2009 29th IEEE International Conference on Distributed Computing Systems Workshops

The Case for Timing-Centric Distributed Software

— Invited Paper —

Edward A. Lee, Slobodan Matic, Sanjit A. Seshia, and Jia Zou
EECS Department
University of California, Berkeley
Berkeley, CA, USA
{eal,matic,sseshia,jiazou}@eecs.berkeley.edu

Abstract

This paper makes the case that the time is right to introduce temporal semantics into programming models for cyber-physical systems. Specifically, we argue for a programming model called PTIDES that provides a coordination language rooted in discrete-event semantics, supported by a lightweight runtime framework and tools for verifying concurrent software components. PTIDES leverages recent innovations in network time synchronization to deliver distributed real-time systems with determinate concurrent semantics, decentralized and robust control, and the potential for rigorous schedulability analysis.

1. Introduction

The problem we address is that real-time embedded software today is commonly built using programming abstractions with little or no temporal semantics. Stankovic et al. [33] cite a need to "raise the level of programming abstraction," stating that "existing technology for RTES [real-time embedded systems] design does not effectively support the development of reliable and robust embedded systems." In this paper, we argue for an approach that raises the level of abstraction, but does much more. It also fundamentally changes the abstractions that are used. Timing cannot be effectively introduced at "higher levels of abstraction" if it is entirely absent from the lower levels of abstraction on which these are built.

We focus on computer-based systems where multiple computers are connected on a network and interact with and through physical processes via sensors and actuators. Such integrations of computing, networking, and physical dynamics are referred to as *cyber-physical systems* (CPS). The passage of time becomes a central feature — in fact, it is key to distinguishing the cyber part of CPS from distributed computing in general. The lack of temporal semantics in programs and networks makes principled design difficult, leaving engineers with a prototype-and-test style of design, which leads to brittle systems that do not easily evolve to handle small changes in operating conditions and hardware platforms.

The foundations of computing, rooted in Turing, Church, and von Neumann, are about the transformation of data, not about physical dynamics. Although computers have become fast enough to adequately measure and control many physical processes, modern techniques such as instruction scheduling, memory hierarchies, garbage collection, multitasking, best-effort networking, and reusable component libraries (which do not expose temporal properties on their interfaces), introduce enormous variability. Those innovations are built on a key premise: that time is irrelevant to correctness; it is at most a measure of quality. Faster is better, if you are willing to pay the price. By contrast, what CPS needs is not faster computing, but physical actions taken at the *right time*. Time needs to be a semantic property, not a quality factor.

We have been lulled into a false sense of confidence by the considerable successes of embedded software, for example in automotive, aviation, and robotics applications. But the potential is vastly greater; we

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. The third author was supported in part by NSF CAREER award #0644436.

have reached a tipping point, where computing and networking may be integrated into the vast majority of artifacts that humans make. However, as we move to more networked and more intelligent applications, the problems are going to get worse. Embedded systems will no longer be black boxes, designed once and immutable in the field. Instead, they will be evolving pieces of a larger system, a dance of electronics, networking, and physical processes.

Applications of CPS demand solid foundations. They include high confidence medical devices and systems, assisted living, traffic control and safety, advanced automotive systems, process control, energy conservation, environmental control, avionics, instrumentation, critical infrastructure control (electric power, water resources, and communications systems for example), distributed robotics (telepresence, telemedicine), defense systems, and manufacturing. It is easy to envision new capabilities that are technically well within striking distance, but that would be extremely difficult to deploy using today's methods. Consider, for example, a city without traffic lights, where each car provides the driver with adaptive information on speed limits and clearance to pass through intersections. We have in hand all the technical pieces for such a system, but achieving the requisite level of confidence in the technology seems decades off.

