed automaton representation of the scheduler is available and the automaton generates $begin!$ and $end!$ synchronization actions [2] that are sent to the timed automata representing all tasks.
- A module M_k in the task T_i corresponds to a location L_k in $\mathcal{A}_i$. Further, when module M_k is executing $\mathcal{A}_i$ is at L_k.
- $\mathcal{A}_i$ is initially at the location L_i^0, which is defined to be the idle location. Further, whenever no module from T_i is executing, $\mathcal{A}_i$ returns to L_i^0.
- When module M_k begins its execution, $\mathcal{A}_i$ receives the $begin?$ synchronization action from the scheduler,

which also sets the variable $modid$ to the value k. This variable ensures that only the transition to location L_k is enabled. On reception of this action, $\mathcal{A}_i$ moves from L_i^0 to L_k, if for every module $M_{k'}$ that M_k depends on, the variable $I_{k'}$ has been set. The variable $I_{k'}$ is set by the scheduler when the module $M_{k'}$ finishes its execution.

- When the module M_k finishes its execution, $\mathcal{A}_i$ receives the $end?$ synchronization action from the scheduler, which also sets the variable $modid$ to the value k. On reception of this action, $\mathcal{A}_i$ moves from L_k back to L_i^0, and sets the variable I_k to indicate the completion of M_k.

- In order to time the execution of the complete task, a clock c_i is associated with each T_i. When the first module in T_i begins its execution, c_i is reset. When the last module in T_i completes, c_i is checked against the deadline d_i.

- For the next run of $\mathcal{A}_i$, all the associated I_k variables are reset to their initial values.

Figure 2 and Figure 3, show the TAs constructed using the above abstractions for the tasks T_1 and T_2 in Figure 1 respectively.

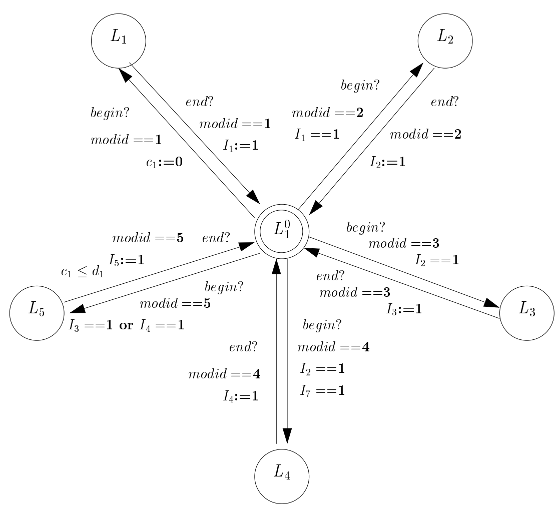

Figure 2. Timed automaton $\mathcal{A}_1$ for Task 1

To further elucidate the model described above, we consider the TA $\mathcal{A}_2$ for Task T_2, in Figure 3. The automaton has three locations L_6, L_7 and L_8 representing the modules M_6, M_7 and M_8 respectively. Initially the automaton is in the idle state L_2^0. When the $begin?$ synchronization action indicating the start of execution of M_6 is received, the automaton moves from L_2^0 to L_6 after resetting the clock timer

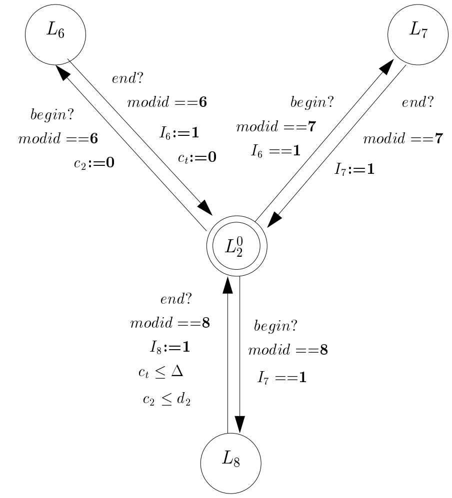

Figure 3. Timed automaton $\mathcal{A}_2$ for Task 2

c_2. On the completion of M_6 the $end?$ synchronization action is received, which triggers the automaton to move back to L_2^0 after setting I_6 and resetting the clock c_t to track the fulfilment of the temporal dependency Δ. Similarly, the execution of modules M_7 and M_8 result in transitions to states L_7 and L_8, respectively. During the transition from L_8 to L_2^0, both clocks c_2 and c_t are verified to be within their corresponding limits.

