Model-based Analysis of Distributed Real-time Embedded System Composition

Gabor Madl†
Center for Embedded Computer Systems
University of California, Irvine, CA 92697
gabe@uci.edu

Sherif Abdelwahed
Institute for Software Integrated Systems
Vanderbilt University, Nashville, TN 37205
sherif@isis.vanderbilt.edu

ABSTRACT
Key challenges in distributed real-time embedded (DRE) system developments include safe composition of system components and mapping the functional specifications onto the target platform. Model-based verification techniques provide a way for the design-time analysis of DRE systems enabling rapid evaluation of design alternatives with respect to given performance measures before committing to a specific platform. This paper introduces a semantic domain for model-based analysis of a general class of DRE systems capturing their key time-based performance measures. We then utilize this semantic domain to develop a verification strategy for preemptive schedulability using available model checking tools. The proposed framework and verification strategy is demonstrated on a mission-critical avionics DRE system case study.

Categories and Subject Descriptors
I.6.4 [Simulation and Modeling]: Model Validation and Analysis

General Terms
design, verification

1. INTRODUCTION
When dealing with multiple Quality of Service (QoS) properties, DRE system designers often face a gap between design and implementation, since properties are specified in declarative way such as worst case execution times, priorities or in even broader range such as adaptable, fault-tolerant etc. On the contrary, implementations typically follow an imperative approach which makes hard to reconstruct system structure and behavior from the source code. To address these challenges, it is useful to analyze system behavior at design-time, thereby enabling developers to select suitable design alternatives before committing to specific platforms. Model-based verification techniques [2] provide a way for the design-time analysis of many engineering systems including DRE systems and allow, through abstraction, to overcome accidental complexities associated with third generation programming languages such as memory management and pointers. Furthermore, they provide a way to observe and prove properties specified in a declarative way by analyzing the composition of imperative component implementations.

In this paper, we extend the earlier works [12, 11] into the generalized DRE SEMANTIC DOMAIN – the basis for the next generation OPEN-SOURCE DREAM [7] – using a formal model of computation (MoC) that captures detailed operation settings, such as asynchronous event channels, as well as essential time-based QoS properties of DRE systems. Based on the proposed extensions, we show how the DRE SEMANTIC DOMAIN can be used to verify the preemptive scheduling of distributed multiprocess real-time embedded systems using a novel conservative discrete approximation scheme. The proposed method and case study are explained in greater detail in [10].

2. THE DRE SEMANTIC DOMAIN
In this section we introduce a computational model that can express the event-driven nature of DRE systems. The detailed formal model is presented in [10]. We define a model on a distributed platform with possible execution preemptions. DRE system models can be built by the composition of these components.

2.1 The Timer
The timer is a simple periodic event generator which releases task initiation events at a specified rate. Timers may represent sensors sampled at a predefined rate and are modeled by a generic timed automaton model shown in Figure 1.

2.2 The Task
Tasks are the main components which describe actors in DRE systems. In addition to the states specifying the execution of the task (idle, enabled, executing, preempted), we also introduce a timeout state which can be used to express undesired behavior of the system such as when the task cannot finish the execution before its respective deadline.
2.3 The Event Channel

Event propagations between tasks follow a non-blocking broadcast semantics – events which are not received are lost. We assume that this is the only case when events get lost. Event channels provide the necessary mechanism to allow reliable asynchronous communication – the publisher does not have to block until the consumer is ready to receive the event. Published events are stored in the event channel until the consumer is ready to receive them. States denoted with the C letter are committed states. This expresses time constraints, an outgoing transition has to be taken instantaneously, otherwise the system will deadlock. Event channels are generically represented by the timed automaton shown in Figure 1. Note that this model takes into account possible communication delay as represented by the channel dependent maximum delay factor $\delta$.

2.4 The Scheduler

To express the mapping of execution tasks to platform processors we introduce the scheduler modeling construct shown in Figure 1. The scheduler selects enabled tasks for execution and triggers preemptions according to the scheduling policy. The scheduler initially starts in the idle state. It will move to the select state if any tasks become eligible for execution. The selection is made instantaneously from the enabled tasks' queue and the selected task will receive the run event which triggers its execution. The scheduler moves to the idle state if no task is ready for execution.