In this paper, we focus on applications with structure like that sketched in Figure 1, which shows a small example with three networked compute platforms each with its own sensors and actuators. The actuators affect the data provided by the sensors through the physical plant. In an automation application, for example, the actuators could be motion controllers for high-speed printing presses, the sensors could detect disruptions, and the control algorithms could include rapid shutdown modes to prevent damage to the equipment in case of paper jams. Such shutdowns need to be tightly orchestrated across the entire system to prevent disasters. Similar situations are found in high-end instrumentation systems and in energy production and distribution.

Today's "best effort" operating system and networking technologies cannot produce the levels of precision and reliability that most of these applications demand, and indeed such systems are not widely used in such applications. Our objective is a technology for *robust* and *predictable* designs with *repeatable* temporal dynamics (for a detailed discussion of the meanings of these terms, see [20]). We do this by building on a rigorous formal model that reflects the realities of distributed systems. The result will be cyber-physical designs that can be much more extensively networked, can include more adaptive control logic, and can evolve over time, without suffering from the *brittleness* of today's designs, where small changes have big consequences. We do this by making timing central to distributed software design.

In this paper, we make a case for a model for timing-centric distributed software called PTIDES (*programming temporally-integrated distributed embedded systems*, pronounced "tides"), a model-based programming technique for CPS [37]. PTIDES models define the interaction of distributed software components, the networks that bind them together, and the interaction via sensors and actuators with physical dynamics. PTIDES bases software models on discrete-event (DE) systems [29], [4], [1], [36] which provide a model of time and concurrency. DE models have traditionally been used to construct simulations, but PTIDES uses them as a programmer's model for deployable cyber-physical systems. It uses one of several variants of DE that has a rigorous, determinate, formal semantics [23], [19], and that has been shown to integrate well with models of continuous dynamics [21]. A practical consequence is to enable co-simulation of software controllers, networks, and the physical plant. It also facilitates *hardware in the loop* (HIL) simulation, where deployable software can be tested (at greatly reduced cost and risk) against simulations of the physical plant. The DE semantics of the model ensures that simulations will match implementations, even if the simulation of the plant cannot execute in real time. Conversely, prototypes of the software on generic execution platforms can be tested against the actual physical plant. The model can be tested even if the software controllers are not fully implemented. This (extremely valuable) property cannot be achieved today because the temporal properties of the software emerge from an implementation on a specific platform, and therefore complete tests of the dynamics often cannot be performed until the final stages of system integration, with the actual physical plant, using the final platform.

The idea of using DE as a programming model instead of a simulation technology was introduced in [37]. Derler et al. [6] describe a simulator for PTIDES built on Ptolemy II [8], and give some preliminary measurements of implementation properties on a small network of prototype platforms using a pre-commercial implementation of IEEE 1588 from Agilent. Feng et al. [9] describe a strategy for incorporating fault-tolerance principles into PTIDES using backtracking techniques.

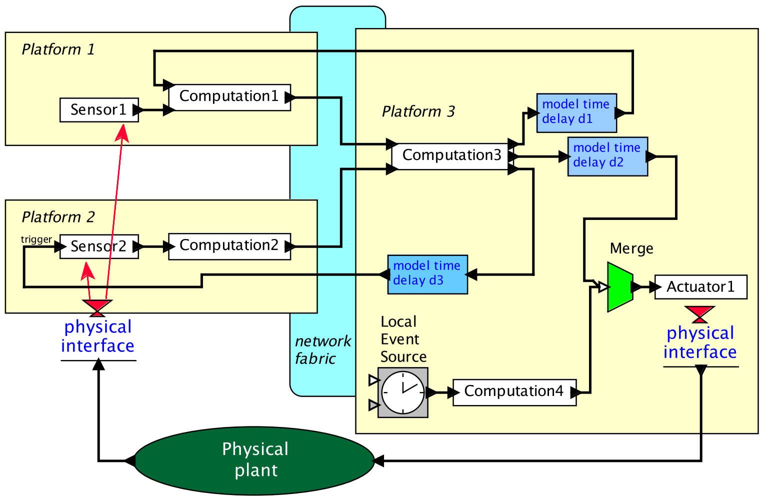

Figure 1: Example structure of applications considered.

