Critical Considerations and Designs for Internet-Scale, Event-Based Compositional Architectures

David S. Rosenblum[†] Alexander L. Wolf[‡] Antonio Carzaniga[‡]

[†]Dept. of Information & Computer Science
University of California, Irvine
Irvine, CA 92697-3425
dsr@ics.uci.edu

[‡]Dept. of Computer Science
University of Colorado
Boulder, CO 80309-0430
{alw,carzanig}@cs.colorado.edu

1 Introduction

A common architectural style for distributed, loosely-coupled, heterogeneous software systems is a structure based on event generation, observation and notification. A notable characteristic of an architecture based on events is that interaction among architectural components occurs asynchronously, thereby simplifying the composition of autonomous, independently-executing components that may be written in different programming languages and executing on varied hardware platforms.

There is increasing interest in deploying these kinds of distributed systems across wide-area networks such as the Internet. For instance, workflow systems for multi-national corporations, multi-site/multi-organization software development, and real-time investment analysis across world financial markets are just a few of the many applications that lend themselves to deployment on an Internet scale. However, deployment of such systems at the scale of the Internet imposes new challenges that are not met by existing technology.

In particular, the technology to support an event-based architectural style is well-developed for local-area networks (e.g., Field's Msg [7], SoftBench's BMS [1], ToolTalk [3] and Yeast [4]), but not for wide-area networks. One of these systems, Yeast, was built and studied by the first author while he was on the research staff at AT&T Bell Laboratories. Yeast is a general-purpose platform for building distributed applications in an event-based architectural style, and it supports event-based interaction quite naturally within local-area networks. However, its centralized-server architecture limits its scalability to wide-area networks, as does its limited support with respect to certain issues that are more important for wide-area networks than for local-area networks, such as naming and security. The experience with Yeast clearly demonstrates that these existing technologies are ill-suited to networks on the scale of the Internet, and that new technologies are needed to support the construction of large-scale, event-based software systems for the Internet.

We have been studying the problem of designing an Internet-scale event observation and notification facility that can serve as a platform for building wide-area distributed systems according to an event-based architectural style [8]. In this paper we briefly outline our achievements to date. In Section 2, we define more precisely what we mean by the notion of "Internet scale". In Section 3, we describe our design framework for an event observation and notification facility; this framework identifies a spectrum of design choices, which are organized according to seven models. In Section 4 we evaluate one existing technology, the CORBA Event Service, with respect to this design framework. We conclude in Section 5 with a discussion of our current work, which focuses on the design and analysis of architectures and algorithms for Internet-scale event notification.

2 Attributes of Internet Scale

The primary distinguishing characteristics of an Internet-scale computer network are the *vast numbers of computers* in the network and the *vast numbers of users* of these computers. An important related characteristic is the worldwide *geographical dispersion* of the computers and their users. As a consequence of geographical dispersion, it becomes necessary to address relativistic issues in multiple observations of the same event. For

This material is based upon work supported by the National Science Foundation under Grant No. CCR-9701973. This effort was also sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-98-1-0061. This work was also supported in part by the Air Force Material Command, Rome Laboratory, and the Advanced Research Projects Agency under Contract Number F30602-94-C-0253. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Advanced Research Projects Agency, Rome Laboratory, the Air Force Office of Scientific Research or the U.S. Government.

instance, observers of two events occurring on opposite sides of the world may observe two different orders for those events. Additionally, an application requesting a notification about an event at roughly the same time as, but prior to, the occurrence of the event of interest may or may not be notified about the event.

At the scale of the Internet, the vast numbers of geographically-dispersed computers and users also have a much greater degree of *autonomy* than in local-area networks. Because of this autonomy, issues of resource usage are of greater concern, such as accounting for resource usage for observation and notification computations, placing limits on resource usage, and preventing misuse of resources or intrusiveness on others' usage of resources.

Related to the issue of autonomy is the *security* of the computers and users. Mechanisms and policies must be established that will allow Internet-scale event observation and notification to take place in a manner that is compatible with security mechanisms such as firewalls, and is consistent with the need to enforce access permissions and other protection mechanisms.