The scheduling policy is encoded in three functions: (1) $\text{Add}(w, i)$, which increases the current priority level when task $i$ becomes ready, (2) $\text{Sub}(w, i)$, which decreases the current priority level when task $i$ becomes ready, and (3) $\text{Enable}(w, i)$, which evaluates to true if the ith task is eligible for execution. For example, in the case where priority is directly proportional to the component index, $\text{Add}(w, i) = w + 2^{-i}$, $\text{Sub}(w, i) = w - 2^{-i}$, and $\text{Enable}(w, i) = 2^{-1} \leq w < 2^i$. Other scheduling schemes can be established by defining appropriate formulas for the three functions outlined above.

2.5 Preemption Model Using Discretized Time

The proposed model of computation corresponds to the stopwatch model due to assignments of variables to clocks. Deciding the preemptive schedulability of the stopwatch model has been shown to be undecidable using timed automata [9].

To address this issue, we implement a discretized preemption scheme in which task interruption can only be granted at specific intervals during the task execution. We implement non-blocking preemptions – after the preemption the task resumes the execution from the last discrete checkpoint. The precision can be set independently for every UPPAAL Task model using the granularity parameter. Non-blocking preemptions imply longer execution time than the parameter WCET time in the preempted tasks. This method gives an overapproximation of the safe states of the system, when the schedulable discretized approximation implies a schedulable system in continuous time.

3. Problem Formulation

This paper considers the problem of deciding the schedulability of a given set of tasks with event- and time-driven interactions, on a distributed preemptive platform as described in Section 2.

**Definition 1.** The system is schedulable if all tasks finish their execution before their respective deadlines.

We use a timed automata formulation of the problem which translates the schedulability problem into a reachability problem in which the set of tasks are schedulable if a predefined error state is not reachable in any of the tasks' timed automata.

**Definition 2.** We define a frame period for a task $t \in T$, referred to as $\text{Period}(t)$ as the period of the slowest timer on which the task depends.

Tasks that are assigned to the same platform processor and have the same frame period are in the same frame. Therefore, the set of frames is a partition of the set of tasks $T$. For a task $t$ we write $\text{Frame}(t)$ to identify the set of all tasks in $T$ belonging to the frame of $t$.

**Theorem 1.** If the system is schedulable using discretized time preemptions it is also schedulable in continuous time if

$$\forall t \in T \quad D(t) \geq \text{Period}(t) - \sum_{t' \in \text{Frame}(t) - t} \text{WCET}(t')$$

where $D(t)$ denotes the deadline of the task and $\text{WCET}(t)$ denotes the worst case execution time of the task.
Figure 2: Uppaal Timed Automata Models

Figure 3: Bold Stroke application deployed on a preemptive multiprocessor platform

The proof, based on the formal specification of the DRE Semantic Domain, is presented in [10].

4. CASE STUDY

In this section we show how we applied model-based verification to a case study of a representative DRE system from the domain of avionics mission computing. Figure 3 shows the component-based architecture of the system, which is built on the Boeing Bold Stroke real-time middleware [14].

The application is deployed on a preemptive multiprocessor platform. The RateGen 4x, RADAR and TACTICAL_STEERING components are deployed on the same application server (CPU1) and are scheduled nonpreemptively. The other application server (CPU2) is a multi-threaded server, where priorities are assigned to the threads. The RateGen 2x, CURSOR_DEVICE, SELECTED_POINT and TACTICAL_DISPLAY components are executed within the context of the higher priority Thread1. Event propagations are denoted with small arrows in Figure 3 which represent the binding between event sources and sinks. Facets are represented by circles which connect to the receptacles.

4.1 Compositional Analysis Using Uppaal

We use the UPPAAL model checker tool [13] for schedulability analysis. Systems in UPPAAL are modeled as a slightly modified variant of timed automata and the specification is expressed in a restricted version of the timed computational tree logic (TCTL) [1], which is temporal logic that can formalize statements about system models.

Figure 2 shows how we modeled the system in the Uppaal model checker tool. The application consists of 12 components. RateGen components are modeled by Timers, the other 9 components are modeled using Tasks in the DRE Semantic Domain. We can abstract out some of the real-time event channel threads as discussed in [12, 11], to reduce the state space. We have to model event channels explicitly (1) when we have to buffer events or (2) on remote event channels which have measurable delays.

The scheduling policies are represented by Schedulers in the DRE Semantic Domain. Since the Bold Stroke application is deployed on a two-processor architecture we define two schedulers as shown on Figure 2. The design models (such as the model on Figure 3) can be used to generate the scheduling policies as shown in [12, 11].