2. PTIDES Overview

We explain the basic PTIDES model by referring to Figure 1. That figure shows three computational platforms (typically embedded computers) connected by a network and having local sensors and actuators. On each platform, there are a number of software components interconnected by communication paths. The communication paths carry time-stamped data packets (each of which is called an *event*). Events are processed by components in time-stamp order.

In Figure 1, on Platform 3, a component labeled “Local Event Source” produces a sequence of events that drive an actuator through two other components. The component labeled “Computation4” processes each event and (typically) produces an output event with the same time stamp as the input event that triggers the computation. Those events are merged in time-stamp order by a component labeled “Merge” and delivered to a component labeled “Actuator1.” The actuator component interprets its input events as commands to perform some physical action at a physical time (wall-clock time) corresponding to the time stamp of the event. This interpretation imposes a real-time constraint on all the software components upstream of the actuator. Each event must be delivered to the actuator at a wall-clock time earlier than the event’s time stamp. This portion of the model represents, for example, some local control of an actuator that is by default unaffected by sensor data. However, the Merge component can inject commands to the actuator depending on sensor data obtained from the network. How those commands are merged with the local commands is entirely determined by the time stamps accompanying the commands.

In Figure 1, we see that the second input to the Merge component comes from components that get inputs from sensors on the remote platforms. The sensor components, like the actuator components, are (typically thin) software wrappers around hardware drivers. They produce on their output ports time-stamped events. Here, the PTIDES model imposes a second relationship between model time stamps and wall-clock time. Specifically, when a sensor component produces a time-stamped output event, that time stamp must be less than or equal to wall-clock time. The sensor can only tell the system about the past, not about the future.

Under benign conditions [23], [19], DE models are determinate in that given the time-stamped inputs to the model, all events are fully defined. Thus, any correct execution of the model must deliver the same time-stamped events to actuators, given the same time-stamped events from the sensors (this assumes that each software component is itself determinate). An execution of a PTIDES model is required to follow DE semantics, and hence deliver this determinacy. It is this property that makes executions of PTIDES models *repeatable*. A test of any “correct” execution of a PTIDES model will match the behavior of any other correct execution.

The key question becomes how to deliver a “correct” execution. Consider in particular the Merge component in Figure 1. This component must merge events in time-stamp order for delivery to the actuator. Given an event from the local Computation4 component, when can it safely pass that event to the actuator? Here lies a key feature of PTIDES. The decision to pass the event to the actuator is made locally by comparing the time stamp of the event against a local clock that is tracking wall-clock time. This strategy results in decentralized control, removing the risks introduced by a single point of failure, and making systems much more modular and composable.

How is it done? There are two key assumptions made in PTIDES. First, distributed platforms have time synchronized wall-time clocks with bounded error. The PTIDES model of computation works with any bound, but the smaller the bound, the tighter the real-time constraints can be. Second, PTIDES requires that there be a bound on the communication delay between any two hardware components. Specifically, a sensor must deliver a time-stamped event to the run-time system within a bounded delay, and a network must transport a time-stamped event with a bounded delay. These assumptions are achievable in practice. In particular, time synchronization techniques [14] such as IEEE 1588 [12] can deliver wall-clock synchronization with bounded errors. In fact, an Ethernet PHY chip introduced in 2007 by National Semiconductor advertises a clock precision on the order 8 ns on a local-area network. Such high precision concurrence of clocks on a network is a game-changing phenomenon that enables a radically different approach to distributed software development.

An interesting example supporting this point is General Electric's Mark^{TM}VIe Control Platform, which has integrated high-precision network time synchronization based on IEEE 1588 into its IO processors. To date, GE has manufactured in excess of 50,000 such units. This control platform is used for gas and steam turbine controls, wind turbines, hydro control, and other distributed control systems. A current challenge for such systems is to enable distributed micro power generation coupled into the power grid. The complexity of the control system becomes much higher, and its structure dynamic. PTIDES could facilitate such a transformation of the power grid.

