The Event Desktop: Supporting Event-enabled Clients on the Web

Richard S. Hall

Freie Universität Berlin
Institute of Computer Science
Takustraße 9, 14195 Berlin
GERMANY
rickhall@inf.fu-berlin.de

Abstract. The quantity of information available on the World Wide Web is staggering. The average computer user is unable to stay abreast of changes and newly available resources. The main deterrence to keeping users informed is that the Web is still largely a passive medium and only ad-hoc mechanisms for notifying users are possible. This paper assumes that wide-area event notification is the natural missing component to alleviate this issue. Based on this assumption, this paper introduces the *Event Desktop* infrastructure. The Event Desktop is built on top of a wide-area event notification system and intends to simplify event client creation, dissemination, and management. The combination of the Event Desktop and a wide-area event notification system creates a platform for experimentation on the next generation of active Web applications.

1 Introduction

The integration of the Web and the file system, as witnessed in Windows 98™, blurs the distinction of local and remote information. For the user, the illusion is created that the Web is nothing more than a huge, globally shared hard disk; this is an appealing and comfortable analogy. This analogy is only partially accurate when the true nature of the Web is taken into consideration. The Web represents a storage capacity that is unparalleled by any other storage mechanism. As a result, there is little hope for an intuitive path and directory structure, such as is available for file systems. Additionally, interrelationships among information on the Web are unprecedented. Lastly, due to the number of participants the Web changes every second of every day.

Despite these differences, "hard disk" management techniques are still used to navigate the Web's information space, such as with search engines and short-cuts (i.e., bookmarks). One reason for lack of advancement is because the Web has remained a largely passive medium. In 1996, "push" technologies, such as Marimba's Castanet [10] and Pointcast [6], received much attention by proclaiming that they would make the Web active and bring a customized view to each computer user, but these promises did not materialize.

There are at least three reasons why "push" technologies failed. One, "push" technologies focused on delivering content to end-users and this ignores the important "global storage" aspect of the Web. Second, "push" technologies focused too narrowly on providing their own content or creating strategic alliances with content providers. The first approach failed since competing against legitimate content providers is very difficult and the latter approach was unsuccessful since it created a segmented information space. Last, "push" technologies did not strive to create an open "push" infrastructure, instead the goal was to create competing, proprietary tools that kept entry barriers artificially high and also led to a segmented information space. To date, the only successful "push" technology is email and this is largely because email does not exhibit any of these previous traits.

The notion of making the Web active is a worthwhile goal. The popularity [16] of chat-rooms and instant messaging software, such as AOL Instant Messenger [2] and ICQ [9], is pointing towards a technology that can deliver on the promise of an active Web, that technology is wide-area event notification – or simply events. Event-based interaction is effective in many distributed systems [8] and this is particularly true in very loosely coupled distributed systems, such as the Web. From a simple perspective, if each Web site merely generated events whenever a Web page was updated, moved, or deleted, it would enable stale bookmarks prevention and automatic link repair. Pushing events further, each Web site has a set of rich semantic events that is of great interest to its users, for example, breaking stories on *CNN.com* or the final score in a specific sporting match on *ESPN.go.com*. To better utilize the information space of the Web, an open event notification mechanism is needed to bridge the Web's dynamism with the user's awareness of those dynamics.

In order for event notification to succeed where "push" failed, it is important that past failures are not repeated. First, the focus of event notification is merely to announce the occurrence of some event or to provide minimum content to facilitate interaction, not to distribute content. Second, event notification services must be open and entry barriers kept reasonably equivalent to those of running a Web server. Third, event notification services must not artificially segment the information space. Last, event notification services must scale to the potential level of event traffic on the Web.

This paper contends that an open, wide-area event notification system is the solution for making the Web active. In this solution, event notification is not a separable or orthogonal activity, but an integral part of the Web that is currently missing. Based on this assumption, this paper introduces the *Event Desktop*, a simple infrastructure tool built on top of a wide-area event notification system. The main focus of the Event Desktop is to create a mechanism to enable the dynamic dissemination and management of event clients. The Event Desktop is not presented as a solution to all that ails the Web; rather it is presented as an open platform on which to experiment with the next wave of active Web applications.

