From the Editor in Chief...

Days of Miracle and Wonder

Robert Filman • *RIACS/NASA Ames Research Center*

Disruptive technologies change how people live — how they work, what they consume, what they earn, what they do with their time. Your feeling about such technologies varies, of course, depending on whether it's your ox that is being gored.

The Internet was the disruptive technology of the nineties. Travel agents, customer-support personnel, booksellers, newspaper publishers, and countless others found the Internet making their old ways of doing business obsolete. The coming disruptive technology will be an epidemic of networks of sensors (and actuators). Devices that measure the environment, process their measurements, communicate the results, and sometimes invoke actions will be pervasive. Sensor nets will ration water, nutrients, and pesticides in agriculture; monitor and control manufacturing processes; detect and guide fire and disaster fighting; monitor structural and earthquake damage; guide autos to less-traveled roads; measure and predict the weather on Earth and other planets; route communications traffic; check and replenish inventory; monitor and optimize habitat environments; track animals; and, most invasively, monitor people's health and movements. Networks of sensors will rush an ambulance to a heart-attack victim, identify who planted the bomb in the baby carriage, and warn a rental-car company of a traffic scofflaw. Like public health, telecommunications, and the automobile before them, sensor nets will be a vehicle of social transformation.

Sensor Nets for Dummies

Of course, we are technologists, and like Tom Lehrer's Werner von Braun, where the missiles come down is someone else's department. What technologies will make developing a sensor net as easy as, for example, building an Internet application? What must we do to make installing a network of sensors as simple as creating a wireless home network?

Back in the computer dark ages, there were a large variety of networking technologies, each with its own set of operations and protocols. We still have several networking technologies, but the actual implementation of the network no longer matters to most application builders. Everyone programs to a common set of Internet protocols and, more importantly, to higher-level protocols such as HTTP, Corba, and Web services. Thus, the software task of creating pervasive sensor nets divides between developing the underlying networking apparatus and creating the high-level view of the system presented to the application programmer.

Transformative sensor nets need a variety of network mechanisms. Most prominent among them are ways of self-organizing into communication structures; manageability protocols to let network owners ensure that the sensor net is well-behaved; common techniques for accounting, security, and privacy (a grievous omission from the current set of Internet protocols); and technology to give the networks the ability to self-repair.

The sensor equivalent to the Dynamic Host Control Protocol ("I'm here, connect me in") is a critical research topic. This is particularly the case for untethered sensing devices, dispersed and likely running on battery or intermittent solar power, for which "each bit communicated is one bit closer to death." Some applications will require sensors to coordinate (for example, tracking); most applications will want the sensor net to cope locally with intermittent and indefinite faults and failures. The simple failure of a few sensors can't be allowed to lead to the death of the community.

Programming Models and Requirements

A critical element of making a technology pervasive is a common, high-level programming model — how the sensor net appears to a programmer. The choice of a programming model also pre-

 Published by the IEEE Computer Society 1089-7801/04/$20.00 © 2004 IEEE

scribes details of the pragmatic effects of actions. For example, a model might provide particular quality of service (QoS) guarantees to applications. In this scenario, too weak a QoS promise puts too great a burden on applications that need that quality; too strong a promise complicates the implementation and extracts a cost throughout the system. This is comparable to the difficulties of running streaming video over the Internet: because TCP/IP makes no promises about QoS, streaming applications have to work hard and pray to get viewable results. However, orienting the entire Internet protocol toward streaming would reduce its overall capacity and performance.

Similarly, because sensor networks will sometimes be deployed within demanding physical constraints, the programming environment might not be able to (or want to) completely shield the application programmer from issues of actual deployment, topology, power consumption, limited storage, intermittent and noisy communications, and failure. On small untethered devices, memory could be more expensive than computation, and communication vastly more expensive than memory. Just as with QoS, application-programming models need to find the right balance between abstraction and transparency.

Communication Abstractions

Internet protocols are dominated by addressable, synchronous, call–response mechanisms. An Internet request is addressed to a particular name, makes that request with parameters, and synchronously waits for and gets a response. One software architecture for sensor nets mimics the traditional Internet: sensors are addressable devices that respond to specific messages. Variants on this idea include treating distribution in terms of remote procedure calls, remote object-method invocation, or distributed shared memory. While this might prove adequate for some applications, such an architecture is limited. The World Wide Web works because there is substantial human interaction in accessing pages, including the human ability to sort through the output of natural-language-based search engines. Service discovery by automated systems on the Internet remains a very open issue, and a limitation on the use of Web services in situations in which the connection is not configured ahead of time.

