Timed Simulation with Multiple Resource Schedulability Analysis

Yang Zhao, Slobodan Matic
EECS 290N Project Report
University of California at Berkeley
ellen_zh, matic@eecs.berkeley.edu

Abstract

Target applications we consider in this project are model-based computer controlled systems, for which the required system response is deterministic both in value and time. Specification of real-time software components for multiple resources and multi-domain timed simulation is discussed. We describe a simulation framework in the Ptolemy modeling environment that takes into account timing requirements of the model software components and allows for schedulability analysis. In this framework both a static and dynamic scheduling mechanism is proposed.

1 Introduction

We are motivated by the current gap between algorithm development (functional level) and tuning of stringent timing constraints (architecture level). The first part of the design is supported by widely accepted modeling and design environments such as MATLAB/Simulink, MATRIXx/SystemBuild or Ascet-SD. The correctness of the model verified at the functional level is preserved by tools for code generation such as Real-TimeWorkshop Embedded Coder or dSpace TargetLink. However, no analysis of timing constraints, let alone worst case execution analysis of the generated code is performed. Very often the outcome of the process are design errors that show up only after deployment. The most common error is that sophisticated control algorithms cannot be implemented with acceptable performance (e.g. sampling time) on the selected platforms. Therefore, the separation of the design environments for functionality and for time, together with the limited support provided by the existing tools make the process quite inefficient. Moreover, in most tools the automatic code generation scheme works only for models whose components have periodic (possibly multirate) activation patterns.

To some extent platform dependent properties must be considered in the design phase. Through appropriate design annotations the part of a model that describes real-time software components could specify: 1. component timing properties (triggering pattern, execution time, deadline), 2. concurrent interaction of components (precedence constraints graph), 3. resource allocation (component to host allocation) and 4. scheduling mechanism. We briefly discuss each of these components with respect to the framework proposed in the project.

1. We build our programming model upon the Ptolemy timed multitasking (TM) event-triggered model [1], that controls timing properties through a deadline of an actor (a task in this model) which specifies the time when the actor computational results are produced to the physical world or to other actors. Unlike the highly periodic Giotto models [2], a task in TM is executed when there is an input event (trigger) that fulfills certain conditions specified by the actor. Since any deployment introduces noneglibile processing delays a model should state task execution (computation) times. Precise estimation of these is a challenging problem and we do not consider it here.

2. The software model also describes the tasks and their relations. An actor assumes a simple task without internal synchronization and blocking. Events that occur between tasks hence represent either data passing or execution precedences. We think that a particularly useful abstraction is a *composite* TM actor specified with a directed acyclic precedence graph of *atomic* TM actors. Only composite actors have trigger conditions and deadline, whereas each contained atomic actor specifies its execution time and computational resource it is allocated to. The entire model consists of a network of composite TM and other actors (say, from the continuous time CT domain).

3. Resource utilization must also be exposed in the model since the tasks compete at least for CPU time. In fact, the emphasis of the project is on the simulation of the multiple resource models, i.e. on multi-resource scheduling strategies.

4. Usually, real-time operating systems provide static

priority-based scheduling mechanism. It is also common both to perform the schedulability analysis and to generate the schedule off-line (static table-based scheduling) and only to dispatch tasks on-line. This, of course, requires fixed and known actor activation pattern. Finally, both scheduling and dispatching can be performed on-line (dynamic planning-based). In our programming environment we allow annotation of a simulation model with a scheduling mechanism and in the next two sections we further describe possible solutions for the two scheduling mechanisms mentioned last.

2 Periodic TM Model

In this section we assume that activation pattern of composite TM tasks is periodic and we offer a static table-based scheduling solution. Formally, let C be the set of composite TM actors. Each composite actor $c \in C$ is specified with its period of invocation $p(c)$, and a directed acyclic precedence graph of simple task actors having the same period. We first assume that deadline of a composite task is equal to its period, but we later discuss the case when it is not. Let $A(c)$ be the set of all atomic actors defined in the composite actor $c \in C$, $A = \bigcup_{c \in C} A(c)$ be the set of all atomic actors and let R be the set of all computation resources that an actor can use. We assume that the tasks are nonpremptible and that task to host (resource) mapping has been already determined. Each task actor $a \in A$ is specified with execution time $t(a) \in Reals$ and is uniquely assigned to a resource $r(a) \in R$.