Bounding network delay is potentially more problematic when using generic networking technologies such as Ethernet and TCP/IP. However, we observe that bounded network delay is required already today in the applications we consider. This has in fact historically forced deployments of these applications to use specialized networking techniques (such time-triggered architectures [18], FlexRay, Foundation Fieldbus systems, and CAN busses). More recent developments, however, promise similar bounds on top of generic technologies. Synchronous Ethernet and Time-Triggered Ethernet are two such promising examples. The GE system mentioned above uses generic Ethernet, leveraging IEEE 1588 time synchronization to facilitate frame synchronization and avoid the network collisions and buffer overflows that make bounded communication latencies difficult.

Once we accept these two assumptions (bounded time synchronization error and bounded communication latencies), we can see how local decisions can be made to deliver events in Figure 1 without compromising the determinate DE semantics. Specifically, in Figure 1, notice that the top input to the Merge comes from Sensor1 and Sensor2 through a chain of software components and a network link. Static analysis of these chains reveals the operations performed on time stamps. In particular, in this figure, we assume that the only components that manipulate time stamps are the shaded components labeled "model time delay d_i". These components accept an input event and produce an output event with the same data payload but with a time stamp incremented by d_i. Examining the figure, we see that the direct path from Sensor1 to Actuator1 contains only one delay component, and that component increments the time stamp by d_2. If the sensor delay of Sensor1 is bounded by s_1, the network latency is bounded by n, and the clock synchronization error is bounded by e, then an event with time stamp τ on the bottom input of the Merge component can be safely passed to the actuator when local wall-clock time exceeds $\tau + s_1 + n + e - d_2$.

It is easy to see that if the model is static (components are not added during runtime and connections are not changed), then given enough information about each component, a similar analysis can be made for any point in the model where two streams of events come together. This analysis is done at design time. At run time, the only test performed is to compare time stamps to wall-clock time, suggesting that efficient execution is possible. PTIDES components include causality interfaces with superb algebraic compositionality properties [39], enabling automatic analysis.

Note that the distributed execution control of PTIDES introduces another valuable form of *robustness* in the system. In particular, in figure 1, if, say, Platform 1 ceases functioning altogether, and stops sending events on the network, that fact alone cannot prevent Platform 3 from continuing to drive its actuator with locally generated control signals.

It is also easy to see that PTIDES models can include components that monitor system integrity. For example, Platform 3 could raise an alarm and change operating modes if it fails to get messages from Platform 1. Time synchronization with bounded error helps to give such mechanisms a rigorous semantics. Moreover, since execution of a PTIDES model carries time stamps at run time, violations of deadlines at actuators can be detected at run time. These can only occur when some assumption is violated, for example due to a fault condition. Most interestingly, in certain circumstances, it is possible to detect deadline violations *before they occur*! This could enable a style of fault-tolerant design where mode changes can be triggered by timing violations that will inevitably occur in the future. More conservative systems could trigger mode changes merely because it is no longer possible to *guarantee* that deadlines will be met in the future. A careful design could do this far enough in advance to ensure physical safety. In general, PTIDES models provide adequate runtime information for detecting and reacting to a rich variety of timing faults.

In all PTIDES models, time stamps represent a *model time* at which the event occurs. Model time need not have any relationship to physical time (wall-clock time), except at sensors and actuators.[1] Thus an execution of a PTIDES model has considerable freedom to process an event earlier or later than the wall-clock time corresponding to the time stamp of the event. As long as events are delivered on time to actuators, the execution will look exactly the same to the environment. This makes PTIDES models much more *robust* than typical real-time software, because small changes in the (wall-clock) timing of internal events are not visible to the environment (as long as real-time constraints are met). Moreover, the run-time system, since it explicitly handles time stamps, has a rigorous basis for determining whether real-time constraints are being met. PTIDES models can be made adaptive, changing modes of operation for example when real-time violations occur.