The above timed automata model is capable of modeling all the three execution dependencies between the tasks of a real-time control system. The order dependencies are modeled by setting and checking for the variables I_k. The control dependencies are modeled by the fact that, based on the $modid$ variable set by the scheduler one of the mutually exclusive execution paths is chosen for execution. Finally, the temporal dependencies are modeled using the mechanism of resetting clocks and associating appropriate clock constraints with the transitions.

3. State space partition model of a decentralized real-time control system

The global state of a centralized real-time system developed so far changes when at least one of the tasks changes its state. Thus the global state of the system can be viewed as a composition of the states of the individual tasks. Consequently, the problem of decentralizing this state over a collection of interconnected processing nodes can be handled on an individual task basis, adhering to the following sequence of steps:

1. The set of concurrent tasks composing the real-time system is identified.

0-7695-2462-1/05 $20.00 © 2005 IEEE

2. The timed automaton for every identified task is derived, as detailed in Section 2.

3. An allocation plan, which maps each module of each task to a node is obtained. This can be viewed as a combinatorial problem that involves deriving a suitable scheduling and allocation of the modules to the processing nodes, while at the same time satisfying the real-time deadline constraints. The work in [8] treats this problem as one of linear optimization, and given a task dependency graph similar to that in Figure 1, generates an optimal allocation plan and a schedule for each node, taking into consideration the completion deadlines, dependencies between modules, and allowed system utilization limits. Such a plan can also be determined by alternate methods [4, 6, 10].

4. Based on the assignment in Step 3, the TAs of the tasks are partitioned into several interacting submachines at each processing node. These timed submachines model the original system functionality by communicating with each other using broadcast messages on the bus.

5. The system bus is modeled by a timed automaton. The automaton models a queued messaging interconnect, introducing either fixed (e.g. synchronous TDMA) or variable transmission delays (e.g. CAN) [5] for the delivery of messages between the nodes.

In our model, the bus automaton $\mathcal{B}$ has the following locations:

1. A $free$ location indicating an idle bus, and an empty message queue.

2. A $busy$ location indicating an occupied bus, and one or more messages in the queue.

$\mathcal{B}$ transitions from $free$ to $busy$ on receiving the $enq?$ synchronization action, and adds the message to be transmitted to the queue. Whenever at the $busy$ location, after a delay time determined by the type of the bus being modeled, $\mathcal{B}$ removes a message from the queue depending on the scheduling mechanism of the bus and delivers it. If the queue is empty, it moves back to the $free$ location. Else, it delivers the next message to be scheduled on the bus.

State machine decomposition is an approach used in the field of digital design to minimize the area, delay, or power required, in sequential logic circuit design [7, 9]. We now detail an algorithmic approach to efficiently perform such a state machine decomposition for a given set of TAs provided as an input. The logic of the algorithm is explained below.

1. The primary inputs to the algorithm are the TAs representing the tasks constituting the centralized real-time control system and an allocation plan. This plan is assumed to be derived as described above in Step 3 of the decentralization procedure.

2. Let $\{M_i\}$ be the set of modules, and $\{N_j\}$ the set of nodes. Each node N_j has a scheduler, which generates $begin_j!$ and $end_j!$ synchronization actions, that are sent to all modules assigned to N_j. Since each node scheduler works in parallel with the other node schedulers, the $modid$ variable cannot be global to all the schedulers. Hence, the scheduler of each node N_j has its own variable $modid_j$ to control the submachines running at that node.

3. The algorithm populates the submachines $\{M_{ij}\}$, where M_{ij} represents a portion of the task T_i executing on the node N_j. Each M_{ij} is initialized with an idle location L_{ij}^0.

4. The algorithm analyzes the TAs given, and for each location L_k in a TA $\mathcal{A}_i$ assigned to N_j it creates a corresponding location L_k in the submachine M_{ij}.