In the context of multiprocessors and nonpreemptive scheduling almost all problems are NP-complete [3]. The schedulability problems for task models similar to the one described above are often represented in a form of mixed integer programs [4], where integer variables describe the order of execution of tasks. However, in case of an infeasible instance, we want to detect a reason for the infeasibility in order to be able to relax the instance in such a way that it becomes feasible. There is no obvious way for getting such information from the mixed integer program branch-and-bound tree. That is why in this project we studied and adapted a different approach, which, to the best of our knowledge, was so far not used in real-time scheduling problems. We briefly present the approach and its application for our setting.

Assume for now that all composite actors have the same period T, i.e. assume $p(c) = T$ for all $c \in C$. In that case all atomic actors will be periodic with period T. Let us interpret the set A as a set of periodic events, and let τ be a function that maps each event $a \in A$ to the event time $\tau(a) \in [0, T)$ which represents actor start time-instant. Finally, let for the ordered pair $(a_1, a_2) \in A^2$ of events $d(a_1, a_2) = [d^-(a_1, a_2), d^+(a_1, a_2)]$ be an interval from $[0, T)$. We say that the pair (a_1, a_2) satisfies the *span constraint* specified with $d(a_1, a_2)$ if there exists an integer $k(a_1, a_2)$ such that

$$d^-(a_1, a_2) \leq \tau(a_2) - \tau(a_1) + k(a_1, a_2)T \leq d^+(a_1, a_2). \quad (1)$$

A span constraint is a generalization of a precendence constraint for periodic events. For example, let $T = 12$ and $d(a_1, a_2) = [5, 7]$. If $\tau(a_1) = 0$ and $\tau(a_2) = 6$ then the event pair (a_1, a_2) satisfies the constraint for $k(a_1, a_2) = 0$ (left side of Figure 1). However, if $\tau(a_2) = 0$ and $\tau(a_1) = 6$ then the event pair (a_1, a_2) again satisfies the constraint, but now for $k(a_1, a_2) = 1$ (right side of Figure 1).

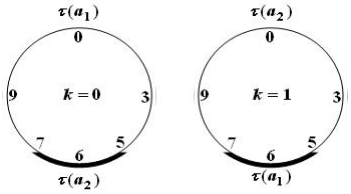

Figure 1. Event pair and span constraint

The problem of determining function τ for a set of periodic events and the set of span constraints is called Periodic Event Scheduling Problem (PESP). Let set A be the set of periodic events and set $E \subseteq A^2$ the set that specifies span constraint event pairs. Formally, an instance of the PESP consists of a period $T \in Reals$, directed constraint graph (A, E) and interval functions $d^-, d^+ : E \rightarrow [0, T)$. A solution of a PESP instance are *event-time* function $\tau : A \rightarrow [0, T)$ and *modulo* function $k : E \rightarrow Integer$ such that all span constraints given by E are satisfied. Although it may be shown that PESP is NP-complete an algorithm for it has been devised which exhibits a satisfactory average computational behavior [5].

The algorithm starts with determining a minimum spanning tree of the problem graph with respect to the cost function $(d^+ - d^-)$. An event-time function is then computed such that span constraints are satisfied for all edges of the spanning tree. The algorithm then successively adds each of the chords (edges not part of the tree) and modifies the event-time function such that the constraints are preserved also on the added edges. The algorithm either finds a feasible schedule, if it exists, or it finds edges that have too tight constraints to allow a feasible schedule to exist.

We formulate our periodic TM scheduling problem as a PESP instance. There are two types of constraints in the problem, mutual exclusion on a shared resource and precedence constraints specified in the definition of the particular composite actors. We next express these constraints as span constraints. The processing times of two different atomic

actors a_1 and a_2 on the same resource $r \in R$ cannot overlap,

$$t(a_1) \leq \tau(a_2) - \tau(a_1) + k(a_1, a_2)T \leq T - t(a_2), \quad (2)$$

for all a_1 and a_2 such that $r(a_1) = r(a_2) = r$. Let a sequence of n atomic actors $a_1, a_2, ..., a_n$ be a path from a source to a sink node of a composite actor. Then the task precedence constraints for this path can be represented as

$$t(a_i) \leq \tau(a_{i+1}) - \tau(a_i) + k(a_i, a_{i+1})T \leq t(a_i) + D, \quad (3)$$

for all $i = 1, 2, ..., n-1$, and

$$t(a_n) \leq \tau(a_1) - \tau(a_n) + (1 - \sum_{i=1}^{n-1} k(a_i, a_{i+1}))T \leq t(a_n) + D, \quad (4)$$

where

$$D = T - \sum_{i=1}^{n} t(a_i). \quad (5)$$

Note that for the set of constraints to be complete there must be a condition of the form (3) for each task precedence relation from the model. The constraints (2) and (3) match the form of span constraint (1). However, that is not the case for the constraint (4), so we adapted algorithm presented in [5].