The analysis models for an application can be built by composing the elements defined in Section 2. The composition is achieved by creating the dependencies using events. Priorities follow dependencies and are therefore fixed. We assign deadlines according to Theorem 1. Table 1 shows the parameter values.

We assign the granularity parameters to 1 millisecond precision in the preempted tasks as shown in Table 1. This analysis has shown that the system is schedulable with the parameters given in 1. We have checked finite buffer sizes with the following TCTL formula: $\forall [\text{Buffer}] \ (\text{Channel.bufferc < Channel.lambda})$. UPPAAL produces a counter-example for invalid properties, which helps identifying the source components are executed within the context of the lower priority Thread1. Event propagations are denoted with small arrows in Figure 3 which represent the binding between event sources and sinks. Facets are represented by circles which connect to the receptacles.

Figure 2: Uppaal Timed Automata Models

Figure 3: Bold Stroke application deployed on a preemptive multiprocessor platform
of undesired behavior. Finally, we checked that eventually every task will execute with: \texttt{E< Task.execute().

5. RELATED WORK

A generic form to analyze scheduling behavior based on the timed automata model was proposed in [4] for single processor scheduling using the Immediate Ceiling Priority protocol and the EDF algorithm. This model does not have a component model and does not support asynchronous event passing.

The Virginia Embedded Systems Toolkit (VEST) [8] is a framework designed for the reliable and configurable composition and analysis of component-based embedded systems. VEST applies key checks and analysis but does not support formal proof of correctness.


The CADENA [6] framework is an integrated environment for building and analyzing CORBA Component Model (CCM) based systems. The emphasis of verification in Cadena is on software logical properties. The generated transition system does not represent time explicitly and requires the modeling of logical time that does not allow quantitative reasoning.

Time Weaver (Geodesic) [3] is a component-based framework that supports the reusability of components across systems with different para-functional requirements. It supports code generation as well as automated analysis. It performs model-based Rate-Monotonic Analysis by real-time model checker tools such as TimeWiz\textsuperscript{26}.

6. CONCLUDING REMARKS

This paper presents the DRE SEMANTIC DOMAIN used in the next generation OPEN-SOURCE DREAM [7] model-based verification and analysis framework. We have shown how to analyze and verify the preemptive scheduling and composition of real-time embedded systems on unsynchronized distributed platforms and how model-based verification provides a way to bridge the gap between declarative specification and imperative implementation. Our results show that this approach captures delays and drifts which are common in unsynchronized reactive real-time systems. The verification is automatic, exhaustive, and capable of producing counter-examples that help pinpoint sources of undesired behavior. The method is explained in greater detail in [10].

7. REFERENCES


---

<table>
<thead>
<tr>
<th>Task</th>
<th>P</th>
<th>S</th>
<th>WCET</th>
<th>D</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>RADAR</td>
<td>HIGH</td>
<td>HIGH</td>
<td>12</td>
<td>84</td>
<td>1</td>
</tr>
<tr>
<td>TACT_ST...</td>
<td>HIGH</td>
<td>MEDIUM</td>
<td>16</td>
<td>88</td>
<td>1</td>
</tr>
<tr>
<td>CURS_D...</td>
<td>HIGH</td>
<td>HIGH</td>
<td>18</td>
<td>155</td>
<td>1</td>
</tr>
<tr>
<td>SEL_P...</td>
<td>HIGH</td>
<td>MEDIUM</td>
<td>24</td>
<td>161</td>
<td>1</td>
</tr>
<tr>
<td>TACT_D...</td>
<td>HIGH</td>
<td>LOW</td>
<td>21</td>
<td>158</td>
<td>1</td>
</tr>
<tr>
<td>INS</td>
<td>LOW</td>
<td>HIGH</td>
<td>32</td>
<td>872</td>
<td>32</td>
</tr>
<tr>
<td>GPS</td>
<td>LOW</td>
<td>HIGH</td>
<td>29</td>
<td>869</td>
<td>29</td>
</tr>
<tr>
<td>AIRFRAME</td>
<td>LOW</td>
<td>MEDIUM</td>
<td>80</td>
<td>920</td>
<td>80</td>
</tr>
<tr>
<td>NAV_D...</td>
<td>LOW</td>
<td>LOW</td>
<td>19</td>
<td>859</td>
<td>19</td>
</tr>
</tbody>
</table>