In proceedings of 12-th IEEE International Symposium on Distributed Simulation and Real Time Applications(IEEE DS-RT 2008), Vancouver, British Columbia, Canada, October 2008

Simulation and Implementation of the PTIDES Programming Model*

Patricia Derler
University of Salzburg
patricia.derler@cs.uni-salzburg.at

Edward A. Lee, Slobodan Matic
University of California, Berkeley
{eal, matic}@eecs.berkeley.edu

Abstract

We have previously proposed PTIDES (Programming Temporally Integrated Distributed Embedded Systems), a discrete-event framework that binds real-time with model time at sensors, actuators, and network interfaces. In this experimental effort we focus on performance issues and tradeoffs in PTIDES implementation. We address event processing performance with respect to other distributed discrete-event approaches that can be applied in a similar setting. The procedure is experimentally evaluated on a distributed setup with standard software and networking components.

1. Introduction

A large amount of research exists in the area of discrete-event simulation. The related modeling frameworks have successfully been applied in practice (e.g. digital circuits, networking protocols, systems of systems). The components of such models send time-stamped events to other components, and react to incoming events in chronological order. Simplicity and determinism are among the advantages of timed discrete-event formalisms.

*This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS: PRET), #0647591 (CSR-SGER), and #0720841 (CSR-CPS)), the U. S. Army Research Office (ARO #W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI#FA9550-06-0312), the Air Force Research Lab (AFRL), the State of California Micro Program, and the following companies: Agilent, Bosch, HSBC, Lockheed-Martin, National Instruments, and Toyota.

Less work has been done in adapting these formalisms as execution platforms for real-time embedded software. Whereas for simulation all that matters is that events are processed in time-stamp order, for the execution of a real-time embedded system the time instants of event processing are of the upmost importance. In particular, interaction with the environment typically involves timing constraints. We have previously proposed PTIDES [1], a discrete-event framework that binds real-time with model time (i.e., time-stamps) at sensors, actuators, and network interfaces. We studied how PTIDES can be used for locally distributed real-time embedded systems and showed its robustness and fault-tolerance properties [2].

1.1. PTIDES Programming Model

Figure 1 shows a model of an embedded system distributed over a set of platforms. In contrast to simulation, when such a model is implemented and executed, some of its components represent wrappers for sensors, actuators and network interfaces. The events at the ports of these components cannot be produced / consumed at arbitrary times.

For an output event of a sensor (e.g. Sensor1 in Figure 1), the time stamp represents the local time at which the sensor reading is taken. Thus, the local real time at which this event is produced, i.e., inserted into the event queue, is *larger* than or equal to the value of the time stamp of the event.

Conversely, for an input event of an actuator (e.g. Actuator1 in Figure 1), the time stamp represents the latest real time at which the actuator action is allowed to take place. The local real time at which this event is produced has to be *smaller* than or equal to the value of the time stamp of the event. In fact, this time stamp can be interpreted as a deadline for the delivery of the event to the actuator.

The second category of PTIDES components that bear the same timing constraints as actuators are

network output interfaces. We assume that platforms use a local network to communicate time-stamped events. In addition, we assume that the network delay is bounded and known in advance. The timing constraint and the bounded delay guarantee an upper bound, relative to the time stamp, of the real time at which a time-stamped event is received at its destination.

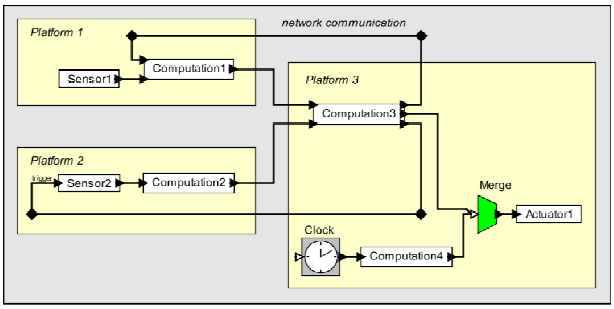

Figure 1. Distributed Embedded System

So, the PTIDES programming model uses discrete-event processing for distributed real-time software by binding time stamps to real time only where it is necessary, i.e., at sensors, actuators and network interfaces. At other components, input events are processed in time-stamp order, but an event can be processed before or after real time reaches its time-stamp.

