CPS Foundations *

Edward A. Lee
EECS Department
University of California, Berkeley
Berkeley, CA, USA
eal@eecs.berkeley.edu

ABSTRACT

This paper argues that cyber-physical systems present a substantial intellectual challenge that requires changes in both theories of computation and dynamical systems theory. The CPS problem is not the *union* of cyber and physical problems, but rather their *intersection*, and as such it demands models that embrace both. Two complementary approaches are identified: *cyberizing the physical* (CtP) means to endow physical subsystems with cyber-like abstractions and interfaces; and *physicalizing the cyber* (PtC) means to endow software and network components with abstractions and interfaces that represent their dynamics in time.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: *real-time and embedded systems*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*cyber-physical systems*; I.6.5 [**Simultation and Modeling**]: Model Development[models cyber-physical interactions]; J.7 [**Computers in Other Systems**][computer-based systems]

General Terms

Design, Theory

Keywords

Cyber-physical systems, embedded systems

*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 and AF-TRUST #FA9550-06-1-0244), the Air Force Research Lab (AFRL), the Multiscale Systems Center (MuSyc) and the following companies: Agilent, Bosch, National Instruments, Thales, and Toyota.

DAC'10, June 13-18, 2010, Anaheim, California, USA.

1. INTRODUCTION

Cyber-Physical Systems (CPS) are integrations of computation with physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa. As an intellectual challenge, CPS is about the *intersection*, not the union, of the physical and the cyber.

In the physical world, a central property of a system is its dynamics, the evolution of its state over time. In the cyber world, dynamics is reduced to *sequences* of state changes without temporal semantics. The intellectual heart of CPS is in studying the joint dynamics of physical processes, software, and networks.

Applications of CPS arguably have the potential to dwarf the 20-th century IT revolution. 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, manufacturing, and smart structures. It is easy to envision new capabilities, such as distributed micro power generation coupled into the power grid, where timing precision and security issues loom large. Transportation systems could benefit considerably from better embedded intelligence in automobiles, which could improve safety and efficiency. Networked autonomous vehicles could dramatically enhance the effectiveness of our military and could offer substantially more effective disaster recovery techniques. Networked building control systems (such as HVAC and lighting) could significantly improve energy efficiency and demand variability, reducing our dependence on fossil fuels and our greenhouse gas emissions. In communications, cognitive radio could benefit enormously from distributed consensus about available bandwidth and from distributed control technologies. Financial networks could be dramatically changed by precision timing. Large scale services systems leveraging RFID and other technologies for tracking of goods and services could acquire the nature of distributed real-time control systems. Distributed real-time games that integrate sensors and actuators could change the (relatively passive) nature of on-line social interactions. The economic impact of any of these applications would be huge.

One approach to modeling and designing cyber-physical systems is to "cyberize the physical" (CtP), which means wrapping software abstractions around physical subsystems.

An early illustration of this approach wraps the physical measurements of a sensor network in database abstractions [25].

A complementary approach is to "physicalize the cyber" (PtC), which means to endow software and networking components with abstractions suitable for physical subsystems. The challenge is to endow software and network components with explicit temporal semantics.

Sensor networks gather information, whereas cyber-physical systems include both sensors and actuators with closed-loop interactions. When closed-loop systems are considered, both the PtC and CtP approaches get more challenging. A key issue is that the formal models that we use for understanding physical dynamics (ODEs, DAEs, frequency-domain techniques, LTI systems, difference equations, PDEs, etc.) do not model well the behavior of software and networks. On the other hand, prevailing cyber abstractions (procedures, objects, functions, state machines, packets, etc.) lack temporal dynamics. Once the cyber components have well-characterized temporal dynamics, their interaction with physical processes will be much easier to understand. We will show below that changes to models of physical dynamics and to the models of software and networks can bring them closer.

Engineers today do successfully design cyber-physical systems. As the complexity of these systems increases, however, our inability to rigorously model the interactions between the physical and the cyber sides creates serious vulnerabilities. Systems become unsafe, with disastrous inexplicable failures that could not have been predicted. Engineers today are stuck 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.

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 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, more complex, and more intelligent applications, the problems and risks are going to get worse. Embedded systems will no longer be black boxes, designed once and immutable in the field. Instead, they will be pieces of a larger system, a dance of electronics, networking, and physical processes.

A typical cyber-physical system will have a 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 [8].