This paper describes the components of the Event Desktop in the next section, followed by a presentation of an example event client in section 3. Section 4 discusses approaches and issues in security, authentication, and privacy. Section 5 discusses the relationship to other prominent work, followed by the conclusion.

2 The Event Desktop

The World Wide Web started out as a grassroots movement of people trying to share information. To participate in this new world, only an HTTP server and browser software were necessary. At its most basic, the Web brought together information producers and information consumers by combining and standardizing existing technologies. Another potential grassroots movement is starting to swell around a new technological combination, one that promises not only information sharing but also interaction and communication. This technology is wide-area event notification. The growing popularity of instant messengers, chat-rooms, and mailing lists is a bellwether of this movement.

Just as the Web and its associated browser software pulled together ad-hoc information sharing methods under one umbrella, so must there be a movement to bring ad-hoc notification technologies, such as email and instant messaging, together. This movement should be viewed as a natural progression of the Web and not an orthogonal movement. It is not enough to bring information publishing to the masses, like the Web has done. It is necessary to push further and bring wide-scale awareness, interaction, and communication to the masses as well.

To this end, the Event Desktop is based on the assumption that a combination of a wide-area event notification system and the Web's information publishing capabilities is a necessary eventuality. In such an environment, access to event clients to interpret and interact in the event space is of paramount importance. Therefore the Event Desktop marries dynamic code loading and automatic state management with a wide-area event notification system to create an infrastructure to support the dissemination and management of specialized event clients, called *eventlets*. An eventlet is a small, independent component used to interpret classes of events in order to facilitate information awareness, communication, or interaction. Eventlets are packaged as Java JAR files [13].

Examples of eventlets are limited only by the variety of available *event sources*; an event source is any service that generates events. The following are typical examples of event clients/sources:

- instant messengers,
- no-maintenance mailing lists,
- anonymous sales material request clearinghouse (e.g., solicited SPAM),
- Web site communities (i.e., combining the Web browsing with awareness of other people's presence),
- Web site(s) monitoring (e.g., specific or aggregated ticker-tapes),
- collaboration and gaming,
- distributed simulation or distributed computation (e.g., SETI@home),
- legacy code monitors (e.g., connected to database triggers), and
- auctions and e-commerce exchanges.

The Event Desktop consists of five major components [see Figure 1]. Two of these components were introduced above; eventlets are interpreters of events and event

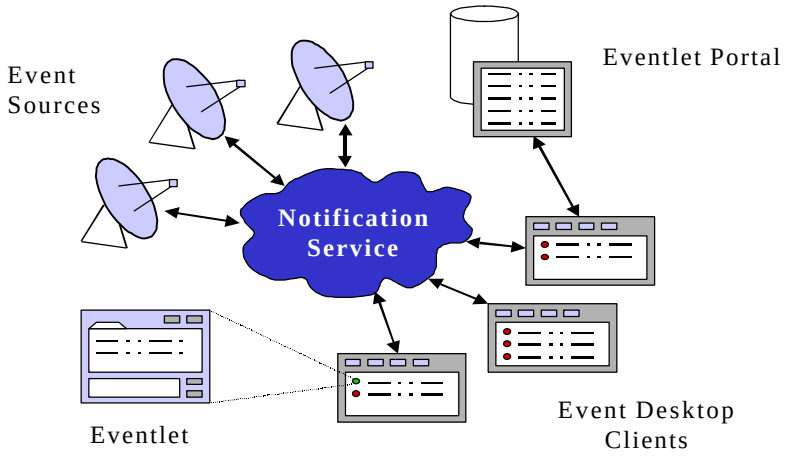


Fig. 1. Event Desktop architecture.

sources are generators of events. The *eventlet portal* is a repository for eventlets where users can search for desired eventlets or receive "new eventlet" notifications. The *event desktop client* is the client-side portion of the of the Event Desktop infrastructure, it provides a central point of management for eventlets on the end-user's computer as well as providing a local event notification system. Finally, the *wide-area event notification system* connects all event desktop clients and event sources. The following subsections examine each of these components in more detail.