5. It then examines each transition in $\mathcal{A}_i$, and if it finds an incoming transition from L_i^0 to L_k it creates a corresponding transition from L_{ij}^0 to L_k in M_{ij}, with the following expressions [2]:

   (a) The $begin?$ synchronization action from the original transition in $\mathcal{A}_i$ is copied as $begin_j?$. This ensures the transition to location L_k is triggered only by the scheduler of node N_j.

   (b) In the check for the $modid$ variable the variable name is changed to $modid_j$.

   (c) Clock reset and clock constraint expressions are copied as is.

   (d) For each variable $I_{k'}$ checked for by the transition in $\mathcal{A}_i$, which is set by a transition from a location $L_{k'}$ assigned to a node $N_{j'}(j \neq j')$, $I_{k'}$ is replaced with a corresponding variable $Y_{k'}$. Otherwise, $I_{k'}$ is copied as is.

6. If the transition in $\mathcal{A}_i$ is an outgoing transition from L_k to L_i^0, and the variable I_k set by this transition is checked for by another transition to a location $L_{k'}$ assigned to a node $N_{j'}(j \neq j')$, a committed location [2] and two transitions are created in M_{ij}, as explained below:

   (a) A transition from L_k to a committed location is created, with the following expressions:

i. The $end?$ synchronization action from the original transition in $\mathcal{A}_i$ is copied as $end_j?$, this ensures the transition to location L_k is triggered only by the scheduler of node N_j.

ii. In the check for the $modid$ variable, the variable name is changed to $modid_j$.

iii. Clock reset and clock constraint expressions are copied as is.

iv. The assignment to set the variable I_k is copied as is.

(b) A transition from the committed location to L_{ij}^0 is created. A synchronization action $enq!$ is added to the transition, and the variable $msgid$ is set to the value k. This action is sent to the bus automaton $\mathcal{B}$. The bus automaton reacts to the $enq?$ synchronization action by adding the message to be delivered to its queue, and eventually setting the corresponding variable Y_k after a delay determined by the type of the bus being modeled.

7. On the other hand, if the transition in $\mathcal{A}_i$ is an outgoing transition from L_k to L_i^0 and the variable I_k set by this transition is not checked for by any other transition, a transition from L_k to the initial location L_{ij}^0 is created with the following expressions:

   (a) The $end?$ synchronization action from the original transition in $\mathcal{A}_i$ is copied as $end_j?$.

   (b) In the check for the $modid$ variable, the variable name is changed to $modid_j$.

   (c) Clock reset and clock constraint expressions are copied as is.

   (d) The assignment to set the variable I_k is copied as is.

It is important to note that the algorithm assumes that the clocks used in the decentralized system are global [2] to all automata and that the read and write operations to these clocks can be performed in a synchronous, consistent manner.

The state partition algorithm detailed thus is now applied to partition the TAs in Figures 2 and 3, from our example in Figure 1. Let us assume that the decentralized system consists of two processing nodes N_1 and N_2, which are connected by a FIFO broadcast bus. We provide the following inputs to the algorithm:

1. The TAs representing the tasks T_1 and T_2.

2. An allocation plan which maps the modules M_1, M_2, M_3, M_6 and M_7 to the node N_1, and the modules M_4, M_5 and M_8 to the node N_2.

The algorithm generates four partitioned submachines M_{11}, M_{12}, M_{21} and M_{22} as shown in Figures 5, 6, 7, and 8 respectively.

The FIFO bus is modeled by the timed automaton shown in Figure 4. The automaton maintains a circular buffer of length len, with two pointers $front$ and $back$ which point to the first and last message in the $queue$ respectively. These pointers are updated whenever a message is added or removed from the $queue$, taking care of the boundary conditions of an empty or a full $queue$. The clock c_d tracks the fulfilment of the message transmission delay t_{xd}.