For discrete-event *simulations*, the most common use of DE modeling, the time stamps typically have no connection with real time, and can advance slower or faster than real time [36]. The time stamps can be real numbers (or their approximations as floating point numbers). This is the choice of many commonly used discrete-event simulators. They may instead be given by integers, as used by most hardware description languages. In our case, we approximate a superdense model of time [24] because it facilitates models that mix continuous dynamics with discrete-event models [21]. Superdense time stamps are a tuple (t, n), where t is a real number, a floating point number, or an integer, and n is an integer. Superdense time supports a notion of a sequence of causally-related simultaneous events. In a deployable run-time framework, one could use the time stamp format of IEEE 1588 augmented as needed with an index n to make the model superdense. But the PTIDES programming model and semantics work for any totally ordered set of time stamps.

1. In our preliminary work with this model, we have discovered that if we also associate model time and real time at network interfaces, then we can get better decoupling in a decentralized scheduling strategy [10].

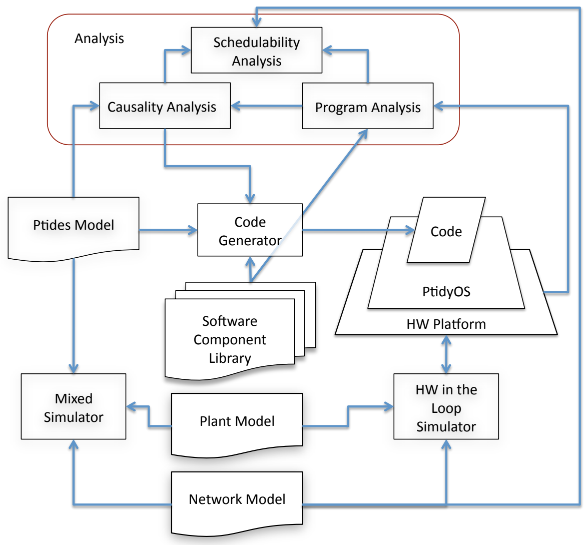

Figure 2: The structure of a PTIDES development environment.

3. How Ptides will Work

We envision a suite of tools supporting modeling and design of PTIDES applications. The overall structure is given in Figure 2. First, a system designer will specify a model by constructing a diagram like that in Figure 1, or by giving an equivalent textual description. This creates a PTIDES model, shown on the left of Figure 2. These models can be fairly coarse grained, where the individual software components may be specified in C, as is commonly done in many embedded applications, or in domain-specific languages. The PTIDES model itself is a coordination layer above conventional software, which is represented in the figure as the "Software Component Library." That is, a PTIDES model defines the causal relationships between pieces of conventional software, written in C or generated from other domain-specific models. A code generator (like that in Ptolemy II [38]) will consolidate and glue together these software components to produce executable programs for each of the platforms in the network. This approach enables designers to integrate legacy code and to leverage their in-house experience base, thus enhancing the prospects that our techniques will be accepted.

We envision a lightweight runtime framework, tentatively called PtidyOS, that realizes the distributed real-time scheduling strategy. Like the high impact TinyOS system [11], this runtime framework could be a library that is linked against the application code, rather than an operating system that is booted and then asked to execute an application. The major requirement of

PtidyOS is that it must guarantee the DE semantics through a scheduling service that dispatches software components. A second requirement is that it must be able to detect violations of the assumptions under which it guarantees DE semantics (such as the bound on clock synchronization error, which may be detected when out-of-order time stamps are encountered). A third requirement is that it must be able to execute PTIDES models that are provably capable of meeting all deadlines in such a way that all deadlines are indeed met. That is, the scheduler must be optimal in terms of feasibility [3].