Direct formulation of the PESP algorithm assumes that all events have the same period. We extend the framework for multi-rate systems where different composite actors can have different periods. The resulting set of span constraints is written for the period that is equal to the least common multiple of all composite actor periods $\texttt{lcm}_{c \in C}\{p(c)\}$. Span constraints for each composite actor are multiplicated accordingly and additional constraints between successive instances of atomic actors are added. Figure 2 illustrates how to transform a span constraint with $d = [d^-, d^+]$ and period T to a set of span constraints if the least common multiple is $2T$.

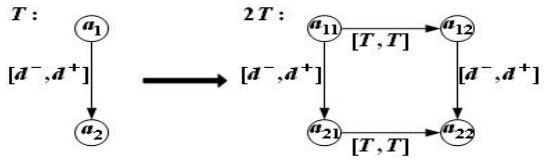

Figure 2. Span constraint transformation for tasks with multiple periods

The span constraint formulation is general enough to describe other scheduling problems with minor modifications. If a deadline Δ of a composite actor is not equal to the period T then the equation (5) should be changed to $D = \Delta - \sum_{i=1}^{n} t(a_i)$. If an actor needs more than one resource with exclusive access then constraints described by equation (2) should be written for each such resource.

3 Aperiodic TM Model

If invocation pattern of a composite actor is not known in advance, the task (actor) release times can only be determined during model execution. Consequently, computation of task execution order, i.e. the schedule generation, is dynamic and can only be suboptimal. If the schedule is not feasible we can only detect missed deadlines once they are reached during the simulation, but we cannot perform schedulability analysis. Here we generate schedule according to appropriately chosen atomic actor deadlines. Since in an aperiodic TM model only entire composite actors are given with their end-to-end deadlines, we perform a deadline distribution technique for all tasks in the precedence graphs [6, 7]. It can be shown that deadline distribution problem is NP-complete, so here we use a *laxity* heuristic procedure. Let a sequence of n task actors $a_1, a_2, ..., a_n$ be a path from a source to a sink node of a composite actor and let Δ be the deadline of this composite actor. The laxity of this path is defined as slack per path task: $L = (\Delta - \sum_{i=1}^{n} t(a_i))/n$. The path with the least laxity determines the critical task sequence of the composite actor. So, to maximize the minimal laxity we would first compute deadlines for actors on the critical path. In each iteration of the procedure a path with the least laxity is chosen, removed from the graph and the deadlines of its tasks are determined such that they all have the same slack. Figure 3 illustrates the deadline distribution for the composite actor whose task precedence graph is given with the left graph. The execution times $t(a_i)$ are given as node labels and the path with the least laxity is shown in boldface. The deadlines are shown on the right graph which is obtained after removal of the path.

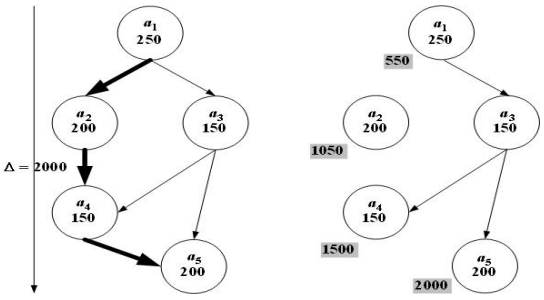

Figure 3. Iteration of the deadline distribution procedure

During the execution of the aperiodic TM model whenever a composite actor is triggered we perform the described deadline distribution procedure. Once the deadlines of all atomic actors are determined we use Earliest Deadline First strategy on each resource to generate the current order of actor execution on the resource.

