Model-Based Development of Deterministic, Event-Driven, Real-Time Distributed Systems

Patricia Derler, John C. Eidson, Edward A. Lee, Slobodan Matic, and Michael Zimmer

***Abstract*—This paper describes a model-based design approach for deterministic, event-driven real-time controllers. The model, called Ptides, allows for explicit, platform independent specification of functionality and timing. From this specification, code is generated for given target platforms. The generated code includes a lightweight operating system which performs I/O handling and scheduling as well as application specific tasks. Currently, code can be generated for 3 different platforms: a Luminary Micro board, a Renesas board and an XMOS board.**

I. Ptides

Cyber-physical systems (CPS) are integrations of (distributed) computations with physical processes. Inputs from the environment are read by computing platforms via sensors. Actuation values are computed and sent to the environment. Physical processes evolve over time, and time passes between reading sensor values and producing actuation values. Time, thus is an important property that must be taken into account when modeling CPS.

Ptides [2] is a programming model for distributed, embedded real-time systems that explicitly models time. The actor-oriented design methodology is used to describe a Ptides model. An actor reacts to input values on ports, transforms the values, modifies its state and/or sends newly computed values to connected actors through output ports. Ptides uses the discrete-event (DE) model of computation, where an event contains a timestamp and a value. When an actor is executed, it consumes events on the input ports. Operations are performed on the event values and/or the timestamps, and then output events are produced. The timestamps are given in *model time*, a logical time independent from *physical time* (real time). This physical time is typically determined by a hardware clock. Event timestamps are related to physical time at platform boundaries: at sensors, actuators and network interfaces. These devices are represented as special actors in a Ptides model. Sensors retrieve values from the environment either by periodic sampling (time-triggered) or when an event is detected in the environment (event-triggered). A sensor actor timestamps the sensed value with the current physical time. The actuator produces an actuation event when the timestamp of the event that triggered the actuator is equal to the physical time of the platform. This timestamp can also be interpreted as the deadline for the actuation.

Actors are executed based on the timestamps of the events at their input ports. An execution is correct if all actors process events that are causally related in timestamp order. Two events are causally related if an actor state or an output depends on both events; i.e. the order of processing matters. However, events can be processed out of timestamp order if they are not causally related. The scheduling of Ptides actors is split into two phases: (a) a safe-to-process analysis and (b) a scheduling algorithm. In (a), all events e are analyzed to determine whether the actor that received e can safely consume e. This means that no event e' that is causally related to e and has an earlier timestamp than e can be received later in the execution. If no such event e' can occur, e is safe to process. In (b), an event is selected out of all the events that are safe to process and the event is executed. In platforms with multiple threads or multiple processors, several events out of those safe-to-process can be chosen for execution. The safe-to-process analysis compares the timestamp of events to the physical time and can additionally take into account timestamps of other, causally related events in the system. The analysis can also be extended with information about architectural properties such as bounds on drifts between clocks on distributed platforms, delays introduced by sensor and actuator devices as well as delays on the network. [5] discusses different execution strategies for Ptides based on practicality, implementability and policies for distribution.

A Ptides model is platform independent. Whether a Ptides model can be executed on a specific platform is determined in schedulability tests. A Ptides model is schedulable if all actuators receive events early enough to perform actuations before physical time exceeds the event timestamp. Schedulability depends on execution times of actors, network delays, clock drift and device delays. For event-triggered sensors, input event patterns or minimum interarrival times of events must be determined. Because schedulability for more complex models is undecidable, we also use simulation to determine whether a certain configuration is schedulable.

We describe a tool chain in Ptolemy II [1] that allows for modeling and simulation of Ptides models and supports code generation for these models. The generated code exhibits the same behavior as the simulation model.

II. Modeling and Simulation

Ptolemy II is a modeling and simulation framework for concurrent models of computation (MoC) and the heteroge-

The authors are with the EECS Department of UC Berkeley. Contact them at {pd, eidson, eal, matic, mzimmer}@eecs.berkeley.edu.

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), #0931843 (ActionWebs), and #1035672 (CSR-CPS Ptides)), 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 Multiscale Systems Center (MuSyC), one of six research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program, and the following companies: Bosch, National Instruments, Thales, and Toyota.

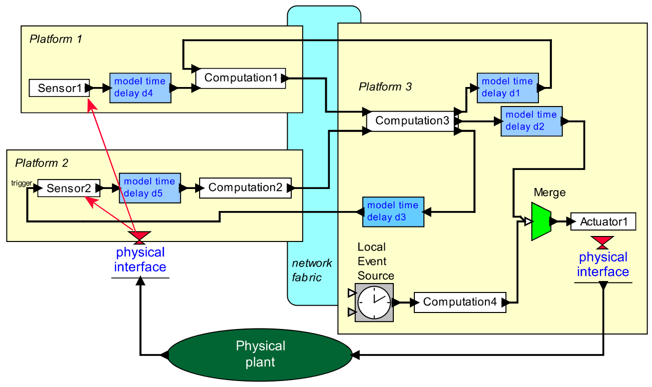

Fig. 1. A Ptides model.

neous mixture of those. Some MoCs that are implemented in Ptolemy include discrete-event (DE), continuous time (CT), synchronous reactive, dataflow and process networks. We implement Ptides as a MoC in Ptolemy and base it on DE. CT is typically used to represent physical processes. This allows for co-simulation of plant models with Ptides systems to study functional and timing properties of the entire system. Ptolemy follows the actor-oriented design principle and graphically represents models where actors are boxes with ports.