A suite of analysis and verification tools will be used to develop PTIDES applications on top of PtidyOS. The code generator will make use of causality analysis performed using algebraic techniques adapted from [39] in order to synthesize the runtime control. Schedulability Analysis will certify for a particular platform that real-time constraints can be met, building on top of the causality analysis and program analysis that computes worst-case execution time (WCET) estimates for software components. A major challenge in program analysis will be to deal with concurrency. Concurrent software components will be a central part of PTIDES implementations. For example, in Figure 1, the Sensor1 component (which could be an interrupt service routine) runs concurrently with the Computation1 component, which additionally invokes network send and receive calls. Timing analysis of Computation1 thus needs to consider all of its possible interleavings with the Sensor1 component, as well as the impact of network delays. Current techniques for worst-case execution time analysis do not work well for concurrent, networked programs; in a recent survey, Wilhelm et al. [35] report that almost all tools can only consider uninterrupted execution of tasks. Techniques will have to be developed for execution time analysis of concurrent software in a networked environment. Note, however, that the PTIDES model and the PtidyOS design offer opportunities to simplify the analysis by imposing constraints on the environment of components, such as bounding the number of pre-emptions of tasks in order to reduce the number of interleavings to be considered.

A Mixed Simulator can be provided that can combine the PTIDES model with a plant model, using the techniques of [21]. A HIL simulator can be provided to stitch together the plant model and the deployable hardware platforms. The semantics in [21] ensures that such a combination will match any correct execution of the model, regardless of whether the plant simulation executes in real time. This addresses a recognized challenge in HIL simulation, the synchronization of a potentially large number of I/O functions.

Obviously, a great deal remains to be done before this vision can be realized. The purpose of this paper is to argue that this agenda is worth pursuing.

4. Why the Time is Right

Real-time software is not a new problem. However, recent trends have drastically changed the landscape. Model-based design [15], for example, has caught on in industrial practice, through the use of tools such as Simulink, Real-Time Workshop, DSpace, LabVIEW, and SCADE. PTIDES is a model-based design approach. Domain-specific modeling languages are increasingly being used because they tend to support higher-level abstractions than general-purpose modeling languages such as UML. An example of such a language is Timing-Augmented Description Language [13], a domain-specific language recently developed within the automotive initiative AUTOSAR. The multiplication of modeling languages raises the question of mutual consistency and interoperability. This is one of the main reasons why the OMG consortium extended UML with a profile called MARTE (Modeling and Analysis of Real-Time and Embedded Systems) [26].

In addition, while computer architecture has traditionally focussed on improving average-case performance, there has been a recent work on precision timed (PRET) machines that offers new opportunities to improve temporal precision [7]. This could make the analysis of PTIDES models stronger, leading to systems that can be certified at moderate cost. Moreover, the move to multicore architectures offers an opportunity in that it requires new programming abstractions, so the community is receptive to advances in programming models.

Bounded network latencies are required by PTIDES in order to guarantee that deadlines are met. Innovations in real-time networking are making it to mainstream industrial practice. Network time synchronization is available on a variety of platforms, with IEEE 1588 being particularly attractive for our target application space. Real time networks such as TTA and FlexRay [17] have also caught on, and their techniques are starting to appear on more generic networking infrastructure such as Ethernet.

Another trend is the acceptance of synchronous/reactive languages, particularly SCADE [2], in safety critical applications. PTIDES borrows sound fixed-point semantics from the synchronous languages by basing on a rigorous form of DE, but is more flexible and concurrent, and it embraces legacy code as components.

We believe that PTIDES will be amenable to automated schedulability analysis, at least for an interesting and useful subset of PTIDES models. This will ultimately rely on being able to determine worst-case execution times (WCET) for code segments. Fortunately, there have been recent advances in WCET estimation [35], making these techniques more useful in practice. However, several obstacles remain [16], most particularly the brittleness of current tools to changes in the program or platform arising in part from the need for detailed manual modeling of the particular target processor. PRET machines and new robust timing analysis methods [32] promise to ameliorate this problem.