2.1 Eventlets

Eventlets are built on top of a simple dynamic code loading mechanism for Java called codelets, which implement a standard "plug in" mechanism. A codelet object implements the `Codelet` interface, defined as:

```
public interface Codelet
{
   public void startup(Context context, Object arg);
   public void activate(Context context, Object arg);
   public void passivate();
   public void shutdown();
}
```

Implementing this interface enables the codelet infrastructure to dynamically load and execute codelet JAR files and automatically persist and restore their state. The JAR file may contain any number of classes and resources, but it must contain a codelet manifest file that lists the "main" codelet class. To use the codelet infrastructure an application only needs to create an instance of the `CodeletManager` class. The

first time a codelet is loaded into an instance of a codelet manager, the codelet's `startup()` function is called; this is for one time initialization and is never called again. An application can passivate its codelet manager, which will in turn call `passivate()` on all loaded codelets and persist the respective state of each. When an application restarts and activates its codelet manager, the codelet manager reconstructs each codelet from their persisted state and calls `activate()` on each. An application can also remove codelets from the codelet manager, which will call `shutdown()` on the codelet and remove its state from persistent storage.

The `Eventlet` interface extends the `Codelet` and `Notifiable` interfaces. The notifiable interface is an entry point for the event notification service [see section 2.5] and is simply a single member function used to receive an event. The `Eventlet` interface adds member functions to define and monitor an event client and is partially defined as:

```
public interface Eventlet extends Codelet, Notifiable
{
   …
   public String getName();
   public String getDescription();
   public String getVersion();
   public String[] getCategory();
   public Icon getIcon();
   public boolean isShowing();
   public void setShowing(boolean b);
   public int getStatus();
   …
}
```

An eventlet may or may not have an associated user interface. In the case of an eventlet with a user interface, the eventlet must provide mechanisms to show, hide, and query the visibility of the interface. An eventlet's status may be active (the eventlet is visible), inactive (the eventlet is not visible), or waiting (the eventlet is not visible and events were received since it was last visible). Eventlets are not serializable by default; it is up to the eventlet designer to decide whether the eventlet's state should be persisted. To implement a serializable eventlet, the eventlet implementation only needs to implement the `Serializable` interface.

2.2 Event Sources

Event sources simply use the interfaces provided by the wide-area event notification system to inject event notifications into the global event space. Typically an event source is an independent service running somewhere on the Web. Eventlets can publish event notifications also, so they are an event source of sorts. It is quite reasonable to create interesting eventlets that do not have an associated event source. For example, in the Siena Messenger [see section 3], the only event sources are the eventlets themselves. As more experience is gained from creating event sources, it is quite

likely that new framework pieces will be introduced to simplify the tasks associated with creating event sources.

2.3 Eventlet Portal

The eventlet portal is a simple database of available eventlets to simplify discovering available or newly release event clients. The eventlet portal database need not contain the actual eventlet JAR files, only a reachable URL. Access to the database is provided via a servlet interface and is not intended for users; rather it is accessed programmatically by the event desktop client. The notion of an eventlet portal is not central to the Event Desktop infrastructure and is largely included as a convenience. Since eventlets are JAR files, they are easily embeddable in Web pages and email as other means of dissemination.

2.4 Event Desktop Client

The event desktop client is the entry point into the global event space for the end-user. The client provides management facilities for the eventlets themselves, including making them accessible to the user, managing state, and providing resources. The event desktop client is very critical in this last respect since it is granted trusted permission to the local file system and network sockets. The event desktop client uses these privileges to provide controlled access to critical resources for downloaded eventlets. For a more detailed discussion of these issues refer to section 4.