Figure 1 shows a Ptides model consisting of 3 platforms which communicate via a network. Platform 1 and 2 read values from the plant, and platform 3 writes values to the plant via actuators. The actors Computation1 through Computation4 modify values of input events, and the model time delay actors with delays d1 through d5 modify timestamps of incoming events. The local event source in platform 3 is used to trigger a periodic computation that drives the actuator. Computation1 and Computation3 receive inputs from different sources and must therefore perform a safe-to-process analysis on events to ensure correct execution.

The simulation of a Ptides model maintains three notions of time: oracle time, physical time and model time. Oracle time models global system time. Such an oracle time does not exist in real life where every platform has its own notion of time, thus it is purely a simulation artifact. For every platform, we simulate a physical time that is defined with respect to oracle time. The simulated platform time can be offset from oracle time, and this offset can be modified during the simulation to represent clock drift and clock synchronization. The timestamps of events within Ptides platforms are given in model time. Model time is used to schedule actor executions. By default, simulations do not include actor execution times and thus show instantaneous execution. However, to study schedulability properties of given applications, we also include the ability to simulate non-zero execution times.

III. Code Generation

Also built into Ptolemy is a code generation framework [3]. Within this framework, we build a code generator for Ptides, which produces platform specific code for given models. The code is split into two parts: (i) a platform specific operating system, PtidyOS [4], which performs I/O handling, context switching and scheduling of actors, and (ii) actor code. The actor code contains platform independent code for functionality and timing specifications and platform dependent code for interfacing with the OS and the hardware. PtidyOS is a lightweight operating system that only includes basic functionality to run Ptides systems and does not deal with other operating system tasks such as a file system.

In order to ensure functional and timing determinism, as well as responsiveness of the real-time program, the PtidyOS scheduler combines Ptides semantics with traditional scheduling methods. In the current implementation, earliest deadline first (EDF) scheduling is used.

The code generation framework for Ptides supports 3 different target platforms: the COTS ARM-based platform Luminary Micro LM3S8962, the Renesas 7216 Demonstration Kit and an XMOS development board with 4 XCores. The XMOS board provides timing predictability for computations, and with the multiple cores allows for parallelization of executions. However, the board does not have analog I/O, a floating-point unit nor a physical clock. The Renesas board is equipped with the DP83640 PHY chip from National Semiconductor which is used to provide hardware support for timestamping events with 8ns precision in the form of an integrated 125MHz IEEE 1588 clock through GPIO ports. For the Luminary Micro and the XMOS board, software clocks are implemented for timestamping.

On the Luminary and the Renesas board, sensor events are handled via interrupts and interrupt handler routines which perform the timestamping and event generation. The XCore processor has hardware support for event-driven communication. Events such as I/O operations are implemented in separate threads, which do not consume processing power while waiting. We study the generated code on a tunneling ball device and a printing press application. In our experiments we show that, for a given Ptides model, code generated for these platforms result in deterministic I/O behavior that is equal to the one obtained in simulations.

References

[1] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity—the Ptolemy approach. *Proceedings of the IEEE*, 91(2):127–144, 2003. Available from: http://www.ptolemy.eecs.berkeley.edu/publications/papers/03/TamingHeterogeneity/.

[2] Y. Zhao, E. A. Lee, and J. Liu. A programming model for time-synchronized distributed real-time systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 259 – 268, Bellevue, WA, USA, 2007. IEEE. Available from: http://ptolemy.eecs.berkeley.edu/publications/papers/07/RTAS/.

[3] Gang Zhou, Man-Kit Leung, and E. A. Lee. A code generation framework for actor-oriented models with partial evaluation. In Y.-H. Lee et al., editor, *Internation Conference on Embedded Software and Systems (ICESS)*, volume LNCS 4523, pages 786 – 799, Daegu, Korea, 2007. Springer-Verlag.

[4] Jia Zou. *From Ptides to PtidyOS, Designing Distributed Real-Time Embedded Systems*. PhD thesis, EECS Department, University of California, Berkeley, May 2011. Available from: http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-53.html.

[5] Jia Zou, Slobodan Matic, E. A. Lee, Thomas Huining Feng, and Patricia Derler. Execution strategies for ptides, a programming model for distributed embedded systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, San Francisco, CA, 2009. IEEE.