Execution-Time Aware Simulink Blocks

Andreas Naderlinger
C. Doppler Laboratory Embedded Software Systems
University of Salzburg
andreas.naderlinger@cs.uni-salzburg.at

Keywords: real-time, scheduling, preemption, execution time, Simulink

Abstract

This paper describes works in progress. We present the concept of a Simulink block that is not characterized by the typical zero execution time behavior. Instead, the execution of such blocks lasts for a finite amount of simulation time and may span several simulation steps. We are able to schedule a set of such execution-time aware blocks with a static priority approach and show preemption effects already in the simulation.

1. INTRODUCTION

Embedded real-time systems are inherently difficult to develop. The ever increasing complexity has led to a change in the development process towards model-based approaches. Model-based development aims at reducing time to market and improving quality by enabling verification and simulation. Synchronous reactive (SR) models [1] are well understood and heavily used in designing hardware logic and control applications. In SR models, a reaction (e.g. the execution of a task) to an event must be completed before the occurrence of another event. Accordingly, task execution times may assumed to be zero. A prominent software product that enables the development and simulation of SR models using block diagrams is MATLAB/Simulink [5]. Control applications are modeled, simulated together with the plant and finally synthesised into C code to be executed on the hardware.

1.1. The Simulation/Execution Mismatch

While tasks are assumed to always complete in zero time in the simulation, they are known to require a finite execution time at run-time on a hardware platform. The introduction of 'artificial' delays in the simulation is not able to overcome the mismatch between the two domains in general, as execution times may be data dependent. Additionally, at run-time, multiple tasks compete for the limited resource *CPU*. With regard to a reasonable CPU utilization, code generators synthesise multi-rate models into multi-task implementations [6]. A real-time operating system (RTOS) employs, for example, a preemptive scheduling policy to execute the tasks at the specified rate. Variations in task execution times, RTOS overhead, and preemption cause nondeterminism and data integrity problems. Even without any data dependencies between tasks, the execution of one task may have effects on the results provided by another one. To summarize, at run-time, the original semantics is not preserved.

1.2. Related Work

Numerous papers propose mechanisms to reduce the mismatch between the simulation and execution of SR models. In the following discussion, we will confine ourselves to a few approaches that are related to Simulink.

Several wait-free mechanisms have been proposed to ensure data consistency. The built-in Rate Transition (RT) block, for example, is able to guarantee data consistency for tasks with harmonic rates under rate monotonic scheduling [5]. More general buffering protocols are described in [6]. These approaches retain the data flow character of Simulink.

A more software centric approach is described in [4]: TrueTime is a toolbox for Simulink that enables the simulation of a real-time kernel (represented as a Simulink block) that is able to schedule task implementations with different policies. It facilitates the simulation of control software close to its true temporal behavior. Currently, however, it requires the implementation code to be in a special format with multiple entry points. In contrast to our approach described below, all TrueTime tasks are executed in the context of the kernel block, which consequently subsumes the whole control software.

2. TASK MODEL AND SCHEDULING

In the following, we sketch our recent research efforts to consider task execution times and scheduling effects (e.g. preemption) in models with SR semantics. In particular, we present the concept of a Simulink block that is not characterized by the typical zero execution time behavior. Instead, the execution of the block lasts for a finite amount of simulation time and may span several simulation steps. During the execution, the block may perform IO operations by reading input signals and updating output signals. Each such block represents one task in the sense of an RTOS and is described by a set of properties, such as *priority* or *worst-case execution time*, and a *task function* implemented in C. The block is based on Simulink's S-Function interface and integrates well with standard Simulink blocks.

2.1. Execution-Time Aware Task Blocks

In our model, a task is a sequential program without any internal synchronization point (comparable to OSEK's basic task concept) and can consequently not be blocked by waiting for an external event. The task function consists of a finite number of contiguous segments. Each segment has an associated execution time. Segment boundaries represent barriers that allow synchronization with the simulation environment. The execution of the segment is performed instantaneously, i.e. the operations are performed in zero execution time. However, the next segment is not executed before the execution time of the previous segment has elapsed. Execution times of the individual segments are assumed to be known, e.g. using WCET analysis tools. Currently, only periodic tasks are supported. The execution time information is part of the C code. The following sample task function has a total execution time of $1s$; it reads from the input port with index 0 exactly $0.25s$ after the task was started, and writes the output port with index 0 exactly $0.75s$ thereafter.

```
void taskFunction(...){
  Task *t = ... //this task
  InputRealPtrsType in; //input
  real_T *out; //output
  ...
  // —— segment boundary ——
  in = (InputRealPtrsType) syncInput(0, t, 0.25);
  //... use input for calculations
  // —— segment boundary ——
  out = (real_T *) syncOutput(0, t, 0.75);
  out[0] = ...; //write output
}
```