The event desktop client presents itself as a "switchboard" [see Figure 2], borrowing from the telephone switchboard paradigm where rows of lighted buttons represent telephone lines and the light itself represents the status of the line. Each installed eventlet is listed in the viewable switchboard area, displayed with a status indicator, the eventlet icon, and the eventlet name. The status indicator is a green light, a red light, or a blinking red light. A green light indicates that the eventlet's user interface is visible. A red light indicates that the eventlet's user interface is not visible. Lastly, a blinking red light indicates that the eventlet's user interface is not visible and new events were received since the last time it was visible. Simply clicking on an eventlet in the switchboard displays its user interface.

The event desktop client also allows the user to configure various aspects of the event desktop infrastructure, such as configuring the default eventlet portal and event server access point. The event desktop client also provides a simple "log in" service that is used to support multiple users and multiple user-profiles.



Fig. 2. Event desktop client.

2.5 Wide-Area Event Notification System

The Event Desktop infrastructure is not intimately tied to any particular event notification service, although it does rely heavily on the availability of a wide-area event notification system. Various alternatives exist, but most are insufficient for the purposes of the Event Desktop. For compelling reasons, the Event Desktop has adopted the Siena [4][5] event notification service. Siena is an open event notification system specifically designed for wide-area scalability with programming interfaces in both C++ and Java. Siena uses a familiar publish-and-subscribe interface. Siena structures event notifications as attribute-value pairs where attributes are typed. Event notification content is not opaque and all attributes are usable for subscription filtering purposes. As a result of this approach, Siena is able to use sophisticated content-based algorithms to route event notifications efficiently among a topology of event notification servers.

3 Example Eventlet

Currently five eventlets exist, two purely for pedagogical reasons. A *Hello World* eventlet demonstrates the minimum requirements for sending and receiving events. A *ViewURL* eventlet demonstrates secure capabilities by allowing the user to view arbitrary URLs by forwarding the request to the event desktop client. A *Desktop Poker* eventlet demonstrates collaborative gaming applications by implementing a six-player version of the card game of Poker; this eventlet is non-centralized and therefore allows an unlimited number of games to be played concurrently. The final eventlet is the *Siena Messenger* that is described in the following paragraphs.

The Siena Messenger is a reasonably full-featured instant messenger program [see Figure 3]. The Siena Messenger is not intended as a replacement for existing instant messengers, rather it illustrates that instant messaging is a subset of the entire event space available on the Web. A benefit of implementing an instant messenger is that all event traffic is generated by the messenger eventlets themselves and no external event source is required.

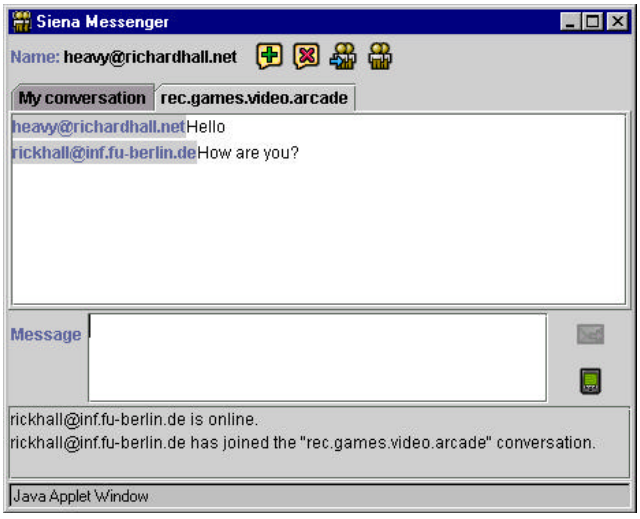


Fig. 3. Siena Messenger eventlet.

The Siena Messenger defines a simple event schema on top of Siena's general attribute-value pair data model. All events have a scope attribute, which is the full Java class name of the messenger eventlet. All events also have a type attribute to indicate whether the notification is a login/logout, announcement, page, ping, or message event. Each of these event notification types has associated attributes to define for example, who sent the event, potentially the specific person who should receive the event (in the case of a page), and the conversation in which the message belongs.

