TIMED DISTRIBUTED SYSTEMS

Edward A. Lee, Principal Investigator
EECS Department, UC Berkeley
Berkeley, CA 94720 USA
eal@eecs.berkeley.edu

Final Report 2005-06, Micro Project No. 05-055
Industrial Sponsors: Agilent, Hewlett-Packard

Abstract

This project seeks models of computation, software techniques, and analytical models for distributed timed systems. Such systems coordinate components on a network, and timeliness matters. Applications include industrial automation, instrumentation systems, and networked embedded software systems. The introduction of network time synchronization such as IEEE 1588 makes possible time coherence that has not traditionally been part of the computational models. Given time synchronization with some known precision, we believe that distributed applications should be designed and developed differently, and that time synchronization can help with robust coordination of unreliable components. Existing methods such as real-time operating systems, time-triggered networks, and network time synchronization deal with parts of the problem. These technologies, however, are used with relatively conventional concurrency models (threads and processes). This project seeks to elevate timing and distribution to the level of the programmers model, so that applications are built by directly expressing timing and distribution properties. The objective is a framework for designing deployable timed distributed systems. Here we report on the outcomes during the first year of the project.

1 Introduction

Distributed timed systems are computer-based systems where multiple computers are connected on a network and interact with sensors and actuators. Applications include manufacturing, instrumentation, surveillance, multi-vehicle control, avionics systems, automotive systems and scientific experiments. Since each computer interacts with physical processes, the passage of time becomes a central feature; it is this key constraint that distinguishes these systems from distributed computing in general.

In addition to interacting over a communication network, the nodes in a distributed embedded system interact through the physical world. Driving an actuator at one node, for example, may affect the data sensed at another node. Moreover, actuation may need to be orchestrated across nodes. The required precision of that orchestration, of course, depends on the application. Robotic applications, e.g. in manufacturing, may require precisions on the order of milliseconds. Instrumentation, where stimuli are generated and responses are measured, may require precisions on the order of nanoseconds or even higher. The question we address in this project is how to construct the distributed software for such systems.

General-purpose distributed software is dominated by distributed object-oriented programming [22] using frameworks such as CORBA, SOAP, DCOM, EJB, and XML Web Services. Some extensions of these frameworks, such as ACE/TAO [21], support real-time scheduling concepts, and have caught on in certain communities (such as avionics) [20]. These technologies are viewed as being too heavyweight for many embedded applications such as industrial control, where software may be written in special purpose languages (e.g. based on the International Electrotechnical Commission's IEC 61131) and executed on special purpose hardware called Programmable Logic Controllers (PLCs). Extensions of these techniques to distributed control systems (e.g. IEC 61499), have not proved satisfactory, because of the non-determinism of implementation. That is, the same standard-compliant application running in two different implementations of the runtime environment may result in different behaviors [2].

Our approach to the nondeterminism challenge in constructing distributed real-time system is to rely on network time synchronization [11], where the computing nodes on the network share a common notion of time to a known precision. This has the potential for being lightweight and delivering repeatable and predictable behaviors at a variety of timing precisions.

Network time synchronization is available on a variety of platforms, including standard computers on the Internet (e.g. NTP [19]), time-triggered buses such as TTA or FlexRay [12], TCP/IP over Ethernet (e.g. IEEE 1588), and wireless networks (e.g. RBS [23]). Implementations of IEEE 1588 have demonstrated time synchronization as precise as tens of nanoseconds over networks that stretch over hundreds of meters, more than adequate for many manufacturing and instrumentation systems. Such precise time synchronization enables coordinated actions over distances that are large enough that fundamental limits (the speed of light, for example) make it impossible to achieve the same coordination by conventional stimulus-response or client/server mechanisms.

Our approach in this project builds on discrete-event (DE) modeling techniques [1, 14, 24]. DE models are concurrent compositions of components that interact via *events*. An event is a time-stamped value, where time is "logical time" or "model time" [13]. Correct execution of such models requires only that the ordering of time stamps be respected. DE is usually a *simulation* technology (e.g. in hardware description languages such as Verilog and VHDL and network modeling languages such as OPNET Modeler[1] and Ns-2[2]). When DE models are executed on distributed platforms, the objective is usually to accelerate simulation, not to implement distributed real-time systems [1, 6, 24].

We call our programming model PTIDES (pronounced "tides"), for Programming Temporally Integrated Distributed Embedded Systems. In our approach, DE is not a simulation technology, but rather application specification language, which serves as a semantic basis for obtaining determinism in distributed real-time systems. Applications are given as distributed DE models, where for certain events, their modeling time is mapped to physical time. For example, a programmer may specify that an actuator must produce a physical output at the time determined by the time stamp of an event sent to the actuator. When these models are executed in a runtime environment that ensures DE semantics, we know that the applications will have deterministic behaviors regardless of the actual implementations.

