Poster Abstract: "PtidyOS: An Operating System based on the PTIDES Programming Model"

Shanna-Shaye Forbes, Jia Zou, Slobodan Matic and Edward A. Lee
Dept. of Electrical Engineering and Computer Sciences
545 Cory Hall, University of California, Berkeley
Berkeley, CA 94720.
{sssf, jiazou, matic, eal}@eecs.berkeley.edu

***Abstract*—Most real-time embedded software is built on programming abstractions that lack precise timing semantics. Our earlier work presented PTIDES, a programming model for distributed real-time software that delivers temporal semantics by exploiting discrete-event model of computation. In this work we introduce work we are doing to develop PtidyOS, a novel lightweight embedded operating system based on PTIDES. In PtidyOS, all event processing is done in interrupt service routines, and we only use interrupts to ensure correct mutually exclusive accesses to memory. Our approach combines PTIDES semantics with traditional scheduling methods. The first implementation leverages EDF scheduling scheme and guarantees correct event order defined by PTIDES. This is achieved without requiring totally ordered event processing. We describe a preliminary implementation on an ARM based microcontroller.** [1]

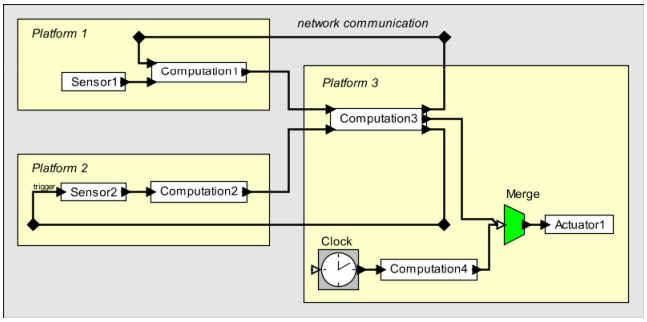

Fig. 1. Networked PTIDES Model

I. PTIDES

We previously proposed PTIDES (Programming Temporally Integrated Distributed Event Systems), a programming model for distributed platforms [3], [4]. PTIDES is based on the discrete-event (DE) model [1] of computation, in which components communicate through time-stamped events. Each component processes events in the chronological order of time stamps. PTIDES extends the DE by bounding time stamps of events at specific ports in the system to physical time. PTIDES leverages this fact to allow better processor utilization in processing events and to guarantee on time delivery of outputs.

Consider Figure 1 that shows a model of an embedded system distributed over a set of platforms. For an output event of a sensor the time-stamp represents the local time at which the sensor reading is taken. Thus, the local real time at which this event is 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, the time stamp represents the latest real time at which the actuator action should take place. The local real time at which this event is produced has to be smaller than or equal to the time stamp of the event. In fact, this time stamp can be interpreted as the 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 interfaces. We assume that platforms use a local network to communicate time-stamped events. In addition, we assume the network delay is bounded and known in advance. These constraints guarantee an upper bound of the real time at which a time-stamped event is received at its destination.

Key to making this programming model effective is to ensure that described inequality constraints are preserved at runtime. To accomplish this, we give a distributed execution strategy that obeys DE semantics without the penalty of totally ordered executions based on time stamps [4]. For instance, at components other than sensors, actuators and network interfaces, input events are processed in time-stamp order, but an event can be processed at a real time unrelated to its time stamp.

PTIDES uses static causality analysis to determine path latencies in a model. These latencies are used during execution to check whether an event can be safely processed. In general, 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 physical time to be used to check if an event is safe to process.

[1]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) 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, Lockheed-Martin, National Instruments, and Toyota. It was also supported in part by the NASA Harriet G. Jenkins Pre-doctoral Fellowship Program (JPFP).

II. PTIDYOS

The PTIDES programming model relies on the discrete-event semantics to achieve determinism. PtidyOS is a proposed lightweight kernel with timing determinism guaranteed by the PTIDES semantics. Currently, our early PtidyOS prototype runs only on a single processor. Like TinyOS, an operating system widely used in sensor networks, PtidyOS is a library that is linked against application C code, and it uses interrupts as the sole mechanism to ensure correct memory synchronization behaviors. In addition, as is common in many real-time microkernels, we do not support dynamic memory allocation.