2. CYBERIZING THE PHYSICAL

The challenge of integrating computing and physical processes has been recognized for some time, motivating the emergence of hybrid systems theories [26]. Progress in that area, however, remains limited to relatively simple systems combining ordinary differential equations with automata. These models inherit from control theory a uniform notion of time, an oracle called t available simultaneously in all parts of the system. Even adaptations to distributed control problems do this. Olfati-Saber et al. [29], for example, translate consensus problems from computer science into control systems formulations, showing connections between such consensus problems and a variety of dynamical systems problems such as synchronization of coupled oscillators, flocking, formation control, and distributed sensor fusion. These formulations, however, break down without the uniform notion of time that governs the dynamics. In networked software implementations, such a uniform notion of time cannot be precisely realized. Time triggered networks [17] and time synchronization [13] can be used to *approximate* a uniform model of time, but the analysis of the dynamics has to include the imperfections. Perfect time synchronization is not physically realizable.

In addition to the assumption of a shared global notion of time, most modeling techniques for physical systems share a second limitation that makes it difficult to integrate these models with cyber components. Namely, the model of time itself does not mesh well with the cyber world. Specifically, in widely used models of physical systems, time is represented as either a continuum, using real numbers, or as discrete time, using integers. In the cyber world, neither of these choices is suitable to describe the behavior of systems. Software, for example, represents sequences of causally related actions that have no semantic notion of time passing. It becomes very difficult to relate a statement like "current time is t" with any property of the software execution.

A model of time that supports both a time continuum and sequences of untimed causally-related actions is called superdense time [27]. Superdense time is a tuple (t, n), where $t \in \mathbb{R}$ is a real number (or an integer representing discrete time), and $n \in \mathbb{N}$ is a natural number (a non-negative integer). The natural number n admits ordered sequences of causally related events without time passing. In particular, whereas in standard models of physical dynamics a signal may have the form

$$x\colon \mathbb{R} \to \mathbb{R},$$

with a superdense model of time a signal has the form

$$x\colon \mathbb{R} \times \mathbb{N} \to \mathbb{R} \ .$$

This means that at a particular time $t \in \mathbb{R}$, x may have several values with a well-defined order, $x(t, 0)$, $x(t, 1)$, etc.

Moreover, if we augment the model of a signal to include a symbol ε denoting *absence* of a value,

$$x\colon \mathbb{R} \times \mathbb{N} \to \mathbb{R} \cup \{\varepsilon\} \ ,$$

then it becomes possible to model ordered discrete events (with or without the passage of time) together with the usual continuous or piecewise continuous signals used in standard models of physical dynamics. Such a framework offers a much richer formulation suitable for cyber-physical systems.

Superdense time is useful in blending synchronous/reactive models such as those used in certain safety-critical embedded software design [2] with discrete-event models like those in PTIDES (discussed below) and hybrid systems models that include models of continuous

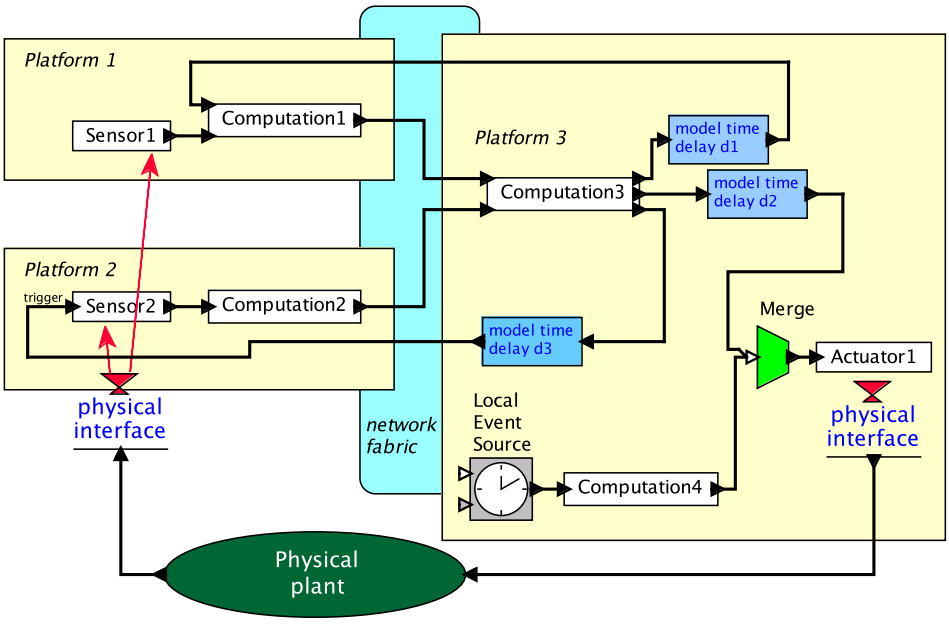

Figure 1: Example structure of a cyber-physical system.

dynamics [21, 22]. This enables comprehensive models of cyber-physical system designs where the software portions are specifications amenable to code generation for so-called "correct-by-construction" design.