Schedulability analysis for PTIDES will also require event models characterizing the behavior of sensors. The real-time interfaces of [34] could help a great deal here. Older contributions, such as the taxonomy of timing properties that must be expressible given in [22] and the annotations on untimed languages given in [25], could also provide a framework for event models. Prior work on timing constructs in languages, such as Ada and SystemC, can contribute some mechanisms, but these languages do not emphasize the timing of I/O interactions, which is what PTIDES does, and hence do not match as well the CPS agenda.

Since our approach is to provide a coordination language, prior work on software component technologies is very relevant. General-purpose distributed software is dominated by distributed object-oriented programming [31] using frameworks such as CORBA, SOAP, DCOM, EJB, and XML Web Services. Some extensions of these frameworks, such as ACE/TAO [30], support real-time scheduling concepts, and have caught on in certain communities (such as avionics) [28]. Whereas object-centric real-time CORBA is best suited for synchronous transactions, a data-centric Publish/Subscribe paradigm is used for quick dissemination to many nodes and flexible delivery requirements. The Data Distribution Service [27], a recent OMG standard in this direction, supports over 20 configurable Quality of Service options. Although messaging technologies are advertised with low latency and high throughput, the reality is more complex since the proper level of control, stability and expressiveness is required for complete solutions.

In embedded applications such as industrial control, component technologies such as International Electrotechnical Commission's IEC 61131 have emerged for programming PLCs and have been extended to distributed control systems (e.g. IEC 61499). The latter extensions have not proved satisfactory because of nondeterminism in implementations. The same standard-compliant application running in two different implementations of the runtime environment may result in different behaviors [5]. PTIDES can fix this problem.

5. Acknowledgements

We would like to thank the following people for thoughtful suggestions and discussions that have helped us understand our approach: Hugo Andrade, Christopher Brooks, Christian Buckl, Patricia Derler, John Eidson, Thomas Feng, Shanna-Shaye Forbes, Alois Knoll, Ben Lickly, Isaac Liu, Hiren Patel, Stavros Tripakis, and Yang Zhao.

References

[1] F. Baccelli, G. Cohen, G. J. Olster, and J. P. Quadrat. *Synchronization and Linearity, An Algebra for Discrete Event Systems*. Wiley, New York, 1992.

[2] G. Berry. The effectiveness of synchronous languages for the development of safety-critical systems. White paper, Esterel Technologies, 2003.

[3] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, second edition, 2005.

[4] C. G. Cassandras. *Discrete Event Systems, Modeling and Performance Analysis*. Irwin, 1993.

[5] G. Cengic, O. Ljungkrantz, and K. Akesson. Formal modeling of function block applications running in IEC 61499 execution runtime. In *11th IEEE International Conference on Emerging Technologies and Factory Automation*, Prague, Czech Republic, 2006.

[6] P. Derler, E. A. Lee, and S. Matic. Simulation and implementation of the ptides programming model. In *IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Vancouver, Canada, 2008.

[7] S. A. Edwards and E. A. Lee. The case for the precision timed (PRET) machine. In *Design Automation Conference (DAC)*, San Diego, CA, 2007.

[8] 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.

[9] T. H. Feng and E. A. Lee. Real-time distributed discrete-event execution with fault tolerance. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, St. Louis, MO, USA, 2008. IEEE.

[10] T. H. Feng, E. A. Lee, H. D. Patel, and J. Zou. Toward an effective execution policy for distributed real-time embedded systems. In *14th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, St. Louis, MO, USA, 2008.

[11] J. Hill, R. Szewcyk, A. Woo, D. Culler, S. Hollar, and K. Pister. System architecture directions for networked sensors. In *9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 93–104, 2000.