The extensive research in distributed discrete-event simulation has focused on conservative and optimistic methods to exploit parallel computing resources [3]. However, optimistic methods (e.g. Jefferson) are of limited use for embedded system execution. This is so because they require roll back mechanisms which cannot be applied on actuators once they perform actions on the physical world.

On the other hand, conservative methods can block event processing unnecessarily long. In such methods, it is safe to process a time-stamped event only if at no time later in the execution will an event with an earlier time stamp occur. In the original Chandy and Misra method, each platform in a distributed simulator sends messages (null messages) even when there are no events to forward in order to provide lower bounds on the time stamps of future events. The PTIDES method is along the lines of several methods that attempt to reduce or entirely eliminate the null messages.

PTIDES uses static causality analysis based on the model-time parameters of the components. This analysis determines model-time path latencies that are used during execution to check whether an event can be safely processed [1]. It also enables independent events to be processed out of time stamp order. For events with mutual dependencies, the technique requires local clocks to be time-synchronized with a bounded and known error. Time synchronization, together with the real-time constraints described above, enables simple passage of time to be used to check if an event is safe to process, thus obviating the need for null messages. For instance, in order for an output event of the Computation4 component to be processed by the Actuator1 component no null message from Platform1 or Platform2 is necessary – it could safely be processed after the local clock reaches a predetermined real-time value. Thus, the PTIDES programming model prevents remote processes from blocking local ones without requiring backtracking.

An important assumption here is that the clocks on the distributed platforms are time-synchronized to some known precision. That is, at any global instant, any two clocks in the system agree on the notion of real time up to some bounded error. As discussed later, in our implementation we exploit some recent techniques for time-synchronization in local networks.

2. Simulation

In [4] we presented a family of strategies that during execution determine events that are safe to process. To study a few of these strategies together with schedulers for processing of safe events we developed a simulation environment for the PTIDES programming model as an experimental domain in Ptolemy II [5]. The Ptolemy framework is a Java-based simulation environment for modeling and simulation of heterogeneous concurrent systems. Ptolemy supports an actor-oriented design methodology. A special actor in a Ptolemy model, the director, manages the interaction of other actors thus representing the model of computation.

Figure 2 shows an example of a PTIDES model in Ptolemy. A PTIDES model consists of platforms represented by composite actors on the top-level of the model. Inside each platform, the set of actors may include components such as sensors, actuators and other computational actors. The worst case execution time of an actor, network delay and synchronization error bounds can be specified as model parameters. The current PTIDES simulation domain is multithreaded but not distributed itself. During the initialization, the PtidesDirector creates a new thread for each platform. During the actual simulation, the

execution of actors inside a platform is controlled by the PtidesEmbeddedDirector.

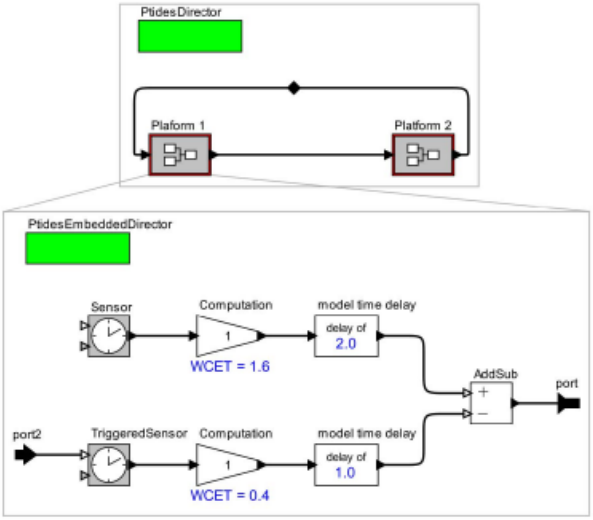

Figure 2. A PTIDES model in Ptolemy

To keep track of the execution of actors, the PTIDES domain maintains a notion of physical time. The physical time simulates real time and is manipulated by the framework. So, at each simulation step, a specified strategy may use physical time to determine whether an event is safe to process. If at the current physical time there are no events that are safe to process, the platform requests the PtidesDirector to resume execution of that platform at a future instance of physical time and the platform thread waits. When all threads are waiting, the PtidesDirector increases physical time and continues checking and executing events.