3. PHYSICALIZING THE CYBER

The foundations of computing, rooted in Turing, Church, and von Neumann, are about the transformation of data, not about physical dynamics. For CPS we need to rethink the core abstractions if we really want to integrate computing with physical processes. In particular, the passage of time must become a central property. 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 and unpredictability in the timing. 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.

But surely the "right time" is expecting too much, the reader may object. The physical world is neither precise nor reliable, so why should we demand this of computing systems? Instead, we must make systems robust and adaptive, building reliable systems out of unreliable components. While I agree that systems need to be designed to be robust, we should not discard the reliability we have. Electronics technology is astonishingly precise and reliable, more than any other human invention. We routinely deliver circuits that perform a logical function essentially perfectly, on time, billions of times per second, for years. Shouldn't we exploit this remarkable achievement?

An interesting CPS challenge problem in scientific instrumentation is provided by the Large Hadron Collider (LHC). The LHC requires instrumentation and control systems structured like that in fig. 1 spread over 27 km. The challenge is that control and measurement of extremely sensitive and elusive physical processes must be carried out by networked computers. It is an extremely challenging CPS problem.

CERN and others have developed a technology called White Rabbit [11] to support distributed feedback control, distributed direct signal synthesis, and a "distributed oscilloscpe" for controlling and observing physical processes throughout the accelerator. To accomplish this, White Rabbit realizes a networked precision time protocol (PTP) based on IEEE 1588 [12, 13, 7] and synchronous Ethernet that reportedly achieves clock concurrence with a standard deviation of 80 picoseconds. CERN reports that two real-time clocks in computers up to 14 km apart can agree on the current time of day to within about 80 ps. This remarkable achievement enables specification of distributed cyber behaviors that are orchestrated with astonishing timing precision. Such clock concurrence demands new design techniques for distributed software.

Today's programming models and networking abstraction cannot directly address such distributed orchestration. CPS needs to provide technologies for *robust* and *predictable* designs with *repeatable* temporal dynamics (for a detailed discussion of the meanings of these terms, see [20]). This will necessarily build on a rigorous formal model that reflects the realities of distributed systems. The result will be CPS 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. Timing must become central to distributed software design.

4. THE PTIDES APPROACH

My research group has previously proposed and continues to work on 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 [31, 3, 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 [24, 18], and that has been shown to integrate well with models of continuous dynamics [22]. 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. [5] describe a simulator for PTIDES built on Ptolemy II [9], 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. [10] describe a strategy for incorporating fault-tolerance principles into PTIDES using backtracking techniques.

PTIDES relies on two key technology assumptions. First, clocks on a network must be synchronized with some known bound on the error [13]. Second, transport latencies on the network must be bounded. Fortunately, technologies are deployed today that satisfy these assumptions, and not just in cutting-edge research platforms like the LHC.

An interesting example is General Electric's Mark^{TM}VIe Control Platform, which has integrated high-precision network time synchronization based on IEEE 1588 into its I/O 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 CPS applications. This has in fact historically forced deployments of these applications to use specialized networking techniques, such as time-triggered architectures [17], 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), it is not hard to imagine modeling and design techniques for CPS that embrace the integration of physical dynamics with the dynamics of software and networks. PTIDES is a promising illustration of such a modeling framework, though I am certain it is not the only one that will prove interesting. The principle it follows is that software and network components have abstractions that makes them look and feel like physical components. They react to stimulus in time and produce reactions with or without latency in time. The ability to produce reactions without latency in time is essential to accurately embrace cyber abstractions and does not undermine the rigorous semantics because of the foundation using superdense time. PTIDES physicalizes the cyber.