Finally, concerns related to *quality of service* obtain much greater visibility at the scale of the Internet. Because of network latencies, outages and other dynamically-varying network phenomena, an Internet-scale event observation and notification facility will have to cope with decreased reliability of observations and notifications, as well as decreased stability of the entities to be observed and notified.

As a consequence of Internet scale, it would be infeasible to employ many kinds of low-level mechanisms that are used to support event observation and notification in a local-area network, such as *broadcast mechanisms* and *vector clocks*. Broadcast mechanisms indiscriminately communicate event occurrences and notifications to all machines on a local network. Vector clocks involve piggybacking a vector timestamp onto each message exchanged between the communicating processes of an application, in order to aid the identification of causally-related events; the size of the timestamp is linear in the total number of processes in the application.

3 Design Framework

Implicit in event observation and notification is a timeline of basic activities, which occur in sequence:

1. *determination* of which events will be made observable;
2. *expression of interest* in an event or pattern of events;
3. *occurrence* of each event;
4. *observation* of each event that occurred;
5. *relation* of the observation to other observations to recognize the event pattern of interest;
6. *notification* of an application that its pattern of interest has occurred;
7. *receipt* of the notification by the application; and
8. *response* of the application to the notification.

To account for these activities, our design framework for Internet-scale observation and notification is organized around the following seven models, each of which focuses on a different domain of concern in the design:

1. an *object model*, which characterizes the components that generate events and the components that receive notifications about events;
2. an *event model*, which provides a precise characterization of the phenomenon of an event;
3. a *naming model*, which defines how components refer to other components and the events generated by other components, for the purpose of expressing interest in event notifications;
4. an *observation model*, which defines the mechanisms by which event occurrences are observed and related;
5. a *time model*, which concerns the temporal and causal relationships between events and notifications;
6. a *notification model*, which defines the mechanisms that components use to express interest in and receive notifications; and
7. a *resource model*, which defines where in the Internet the observation and notification computations are located, and how resources for the computations are allocated and accounted.

Each of these models has a number of possible realizations, which together define a seven-dimensional design space for Internet-scale event observation and notification facilities. Of course, these dimensions are not completely independent, because the models are interrelated in various ways. Because of these interrelationships, only a proper subset of the points in this space will correspond to adequate designs for Internet-scale facilities.

4 Evaluation of the CORBA Event Service

The Common Object Request Broker Architecture (CORBA) is a general-purpose, Internet-scale software architecture for component-based construction of distributed systems using the object-oriented paradigm [5,9]. The CORBA specification includes specifications for a number of Common Object Services, one of which is the CORBA Event Service [6]. The CORBA Event Service defines a set of interfaces that provide a way for objects to synchronously communicate event messages to each other. The interfaces support a *pull* style of communication (in which the *consumer* requests event messages from the *supplier* of the message) and a *push* style of communication (in which the supplier initiates the communication). Additional interfaces define *channels*, which act as buffers and multicast distribution points between suppliers and consumers. The TINA Notification Service is a similar service defined on top of the CORBA Event Service [10].

The CORBA Event Service lacks support for many aspects of event observation and notification defined in Section 3. The object model is the object model of CORBA, and an event is simply a message that one object communicates to another object as a parameter of some interface method. The specification of the CORBA Event Service does not define the content of an event message, so objects must be pre-programmed with "knowledge" about the particular event message structure that is to be shared between communicating suppliers and consumers. Given this view of events, a naming mechanism is unnecessary, as is an observation mechanism, and any attempt to identify patterns of events is the responsibility of the consumers of event messages. Timestamps can be associated with events, but the meaning of such timestamps is at the discretion of the objects exchanging the event messages. Being a message, an event is its own notification. Computational and other resource-related aspects of events are subsumed by those of CORBA as a whole.