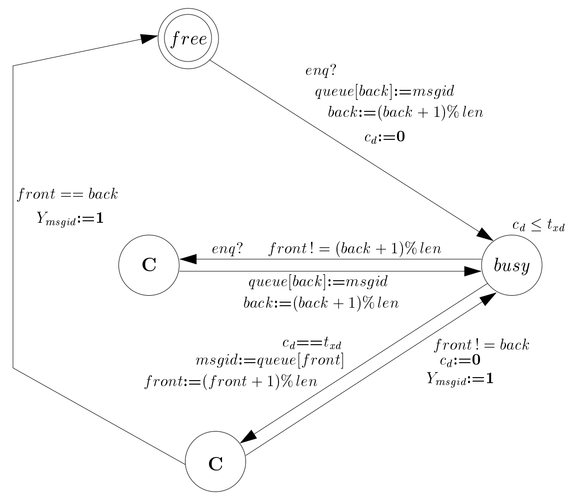

Figure 4. Timed automaton for a simple FIFO broadcast bus

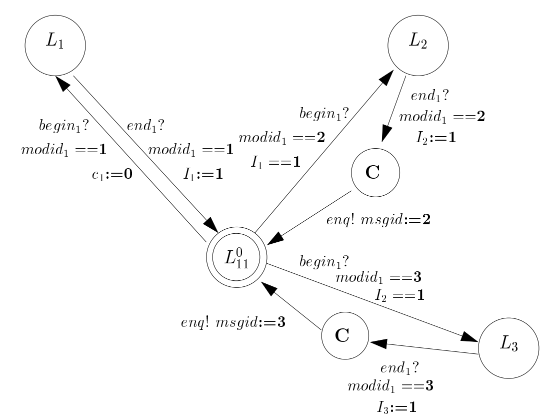

Figure 5. Submachine M_{11}

The submachines M_{11}, M_{12}, M_{21} and M_{22}, in combination with the bus automaton realize an equivalent decentralized model of the centralized real-time control system shown in Figure 1. The functional equivalence of the automata representing the centralized system and the subma-

0-7695-2462-1/05 $20.00 © 2005 **IEEE**

chines in combination with the bus automaton representing the decentralized system can be confirmed in UPPAAL, by verifying that the relevant reachability, safety and liveness properties identified for the centralized case continue to hold for the decentralized case. Thus the decentralized model provides a verfiably correct decentralization of the given centralized system.

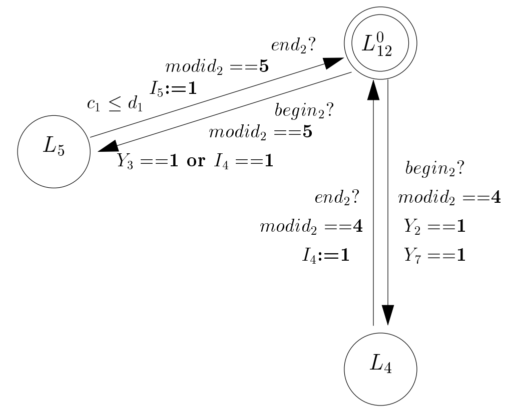

Figure 6. Submachine M_{12}

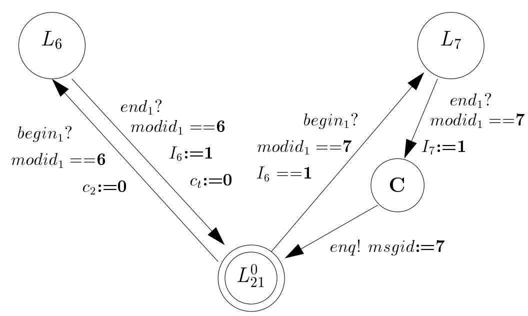

Figure 7. Submachine M_{21}

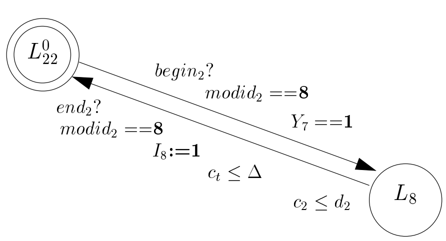

Figure 8. Submachine M_{22}

4. Complexity analysis

We now present a brief analysis of the time and space complexity of the state partition algorithm presented in Section 3. This analysis provides an important insight into the processing requirements of the algorithm.