With only these five event types, the Siena Messenger supports multiple conversations in a tabbed-pane user interface; each conversation appears in its own separate tab in the main interface window. The user can create any number of personal conversations or can join standard "chat-room" conversations.

When a user is in a conversation, he or she can page other users to join the conversation. Any users who are paged are given the option to accept of reject the invitation; accepting an invitation merely creates another conversation pane in the tabbed interface. The Siena Messenger supports "buddy" lists to monitor when specific users log in or out. Siena Messenger also supports sound effects for many of the various action events; all sound files are loaded directly from the eventlet JAR file using the Java resource interface. Since the event desktop client handles state persistence, all configuration settings are automatically saved across sessions. The Siena Messenger provides all of these features without access to any secure or critical resources.

4 Security, Authentication, and Privacy

The following subsections summarize both what is currently available and what is proposed for security, authentication, and privacy in the Event Desktop.

4.1 Security

The client-side portion of the Event Desktop infrastructure is completely implemented in Java and as a result, relies on Java's execution sandbox for its protection mechanisms. A standard security policy file is distributed with the event desktop client. The included security policy file gives the event desktop client trusted access to sockets and files; file access does not include delete permission. The event desktop client does not require all of these permissions to perform its main functions; rather it uses these extended permissions to grant controlled access to installed eventlets. For example, an eventlet can request to read/write a file with the user's permission or to read/view a URL.

It is important to point out that a downloaded eventlet is granted no permissions by the event desktop security policy. This means that eventlets do not have direct access to sockets or files and any attempt to directly access these resources will result in an access exception. To provide eventlets with controlled access to these resources, the event desktop client augments the Java security sandbox with capabilities. A capability is an object that gives limited access to a secure resource. As of version 1.2 of the

Java Development Kit, the Java security access controller allows privileged code execution within a `doPrivileged()` block. When code is executed inside such a privileged block, the Java access controller ignores the caller's permissions and only considers permissions granted to the codebase where the privileged block is defined and its dependent codebases.

As a result, the event desktop client can open a file input stream in a privileged block, for example, and pass back a reference to an eventlet for subsequent reading. Since the security manager does not perform security checks on subsequent reads to the input stream, the untrusted eventlet is able to successfully read the file, but nothing else. On the other hand, if the eventlet tries to directly open the same file, a security exception is thrown.

Regarding firewall and Web server proxy support, since all event and URL access is handled in the event desktop client, no eventlets need to concern themselves with these issues. Currently the event desktop client uses the standard Java mechanisms to support URL access through proxies. Siena does not currently operate through proxies and requires a remote socket for sending events and a local socket for receiving events. The goal is to extend Siena with a simple HTTP interface for sending events and to create a bridge from HTTP to Siena events in the form of a servlet or some similar technology.

4.2 Authentication

The event desktop client offers a simple "log in" service, but currently this service does not perform user authentication. The goal is to utilize a public key server on the Web for the purpose of authenticating users. The first time a user logs into the event desktop client, the event desktop client will ask for the user's email address and will generate a public/private key pair for them. The event desktop client will store the private key in the user-profile directory and will send the public key to the public key server along with the user's email address. The public key server will use the email address to query the user via email for purposes of verification of intent to create an Event Desktop user account.

With this in place, the event desktop client is able to use content signing techniques to authenticate event notifications published by a user; whether all client-side events are authenticated or only user-selected events are authenticated is undecided at this point. Authenticating all client-side events is reasonable with respect to generated workload because it is unlikely than any given user will generate an overwhelming number of notifications. Event sources will use the same techniques as client-side authentication to authenticate their notifications. For both event clients and sources, a trade-off exists between the desire for authenticity and the desire for efficiency.

4.3 Privacy

Privacy issues are much more complex than security and authentication. The algorithms that Siena uses for subscription filter propagation alleviate some privacy is-

sues. For example, it is not possible for a rogue event server to listen for all subscriptions in order to build a complete list of subscription information. Siena event servers do not forward the identity of the requester of a particular subscription, nor do they forward the precise subscription; Siena aggregates similar subscriptions using covering relationships. As a result of these subscription propagation algorithms, Siena exhibits some characteristics of a subscription anonymizer. An end-user need only trust its entry-point Siena server to not divulge such information.