In summary, an event as defined by the CORBA Event Service is nothing more than a parameter in a standard CORBA method invocation, with options available for multicasting and buffering message parameters. The application programmer wanting to use this service is still faced with the problem of how to locate events of interest, how to advertise new kinds of events, how to match patterns of events, and how to create and maintain networks of event channels to perform matching. Thus, the CORBA Event Service provides only a small subset of the capabilities needed in an Internet-scale event observation and notification facility.

5 Current Work

We have begun to design an improved event observation and notification facility. It is clear, based on our experience, that simply trying to scale a design intended for a local-area network is a flawed approach. The first step is therefore to formulate new architectures and related algorithms for event notification that are scalable to the Internet. Starting by adapting known Internet-scale architectures, such as that of Network News and the Domain Name Service, we simulate event observation and notification performance behavior in a variety of wide-area network scenarios. To date, four different architectures have been studied together with nine algorithms that implement both the recognition of event sequence patterns and the delivery of event notifications. The simulation is carried out by means of a network simulator that we have implemented. Our simulator allows us to configure a network model, and to define the event facility components, event generators, and event consumers. The simulator also accounts for computation and network resource usage on a per-host, per-process, and global-network basis.

Our initial target for the event observation and notification facility is the Software Dock [2], which is an agent-based system we are developing for Internet-scale distributed configuration management and deployment. The architecture of the Software Dock consists of *release docks*, representing producer sites, and *field docks*, representing consumer sites. The docks communicate through an event facility. The Software Dock is therefore a prototypical instance of a distributed compositional architecture. As one simple example of how the Software Dock would use an event facility, consider what happens when the producer of a system releases a bug fix. This would generate an event that results in notifications being sent to consumer sites interested in bug fixes for that system. The notification triggers an orchestration of agent activities that configure, retrieve, and install the fix.

As we gain experience in designing and constructing an Internet-scale event observation and notification facility, we will refine the models to incorporate lessons learned from our experience. A number of these refinements will likely be made to the observation and notification models, whose realizations will require careful engineering to ensure efficient and reliable operation on an Internet scale. Such refinements might involve the definition of a formal calculus of event operations that would support systematic optimization of the configuration of a network of observers, much in the same way that optimizations are applied to relational database queries in query languages such as SQL. Some operations that the calculus could support include generation, filtering, observation, notification, advertising, publication, subscription and reception.

References

[1] C. Gerety, “HP SoftBench: A New Generation of Software Development Tools”, *Hewlett-Packard Journal*, vol. 41, no. 3, pp. 48–59, 1990.

[2] R.S. Hall, D. Heimbigner, A.v.d. Hoek, and A.L. Wolf, “An Architecture for Post-Development Configuration Management in a Wide-Area Network”, *Proc. 1997 International Conference on Distributed Computing Systems*, Baltimore, MD, pp. 269–278, 1997.

[3] A.M. Julienne and B. Holtz, *ToolTalk and Open Protocols: Inter-Application Communication*: Prentice Hall, 1994.

[4] B. Krishnamurthy and D.S. Rosenblum, “Yeast: A General Purpose Event-Action System”, *IEEE Transactions on Software Engineering*, vol. 21, no. 10, pp. 845–857, 1995.

[5] Object Management Group, “The Common Object Request Broker: Architecture and Specification”, revision 2.0, July 1995.

[6] Object Management Group, “CORBAservices: Common Object Services Specification”, formal/97-07-04, July 1997.

[7] S.P. Reiss, “Connecting Tools Using Message Passing in the Field Environment”, *IEEE Software*, vol. 7, no. 4, pp. 57–66, 1990.

[8] D.S. Rosenblum and A.L. Wolf, “A Design Framework for Internet-Scale Event Observation and *Proc. 6th European Software Engineering Conference/5th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Zurich, Switzerland, pp. 344–360, 1997.

[9] J. Siegel, *CORBA Fundamentals and Programming*. New York, NY: Wiley, 1996.

[10] Telecommunications Information Networking Architecture Consortium, “TINA Notification Service , July 1996.