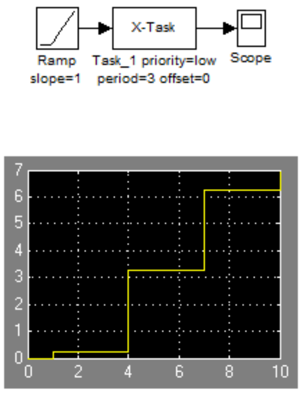
2.2. Scheduling

As described above, when executed on a hardware platform, the tasks compete for the CPU and the scheduling schema applied considerably influences the behavior. Similarly, in our simulation, all execution-time aware tasks share the CPU and are subject to scheduling. Currently, we only support *static-priority deadline monotonic scheduling* [2]. We plan to further investigate, if dynamic-priority scheduling mechanisms are also possible. Compared to other simulation environments, such as Ptolemy [3] that fosters different execution strategies (models of computation), Simulink does not give much leeway in this respect and ensuring correct semantics is nontrivial. Our current implementation assumes that only one active instance for each task exists at any time, i.e. the worst-case reaction time of each task is assumed to be smaller than its period.

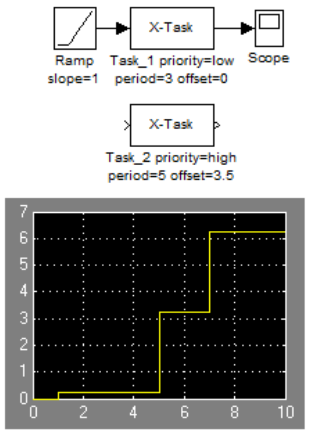
Preemption. The presented approach does not provide full preemption support. The scheduling rather relies on cooperative tasks. Tasks cannot be interrupted arbitrarily, but only at defined points (see `syncInput` and `syncOutput` calls above). As long as all IO operations are synchronized, the observable behavior is the same. It must be noted, however, that IO operations are assumed to be atomic.

2.3. Example

Figure 1 shows two Simulink models. The first (a) contains the execution time aware task from above. The task is executed every 3 secs, reads the input after $0.25s$ and outputs these values $1s$ after the start. In (b), a copy of the task is added with a higher priority. As this `Task_2` has an offset of $3.5s$, it preempts the second invocation of `Task_1`. The effect of the preemption is shown in the Scope block in (b). The output of `Task_1` is provided with a delay of $1s$ (i.e. the execution time of `Task_2`).



(a)



(b)

Figure 1. Example (a) without and (b) with preemption

2.4. Summary and Outlook

We presented a basic mechanism to simulate task scheduling and preemption effects in MATLAB/Simulink. It is based on the concept of individual execution-time aware task blocks. We will continue this work and plan to evaluate it in a top-down model-based and a bottom-up use case simulating a legacy system.

REFERENCES

[1] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. L. Guernic, Robert, and D. Simone. The synchronous languages 12 years later. In *Proc. of The IEEE*, 2003.

[2] G. C. Buttazzo. *Hard Real-Time Computing Systems*. Kluwer Academic Publishers, Boston, Mass., 2002.

[3] J. Eker, J. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. R. Sachs, and Y. Xiong. Taming heterogeneity - the Ptolemy approach. *Proc. of the IEEE, Special Issue on Modeling and Design of Embedded Software*, 91(1), 2003.

[4] D. Henriksson, A. Cervin, and K.-E. Arzen. TrueTime: Real-time control system simulation with MATLAB/ Simulink. In *Nordic MATLAB Conf.*, 2003.

[5] The MathWorks. Simulink, User's Guide, R2011b, 2011.

[6] G. Wang, M. D. Natale, P. J. Mosterman, and A. Sangiovanni-Vincentelli. Automatic code generation for synchronous reactive communication. In *Proc. of ICESS*, Washington, DC, USA, 2009. IEEE CS.