5. DISCUSSION

Real-time software is not a new problem. Neither is real-time networking, where the phrase "quality of service" (QoS) has been used for the practice of controlling timing. However, CPS requires that timing be a *correctness* criterion, not a quality factor. Time must be part of the semantics. Moreover, recent trends have drastically changed the landscape making this perspective more realistically applicable. Model-based design [14], for example, has caught on in industrial practice, through the use of tools such as Simulink, Real-Time Workshop, DSpace, LabVIEW, and SCADE. Domain-specific modeling languages (DSMLs) are increasingly being used because they tend to support higher-level abstractions than general-purpose modeling languages such as UML. DSMLs have been recently introduced to represent timing behavior in the automotive initiative AUTOSAR for component software architectures. Also, OMG has recently extended UML with a profile called MARTE (Modeling and Analysis of Real-Time and Embedded Systems).

In addition, while computer architecture has traditionally focused on improving average-case performance, there has been a recent work on precision timed (PRET) machines that offers new opportunities to improve temporal precision [6]. This could make the temporal analysis of cyber 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 many distributed real-time software approaches 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 [16] 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. These languages offer concurrency models that are much more understandable and analyzable than those prevalent in software engineering, based for instance on threads [19]. Recent advances in execution time estimation of software [35] also help, but several obstacles remain [15]. Most particularly, these techniques are brittle 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 [33] promise to ameliorate this problem.

At a higher level, the real-time interfaces of [34] offer possibilities for component architectures for software that include temporal dynamics. Older contributions, such as the taxonomy of timing properties that must be expressible given in [23] and the annotations on untimed languages given in [28], could also provide a framework for timed 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, and hence do not solve the CPS problems by themselves. More recent innovations in middleware technologies, such as ACE/TAO [32], support real-time scheduling concepts, and have caught on in certain communities (such as avionics) [30]. The Data Distribution Service, a recent OMG standard, supports over 20 configurable Quality of Service options. These mechanisms, however, have a very ad-hoc flavor, and seem to insist on a prototype-and-test style of design.

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 non-determinism in implementations. The same standard-compliant application running in two different implementations of the runtime environment may result in different behaviors [4].

In summary, although there is a great deal of prior work to draw upon, a concerted effort that directly confronts the CPS challenges by rebuilding the foundations has the best promise for the future. The goal must be to enable harnessing technology for the betterment of society. Our inability to manage the increasing complexity of cyber-physical systems is an unnecessary obstacle.

6. ACKNOWLEDGEMENTS

I thank the following people for ideas that I have included in this paper: David Culler, John Eidson, Rajesh Gupta, Sanjit Seshia, and Stavros Tripakis.

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

[4] 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.

[5] 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.

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

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

[8] J. C. Eidson, E. A. Lee, S. Matic, S. A. Seshia, and J. Zou. Time-centric models for designing embedded cyber-physical systems. Technical Report UCB/EECS-2009-135, EECS Department, University of California, Berkeley, October 9 2009.

[9] 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.

[10] 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.

[11] G. Gaderer, P. Loschmidt, E. G. Cota, J. H. Lewis, J. Serrano, M. Cattin, P. Alvarez, P. M. Oliveira Fernandes Moreira, T. Wlostowski, J. Dedic, C. Prados, M. Kreider, R.Baer, S.Rauch, and T.Fleck. The white rabbit project. In *Int. Conf. on Accelerator and Large Experimental Physics Control Systems*, Kobe, Japan, 2009.

[12] IEEE Instrumentation and Measurement Society. 1588: IEEE standard for a precision clock synchronization protocol for networked measurement and control systems. Standard specification, IEEE, November 8 2002.

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

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

[15] 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.

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

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

[18] E. A. Lee. Modeling concurrent real-time processes

using discrete events. *Annals of Software Engineering*, 7:25–45, 1999.

[19] E. A. Lee. The problem with threads. *Computer*, 39(5):33–42, 2006.

[20] E. A. Lee. Computing needs time. *Communications of the ACM*, 52(5):70–79, 2009.

[21] E. A. Lee and H. Zheng. Operational semantics of hybrid systems. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume LNCS 3414, pages pp. 25–53, Zurich, Switzerland, 2005. Springer-Verlag.

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

[23] 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.

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

[25] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. In *ACM Symposium on Operating System Design and Implementation (OSDI)*, 2002.

[26] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In *Real-Time: Theory and Practice, REX Workshop*, pages 447–484. Springer-Verlag, 1992.

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

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

[29] R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and cooperation in networkedmulti-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

[30] 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.

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

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

[33] 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.

[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.