In order to calculate the complexity measures of the algorithm, we first define the following execution parameters:

- n is the number of tasks in the real-time system being modeled.
- m is the number of processing nodes.
- p is the maximum number of elementary modules in a task.
- k is the maximum number of elementary modules of a task assigned to a node.
- l is the maximum number of variables of the type $I_{k'}$ associated with the incoming transition to a location.

4.1. Time Complexity

An expression for the time complexity of the algorithm is derived from the following observations:

- Each location in a TA representing each task is processed once. This processing has a complexity of $O(n \cdot p)$. For each such location, the incoming and outgoing transitions associated with it are examined.
  - Each variable checked for by the incoming transition is examined. The complexity of this processing is approximately $O(l)$.
  - For each variable set by the outgoing transition from the location, the algorithm verifies whether it is checked for by an incoming transition to a location on a different node. The complexity of this processing is $O(n \cdot p \cdot l)$.

Combining the above, we conclude that the time complexity of the algorithm is $O(n \cdot p \cdot l + n^2 \cdot p^2 \cdot l)$.

4.2. Space complexity

The approximate space complexity of the state partition algorithm, calculated by a reasoning similar to that for ascertaining time complexity, is $O(n \cdot p \cdot l + m \cdot n \cdot k \cdot l)$, and stems from the following facts:

- The storage requirement for the TAs grows as $O(n \cdot p \cdot l)$.

Proceedings of the 2005 Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'05)
0-7695-2462-1/05 $20.00 © 2005 IEEE

- The storage requirement for the submachines grows as $O(m \cdot n \cdot k \cdot l)$.

Combining the two measures, we arrive at the space complexity measure given above.

In practice, an efficient implementation of the algorithm can mitigate the above complexity estimates, by means of well-designed data structures that support efficient search and update operations. Further, in case of a real-time control system consisting of a large number of concurrent tasks, both time and space complexity of the model can be significantly reduced by considering separately the operation of the system in different *functional modes*, where a mode is a clearly distinguishable operational phase of the system. This is because not all tasks in the system are required in all of these functional modes, i.e. certain tasks may be exclusive to one mode of operation. This approach would provide a suitable means for managing the potential state space explosion problem that could occur when modeling large real-time control systems.

5. Conclusion

In this paper we have discussed a state space partition methodology for verifying the decentralization of a centralized real-time control system. The timed automata modeling a centralized system are partitioned into a collection of interacting submachines running on the different nodes of a decentralized system. Further, the process of decentralization also models an automaton for the bus and the message transmission delays therein. The distributed submachines, along with the bus automaton, provide a verfiably correct decentralization of the given centralized system. When integrated into an overall system design, the state partition algorithm would serve as a base for better visualization of the dynamics of the decentralized system, and provide for validation and verification of the decentralized real-time architecture, and its reaction to failures, both of the processing nodes, and of the bus.

References

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems: Real Time*, pages 200–236. Springer–Verlag, September 2004.

[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] H. Gomaa. A software design method for distributed real-time applications. *J. Syst. Softw.*, 9(2):81–94, 1989.

[5] J. Krakora and Z. Hanzalek. Timed automata approach to CAN verification. In *IFAC Symposium on Information Control Problems in Manufacturing*, 2004.

[6] D.-T. Peng, K. G. Shin, and T. F. Abdelzaher. Assignment and scheduling communicating periodic tasks in distributed real-time systems. *Software Engineering*, 23(12):745–758, 1997.

[7] R. Shelar, M. Desai, and H. Narayanan. Decomposition of finite state machines for area, delay minimization. In *IEEE ICCD*, pages 620–625, October 1999.

[8] T. Sivanthi and U. Killat. Static scheduling of periodic tasks in a decentralized real-time control system using an ILP. In *IEEE ICPADS*, July 2005.

[9] G. Sutter, E. Todorovich, S. Lopez-Buedo, and E. I. Boemo. FSM decomposition for low power in FPGA. In *12th International Conference on Field Programmable Logic and Application*, pages 350–359. Springer-Verlag, 2002.

[10] M. Trngren and J. Wikander. A decentralization methodology for real-time control applications. *J. of Control Engineering Practice*, 4(2), 1996.