[12] IEEE Instrumentation and Measurement Society. 1588: IEEE standard for a precision clock synchronization protocol for networked measurement and control systems. Standard specification, IEEE, 2002.

[13] M. Jersak. Timing model and methodology for autosar. In *Elektronik Automotive. Special issue AUTOSAR*, 2007.

[14] S. Johannessen. Time synchronization in a local area network. *IEEE Control Systems Magazine*, pages 61–69, 2004.

[15] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-integrated development of embedded software. *Proceedings of the IEEE*, 91(1):145–164, 2003.

[16] R. Kirner and P. Puschner. Obstacles in worst-case execution time analysis. In *Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, pages 333–339, Orlando, FL, USA, 2008. IEEE.

[17] H. Kopetz. *Real-Time Systems : Design Principles for Distributed Embedded Applications*. Springer, 1997.

[18] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.

[19] E. A. Lee. Modeling concurrent real-time processes using discrete events. *Annals of Software Engineering*, 7:25–45, 1999.

[20] E. A. Lee. Computing Needs Time. Technical Report UCB/EECS-2009-30, EECS Department, UC Berkeley, February 18 2009. To appear in *Communications of the ACM*, May, 2009.

[21] E. A. Lee and H. Zheng. Leveraging synchronous language principles for heterogeneous modeling and design of embedded systems. In *EMSOFT*, Salzburg, Austria, 2007. ACM.

[22] I. Lee, S. Davidson, and V. Wolfe. Motivating time as a first class entity. Technical Report MS-CIS-87-54, Dept. of Comp. and Infor. Science, Univ. of Penn, Aug. (Revised Oct.) 1987.

[23] X. Liu and E. A. Lee. CPO semantics of timed interactive actor networks. *Theoretical Computer Science*, 409(1):110–125, 2008.

[24] Z. Manna and A. Pnueli. Verifying hybrid systems. *Hybrid Systems*, pages 4–35, 1992.

[25] A. K. Mok. Annotating ada for real-time program synthesis. In *IEEE Conference on Computer Assurance (COMPASS)*. IEEE, 1987.

[26] OMG. UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE), 2008.

[27] G. Pardo-Castellote. Omg data-distribution service: Architectural overview. In *ICDCS*, pages 200–206, 2003.

[28] J. L. Paunicka, D. E. Corman, and B. R. Mendel. A CORBA-based middleware solution for UAVs. In *Fourth International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 261 – 267, Magdeburg, Germany, 2001. IEEE.

[29] P. Ramadge and W. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[30] D. C. Schmidt, D. L. Levine, and S. Mungee. The design of the TAO real-time object request broker. *Computer Communications*, 21(4), 1998.

[31] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture - Patterns for Concurrent and Networked Objects*. Wiley, 2000.

[32] S. A. Seshia and A. Rakhlin. Game-theoretic timing analysis. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 575–582. IEEE Press, 2008.

[33] J. A. Stankovic, I. Lee, A. Mok, and R. Rajkumar. Opportunities and obligations for physical computing systems. *Computer*, pages 23–31, 2005.

[34] L. Thiele, E. Wandeler, and N. Stoimenov. Real-time interfaces for composing real-time systems. In *EMSOFT*, Seoul, Korea, 2006. ACM Press.

[35] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenstr. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):1–53, 2008.

[36] B. P. Zeigler, H. Praehofer, and T. G. Kim. *Theory of Modeling and Simulation*. Academic Press, 2nd edition, 2000.

[37] 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)*, Bellevue, WA, USA, 2007. IEEE.

[38] G. Zhou, M.-K. Leung, and E. A. Lee. A code generation framework for actor-oriented models with partial evaluation. In Y.-H. L. et al., editor, *Internation Conference on Embedded Software and Systems (ICESS)*, volume LNCS 4523, page 786799, Daegu, Korea, 2007. Springer-Verlag.

[39] Y. Zhou and E. A. Lee. Causality interfaces for actor networks. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):1–35, 2008.