3. Implementation

Since our objective is distributed and embedded system execution, our efforts are targeted towards efficient implementations of PTIDES programming model. In particular, we want to compare PTIDES with more classical conservative distributed discrete-event implementations with respect to event processing rates and feasibility of real-time constraints.

In conservative methods, the null message mechanism is typically implemented either through periodic messages to outgoing neighbors or through polling of incoming neighbors whenever there is an event to process. For certain models both of the two approaches may require a lot of network traffic in order to prevent violation of real-time constraints. In this work we try to demonstrate how this can be avoided using our PTIDES programming model. We experiment with both timed-triggered and event-triggered communication, and measure respective network delays. PTIDES execution engine has to take into account these delays.

We are working on two experimental setups that are equipped with two different implementations of the time-synchronization protocol IEEE 1588 [6]. This recently developed protocol achieves synchronization precision in nanosecond range. It is suitable for local networks comprising of several subnets and it only minimally uses bandwidth, computing and memory resources.

Our first setup is a set of standard laptops with no specialized synchronization or networking hardware running open source software implementation of the IEEE 1588 protocol. A real-time version of the Linux kernel that runs on each laptop is needed to achieve software preemption and interrupt latencies in the order of tens of microseconds. In this project we used real-time kernel patch Xenomai [7] which runs the conventional Linux kernel as the idle task, i.e., only when all real-time tasks are inactive.

The second setup, shown in Figure 3, consists of Agilent demo nodes [8]. In this solution each node consists of an FPGA device and an embedded processor. Each FPGA device performs the IEEE 1588 protocol such that the packets of the synchronization messages are captured and time-stamped low in the protocol stack to reduce the jitter, thus increasing the precision of the synchronization. For both setups our experiments show effects of network loads on the synchronization error, and thus, on real-time properties of the model execution.

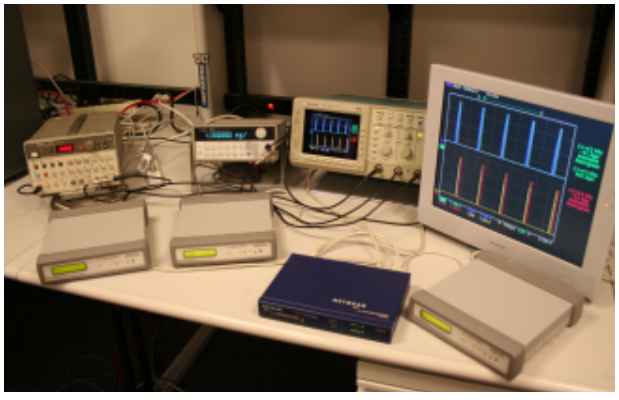

Figure 3. Experimental setup

4. References

[1] Y. Zhao, J. Liu, and E. A. Lee, "A Programming Model for Time-Synchronized Distributed Real-Time Systems". In *Proc. RTAS*, 2007, pp. 259–268.

[2] T. H. Feng, E. A. Lee. "Real-Time Distributed Discrete-Event Execution with Fault Tolerance". In *Proc. RTAS*, 2008.

[3] R. M. Fujimoto. "Parallel discrete event simulation". In *Commun. ACM*, 33(10):30-53, 1990.

[4] P. Derler, T. H. Feng, E. A. Lee, S. Matic, H. Patel, Y. Zhao, J. Zou, "PTIDES: A Programming Model for Distributed Real-Time Embedded Systems". In *Tech.Report UCB/EECS-2008-72*, EECS Department, University of California, Berkeley, 2008.

[5] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. "Heterogeneous concurrent modeling and design in Java". In *Tech. Report UCB/ERL M04/27*, EECS Department, University of California, Berkeley, 2004.

[6] J. C. Eidson, *Measurement, Control and Communication Using IEEE 1588*, Springer, 2006.

[7] P. Gerum, "Xenomai – Implementing a RTOS emulation framework on GNU/Linux". 2005

[8] Agilent Technologies, *IEEE 1588 Demonstration Kit*, 2005.