Other programming models are more intriguing. Sensor networks generate many "sensor readings" or events. In practice, dealing with such a multitude of events requires filtering, combining, and abstracting (perhaps recursively) up a hierarchy of event abstractions. One approach is to treat a sensor net as a database, accessing it with SQL-like queries and assertions. The equivalent of the distributed query optimization system would then be responsible for translating the high-level query into collections of individual sensor readings. To the programmer, the result looks like a database

IC Welcomes New Editorial Board Member

Dan S. Wallach is an assistant professor in Rice University's Department of Computer Science. He has a variety of interests in computer security, including architectures to run untrusted programs (such as Java applets), improved robustness in P2P network architectures, and making electronic voting systems robust against threats. Wallach is a member of the ACM, the IEEE, and Usenix. Contact him at dwallach@cs.rice.edu.

query set. The Tiny Aggregation service (TAG)[1] and Cougar[2] are examples of this approach.

As you might guess from my last column, my current interest centers on treating sensor management with publish-and-subscribe architectures. In such an approach, applications interested in sensor results describe to an *event channel* the kinds of events they care about (subscriptions). Sensors publish events to the channel, which arranges to route interesting events to relevant subscribers. Examples of event-based sensor control systems include DSWare,[3] Directed Diffusion,[4] and nesC.[5]

Another approach is to bring the code to the sensor, as with the mobile agents seen in SensorWare[6] and Maté.[7] This implies having some communicable "scripting language" to describe what is to be done at the sensor. There can be considerable economy in bringing the processing to the data, rather than sending the data to the processing — particularly when computation is cheap, communication expensive, and the desired answer is a digest of a lot of local information.

Research Directions

Remote procedure calls, database queries, events, and agents are foundation architectures for sensor nets. On such foundations, real architecture designers need to provide linguistic and semantic mechanisms for various problems that conventional languages usually ignore. These include the ability to integrate operating context into the net's behavior, more robust mechanisms for failure recovery, primitives for controlling device management, access to the underlying cost of operations (in power, for example), synchronization, and mechanisms for self-organization.

Sensor architectures will need to deal as much with what happens as with what doesn't happen. The RFI'd product on the shelf repeatedly broadcasts, "I'm here." The noteworthy occasion is when it stops. In more complicated sensing applications, an element's track through the sensor net might be what is interesting, rather than the particular tracking events. Stale data — where the element was 10 minutes ago — might well be useless. High-level models will need to develop mechanisms for expressing and realizing such concepts.

Sensor networks are a prime area for experimentation and research. Shoot it up, and we'll see where it comes down.

References

1. S. Madden et al., "TAG: A Tiny Aggregation Service for Ad Hoc Sensor Networks," *ACM SIGOPS Operating Systems Rev.*, Winter 2002, pp. 131–146.
2. P. Bonnet, J. Gehrke, and P. Seshadri, "Querying the Physical World," *IEEE Personal Comm.*, vol. 7, no. 5, Oct. 2000, pp. 10–15.
3. S. Li, S.H. Son, and J.A. Stankovic, "Event Detection Services: Using Data Service Middleware in Distributed Sensor Networks," *Proc. Workshop Information Processing in Sensor Networks* (IPSN '03), F. Zhao and L. Guibas, eds., LNCS 2634, Springer-Verlag, 2003, pp. 502–517.
4. C. Intanagonwiwat et al., "Directed Diffusion for Wireless Sensor Networking," *IEEE/ACM Trans. Networking*, vol. 11, no. 1, 2003, pp. 2–16.
5. D. Gay et al., "The nesC Language: A Holistic Approach to Networked Embedded Systems," *Proc. ACM Conf. Programming Language Design and Implementation* (SIGPLAN '03), ACM Press, 2003, pp. 1–11.
6. A. Boulis, C.C. Han, and M.B. Srivastava, "Design and Implementation of a Framework for Efficient and Programmable Sensor Networks," *Proc. Conf. Mobile Systems* (MobiSys '03), Usenix Assoc., May 2003.
7. P. Levis and D. Culler, "Maté: A Tiny Virtual Machine for Sensor Networks," *Proc. 10th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS X), ACM Press, 2002, pp. 85–95.

How to Reach *IC*

Articles

We welcome submissions about Internet application technologies. For detailed instructions and information on peer review, see *IEEE Internet Computing*'s author guidelines at www.computer.org/internet/author.htm, or log on to *IC*'s author center at Manuscript Central (www.computer.org/mc/internet/author.htm).

Letters to the Editor

Read something in *IC* that you want to respond to? Please email letters, including a reference to the article in question and when it appeared, to internet@computer.org.

Reuse Permission