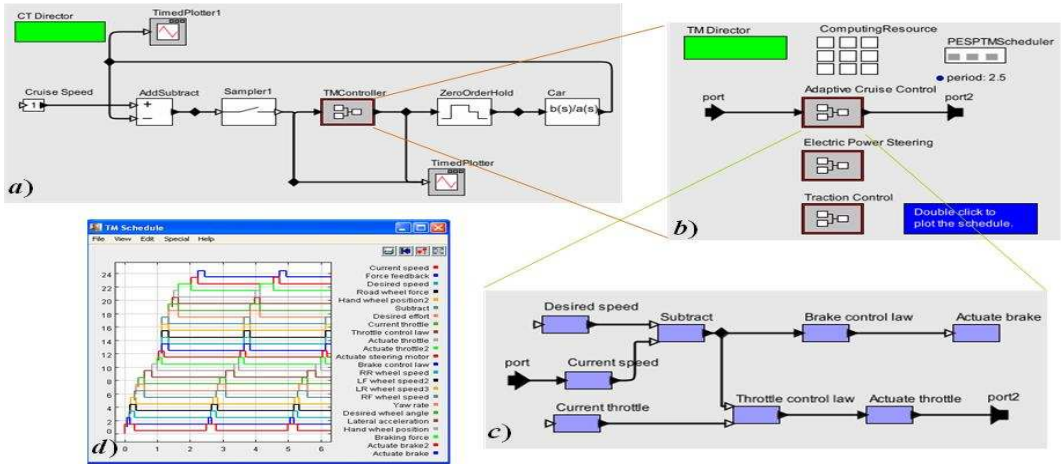

Figure 4. Timed Multitasking X-by-wire car model

4 Implementation in Ptolemy

We first present some implementation details irrespective of the scheduling algorithm that we developed. An atomic actor (i.e. task) can be triggered by an input event (i.e. token) or by a timer (i.e. at the time at which the task is statically determined to execute). The TM director uses a list to store the triggered tasks. A special component, called `TMScheduler`, is used as a plug-in to the TM director. It is up to the scheduler to decide which task should be taken from the task list of the TM director. The schedulers that implement the `TMScheduler` interface can be provided as library components. The computing resources (i.e. CPUs) and communicating resources (i.e, I/O, buses) are also modeled through a plug-in component, called `TMResource`. After the scheduler selects a task, it asks the resource to execute the task if it is available. If the resource is not available, the scheduler will put the task back to the list for later rescheduling. The execution time parameter of a task is used to decide when to output the result of the task. Another plug-in component may further be implemented to provide variable execution times according to some stochastic model.

The current implementation works with other timed domains in Ptolemy II (e.g. CT and DE), but it is not necessary to embed the TM model of software components in other models. An interesting question occurs when a periodic TM model is embedded in CT or DE model. What should be the exact sematics if a task input comes from outside of the TM model? The current implementation uses the last token if there are several tokens available at the task start time instant, or does nothing if there is no token available. This is easy to achieve if the TM model is embedded in a CT model where the CT `Receiver` is one-place buffer. How to achieve this with DE domain is still open. For some applications, e.g. control, it makes sense to drop older inputs. However, since some applications are not allowed to lose data, in the future, we will support both choices. For aperiodic case a triggering event will be buffered rather than dropped if it happens before the task can process it.

Case Study. We tested the framework on a TM+CT model of an X-by-wire automotive control system described in [7]. The continuous model component is simplified in a single car speed control loop shown in Figure 4a). As shown on Figure 4b), the `TMController` software component consists of three composite TM actors, each modeling an independent car control subsystem. Figure 4c) shows atomic tasks and precedence graph for the Adaptive Cruise subsytem (task worst-case execution times and resources are not shown). Entire model contains 24 atomic actors to be scheduled on 17 processing units. This TM model is periodic and `PESPTMScheduler` is used to generate static schedule shown on Figure 4d). In this example the run-time overhead of generating schedule is negligible, but further tests and comparisons are necessary in order to check how efficient the scheduler really is.

References

[1] J. Liu and E. Lee. Timed Multitasking for Real-Time Embedded Software. In *IEEE Control Systems Magazine*, Vol. 23, pp. 65-75, February 2003.

[2] T. A. Henzinger, B. Horowitz and C. M. Kirsch. Giotto: A Time-Triggered Language for Embedded Programming. In *Proc. IEEE*, Vol. 91, pp. 84-99, 2003.

[3] J. A. Stankovic, M. Spuri, M. D. Natale, and G. C. Buttazzo. Implications of Classical Scheduling Results for Real-Time Systems. *IEEE Computer*, pp.16-25, 1995.

[4] P. Chrtienne, E. G. Coffman, J. K. Lenstra, Z. Liu. Scheduling Theory and Its Applications. *Wiley*, 1995.

[5] P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. In *SIAM Journal on Discrete Mathematics* 2:550-581, 1989.

[6] M. D. Natale and J. A. Stankovic Title Dynamic End-to-End Guarantees in Distributed Real Time Systems. *IEEE Real-Time Systems Symposium*, pp.215-227, 1994.

[7] N. Kandasamy, J. Hayes, and B. T. Murray. Dependable Communication Synthesis for Distributed Embedded Systems. In *Proceedings of SAFECOMP 2003*, LNCS 2788, pp.275-288, 2003.