Preserving DE semantics at runtime can be challenging, since the global, consistent notion of time may lead to a total ordering of execution in a distributed system, an unnecessary waste of resources. PTIDES takes an event-driven execution strategy. Unlike many hard real-time distributed systems that depend on domain specific network architectures, our only assumption of communication behavior is that it delivers packets reliably with a known bounded delay. We divide our execution strategies into two layers: global coordination, and local resource scheduling. When receiving an event from the network, the global coordination layer determines whether the event can be processed immediately or it has to wait for other potentially preceding events. Once it is sure that the current event can be processed according to DE semantics, it hands the event over to a local resource scheduler, which may use existing real-time scheduling algorithms, such as earliest deadline first (EDF) to prioritize the processing of all pending events.

The approach in this project leverages the concept of actor-oriented design [15], borrowing ideas from Simulink and from Giotto [9]. However, it addresses a number of limitations in Simulink and Giotto by building multitasking implementations from specifications that combine dataflow modeling and distributed discrete-event modeling. In discrete-event models, components interact with one another via events that are semantically placed on a time line. Some level of agreement about time across distributed components is necessary for this model to have a coherent semantics. While distribution of discrete-event models has long been used to exploit parallel computing to accelerate execution [24], our project is not concerned with accelerating execution. The focus is instead on using a model of time as a binding coordination agent. This steers us away from conservative techniques (like Chandy and Misra [4]) and optimistic techniques (like Time Warp [10]). The use of dataflow formalisms [16] supports mixing untimed and event-triggered computation with timed and periodic computation.

2 Progress To Date

The project started Fall 2005, and we have made considerable progress. The first step was to define the computational models that we will use. We combine discrete-event (DE) models, synchronous dataflow (SDF) models, and finite state machines (FSM). This combination overcomes significant weaknesses of prior models, in particular by supporting preemtable concurrent tasks (unlike TinyOS and nesC) and aperiodic, event triggered tasks (unlike Simulink and Giotto).

The ability of TinyOS and nesC to create thin wrappers around hardware provides a simple and understandable mechanism for creating event-triggered, fine-grained, atomic reactions to external events. When these external events trigger significant computations, nesC programs will post tasks that are executed later. These tasks, however, all execute atomically with respect to one another, and hence a long-running task will block other tasks. This can create unacceptable latencies, and often forces software designers to manually divide long-running tasks into more fine-grain ones.

[1]http://opnet.com/products/modeler/home.html
[2]http://www.isi.edu/nsnam/ns

Simulink and Giotto, by contrast, freely mix long-running tasks with hard-real-time fine-grained tasks by exploiting the properties of an underlying priority-driven multitasking real-time operating system. They do this without requiring programmers to specify priorities or use mutexes or semaphores. However, these tasks are required to be periodic, and their latencies are strongly related to their periods, so they lack the event-triggered, reactive nature of nesC programs.

These two ideas can be combined within a dataflow framework with elements borrowed from DE models to specify timing properties. Dependencies within the dataflow model can be statically analyzed, and with a carefully chosen variant of dataflow that combines SDF with FSMs, called heterochronous dataflow (HDF) [8], schedulability becomes decidable and synthesis of efficient embedded software becomes possible. We believe that the resulting language will prove expressive, efficient, understandable, and analyzable.

For the DE models, we have developed a framework that we call the PTIDES [25]. PTIDES has DE semantics, but with carefully chosen relations between model time and real time. Model time and real time are related only at sensor and actuator boundaries, using complementary inequalities. Specifically, time-stamped sensor data must be delivered to actuators before real time matches the model time stamp, and time-stamp sensor data is delivered to the software after real time has passed the model time of the time stamp. Key to making this model effective is to ensure that these inequality constraints are preserved at runtime. In addition, the chronological semantics of model time is respected throughout the execution. To accomplish this, we give a distributed execution strategy that obeys DE semantics without the penalty of totally ordered executions based on time stamps. Our technique relies on having a distributed common notion of real time, known to some precision, and on causality analysis of DE models. For the causality analysis, we define relevant dependency and relevant orders to enable out-of-order execution without compromising determinism and without requiring backtracking. These concepts, adapted from causality interfaces [18], formally capture the ordering constraints of temporally ordered events that have a dependency relationship. This formal structure provides an algebra within which we can perform schedulability analysis of distributed discrete-event models given bounds on the accuracy of time synchronization (using IEEE 1588 or more loosely coupled time synchronization technologies). Details are given in [25].

We envision using the PTIDES model to provide temporal semantics. It will function as a coordination language for distributed software components. The behavior of the components themselves can be given in a low-level programming language, such as C, but more usefully would also be specified using a model-based mechanism. We will be leveraging a code generation framework using the Ptolemy II framework [5] that we have built, described in [26], that synthesizes C code from HDF models. We are building a small runtime kernel that uses the PTIDES semantics to coordinate components synthesized by this code generator.