To reduce the latencies between PtidyOS software components, all event processing is performed in interrupt service routines. In our prototype, interrupts do not have priorities, only events do. In particular, an interrupt is able to preempt any other interrupt service routine. Though this is not directly supported by most off-the-shelf microprocessor architectures, we achieve this behavior through stack manipulations. Using event priorities, the PtidyOS scheduler would return to the preempted interrupt only if it was executing an event of higher priority than the preempting one.

We divide our local execution strategy into two layers: safe-to-process coordination, and local resource scheduling. When receiving an event, the coordination layer determines whether the event can be processed immediately or it has to wait for other potentially preceding events. In particular, a safe-to-process component of PtidyOS is used to specify when in physical time this event can be processed.

Once it is sure that the current event can be processed according to DE semantics, the event is handed over to a local resource scheduler, which uses existing real-time scheduling algorithms to prioritize the processing of all safe-to-process events. Our preliminary investigations indicate some interesting properties. In particular, PTIDES models provide a solid foundation for earliest deadline first (EDF) scheduling, where time-stamped events have a statically-computable deadline based on causality analysis. While the PTIDES semantics define a set of events that are safe to process, the deadline-ordered queue ensures the event with smallest deadline from that set is processed.

Our prototype platform for PtidyOS is the Luminary microcontroller LM3S8962. It is based on the 32-bit ARM Cortex-M3 controller (50 MHz, 256 kB flash, 64 kB SRAM) and equipped with a series of standard peripheral devices. Our preliminary experiments were performed on relatively simple graphs of actor components with rates of input events in the millisecond order. We demonstrated time-stamp deterministic output behavior with microsecond level jitter which is generally acceptable in embedded controllers.

III. WORK IN PROGRESS

Our main objective in PtidyOS is to investigate the potential advantages of PTIDES model for locally distributed systems. The Luminary board contains an Ethernet controller that is augmented with hardware assisted IEEE 1588 Precision Time Protocol (PTP). PTP is a protocol for time synchronization of processors over local networks that support multicast communication. With this hardware, on small local switched Ethernet networks we expect to be able to achieve synchronization error bounds of about one microsecond.

It appears that it may be possible to implement PTIDES with EDF using a single stack, thus avoiding much of the overhead of multithreading that dramatically increases timing variability in conventional real-time operating systems. However, we observe that such a solution may lead to priority inversion problems. As a part of the event scheduling algorithm, we plan to develop a more sophisticated stack manipulation algorithm for PtidyOS.

The current PtidyOS design flow requires an executable PTIDES model to be built in C, with causality analysis of the model performed during system initialization. Moreover, the current causality analysis works only for a subset of PTIDES models but not for hierarchical and modal models for instance. These limitations motivate us to employ the Ptolemy II framework along with its code generation component [2] as a design and implementation platform for PTIDES models. Beside extending the set of PTIDES models, the partial evaluation methods of code generation would allow efficient memory and processor usage.

Finally, in a standard RTOS, a device driver typically interacts with the executing program through shared memory. This threading model is often problematic, and may result in system crashes due to unexpected device driver interactions. Our preliminary design shows that writing device drivers for PtidyOS may in fact be much easier because of the clean concurrency model. We plan to experiment with PtidyOS sound drivers since audio applications often require millisecond precision.

REFERENCES

[1] R. M. Fujimoto. Parallel discrete event simulation. *Commun. ACM*, 33(10):30–53, 1990.

[2] Gang Zhou, Man-Kit Leung, and Edward A. Lee. A code generation framework for actor-oriented models with partial evaluation. In *International Conference on Embedded Software and Systems*, LNCS 4523, pages pp. 786–799, May 2007.

[3] Y. Zhao, J. Liu, and E. A. Lee. A programming model for time-synchronized distributed real-time systems. In *Proceedings of RTAS*, pages 259–268, Bellevue, WA, USA, Apr 2007.

[4] J. Zou, S. Matic, E. A. Lee, T. H. Feng, and P. Derler. Execution strategies for ptides, a programming model for distributed embedded systems. In *to appear in RTAS*, 2009.