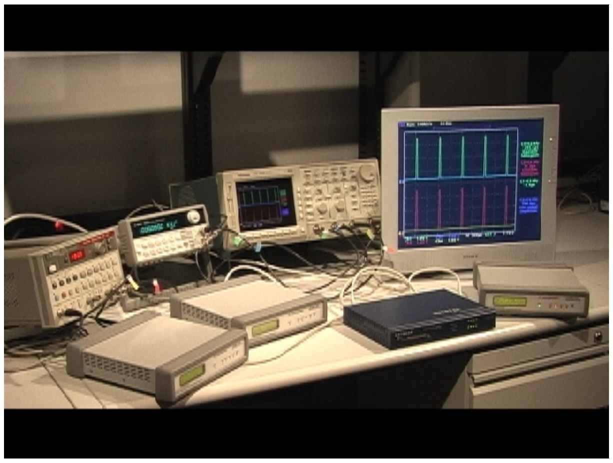

Figure 1. Time Synchronized Network Lab.

In the last year, we have also established a lab setup, shown in figure 1, for experimenting with real-time distributed software. In January of 2006, we obtained from Agilent, our industrial partner, three prototype implementations of networked embedded computers implementing IEEE 1588 time synchronization. These P1000 systems include digital I/O that can time-stamp incoming events with a (distributed) precision of about 10 ns, and can generate digital output signals with the same precision. The systems include a power-PC computer running embedded Linux. We have instrumented the experimental setup and made careful measurements of the real-time behaviors. The next phase is to integrate our code generation to build non-trivial applications on the distributed setup.

2.1 Importance

Existing methods for addressing real-time computation typically deal with a portion of the problem of constructing and executing real-time programs. Real-time operating systems (RTOSs) provide mechanisms for prioritizing tasks and triggering computations in response to timer or event interrupts. Time-triggered networking techniques such as the Time Triggered Architecture (TTA) provide deterministic sharing of networking resources and insulation from faults. Network time synchronization protocols such as NTP and IEEE 1588 provide a common time base across computers on a network [11]. All of these technologies, however, are used with relatively conventional concurrency models (threads and processes) and conventional programming languages. This project elevates timing and distribution to the level of the programmer's model, so that applications are built by directly expressing timing and distribution properties [17].

Our use of DE models as a distributed programming model for real-time software builds on distributed discrete-event simulation, which has a rich history [7]. So-called "conservative" techniques advance model time to t only when each node can be assured that it has seen all events time stamped t or earlier. In the well-known Chandy and Misra technique [3], extra (null) messages are used for one execution node to notify another that there are no such earlier events. For our purposes, this technique binds the execution at the nodes too tightly, making it very difficult to meet realistic real-time constraints. So-called "optimistic" techniques perform speculative execution and backtrack if and when the speculation was incorrect [10]. Such optimistic techniques will also not work in our context, since backtracking physical interactions is not possible.

Our approach is conservative, in the sense that events are processed only when we are sure it is safe to do so. But we achieve significantly looser coupling than Chandy and Misra using relevant dependency analysis.

References

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

[2] 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.

[3] K. M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Trans. on Software Engineering*, 5(5), 1979.

[4] K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison Wesley, 1988.

[5] 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.

[6] G. S. Fishman. *Discrete-Event Simulation: Modeling, Programming, and Analysis*. Springer-Verlag, 2001.

[7] R. Fujimoto. *Parallel and Distributed Simulation Systems*. John Wiley and Sons, 2000.

[8] A. Girault, B. Lee, and E. A. Lee. Hierarchical finite state machines with multiple concurrency models. *IEEE Transactions On Computer-aided Design Of Integrated Circuits And Systems*, 18(6):742–760, 1999.

[9] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: A time-triggered language for embedded programming. In *EMSOFT 2001*, volume LNCS 2211, Tahoe City, CA, 2001. Springer-Verlag.

[10] D. Jefferson. Virtual time. *ACM Trans. Programming Languages and Systems*, 7(3):404–425, 1985.

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

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

[13] L. Lamport, R. Shostak, and M. Pease. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

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

[15] E. A. Lee, S. Neuendorffer, and M. J. Wirthlin. Actor-oriented design of embedded hardware and software systems. *Journal of Circuits, Systems, and Computers*, 12(3):231–260, 2003.

[16] E. A. Lee and T. M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, 1995.

[17] E. A. Lee and Y. Zhao. Reinventing computing for real time. Technical Report UCB/EECS-2006-83, EECS Department, University of California, Berkeley, May 30 2006.

[18] E. A. Lee, H. Zheng, and Y. Zhou. Causality interfaces and compositional causality analysis. In *Foundations of Interface Technologies (FIT), Satellite to CONCUR*, San Francisco, CA, 2005.

[19] D. L. Mills. A brief history of NTP time: confessions of an internet timekeeper. *ACM Computer Communications Review*, 33, 2003.

[20] 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.

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

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

[23] H. Wang, L. Yip, D. Maniezzo, J. Chen, R. Hudson, J. Elson, and K. Yao. A wireless time-synchronized COTS sensor platform part ii: applications to beamforming. In *IEEE CAS Workshop on Wireless Communications and Networking*, 2002.

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

[25] 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.

[26] G. Zhou, M. Leung, and E. A. Lee. A code generation framework for actor-oriented models with partial evaluation. Technical Report UCB/EECS-2007-29, University of California, EECS Department, February 23 2007.