Unfortunately privacy degrades if standard user authentication is applied to each event notification. In this case it is possible for the actions of a specific user to be tracked since a public key uniquely identifies the user. This may result in the use of real anonymizers as an orthogonal issue that an end-user may or may not choose to use. Additionally, an end-user could choose to authenticate only certain event notifications to limit their visibility. One final point, even with full authentication it is only possible to determine who is sending information, but not who is receiving it.

5 Related Work

It is possible to create a similar environment for event clients using applets [7] and Web browsing software, but the complications of this approach appear to outweigh the benefits. The event desktop client requires relatively strict control of the life cycle of an eventlet. Implementing an eventlet via an applet inside the context of a Web browser limits control over life cycle issues. Web browsers do not offer long-term applet and state persistence, nor do they provide life cycle access for applets. Despite these issues, it is possible to implement a transient eventlet (i.e., no state persistence) as an applet or to persist state on a server, assuming that the Web browser security policy is sufficiently flexible.

JavaBeans [14] technology is compatible with the notion of an eventlet. As a component model, JavaBeans has defined only a few semantics on interfaces associated with any given Java class. JavaBeans technology provides a stylized way to create Java classes so that programming tools can manipulate and connect them visually. The eventlet interface defines an additional, minimal set of semantics for a component to deal with life cycle and monitoring issues. It is feasible to implement an eventlet as a JavaBean, although the value in doing so is small. Eventlets are small, independent components and are not intended for manipulation in programming editors.

A more comparable framework to the Event Desktop is the recently introduced Java Network Launcher Protocol (JNLP) [12]. JNLP defines a framework for simplifying the client-side experience for Java by enabling the dynamic downloading, update, and execution of Java applications directly from the Web. JNLP also provides services to applications, such as file and URL access, but does not include any event service. JNLP applications are described using a separate XML descriptor file that contains, among other information, the location of JAR files and the “main” class used to execute the application. JNLP executes all applications in separate Java virtual machines, which is too heavyweight for event clients. JNLP also does not provide automatic state persistence. Despite these differences, future versions of the Event Desk-

top may gravitate towards the JNLP interface definitions in order to provide interoperability as well as to extend JNLP functionality.

A host of proprietary instant messengers and event clients exist for many of the current major Web portal sites; the major players among them are AOL Instant Messenger (AIM) [2], ICQ [9], Excite Assistant [3], Yahoo Messenger [15], and MSN Messenger [11]. AIM and ICQ were the first to gain wide-scale popularity and other event clients have come on strong since. Each of these systems is following a very similar path as "push" technology; they are focusing too narrowly on being both the content provider and the infrastructure creator, thus each is an independent solution trying to tie end-users to a specific Web portal. These event clients offer more than simple instant messaging capabilities, such as news and file transfers. Unfortunately the creators of these systems are not abstracting to the general notion of wide-area event notification. As a result, the event infrastructure is centralized, the event spaces are closed, and the event clients are monolithic. There is some interest in standardizing the instant message event subspace [1], but this is largely a result of competitors seeking access to the market share of the leading instant messenger.

6 Conclusion

Despite the instant appeal of treating the Web as if it were merely a global storage space, this approach does not reflect the Web's true nature. The Web's enormous capacity and meteoric rate of change nullify simple "hard disk" management techniques, such as hierarchical organization, short-cuts, and "find" operations. As a result of these unique characteristics, this paper suggests that the Web requires an "active" element in the form of wide-area event notification. An open, standard wide-area event notification system is not a panacea, rather it is simply a necessary extension that is currently missing. The importance of such a mechanism is witnessed by the myriad of companies and technologies using ad-hoc methods and mechanisms to achieve the same effect.

Under the belief that wide-area event notification is a necessary eventuality, this paper introduces the Event Desktop infrastructure. The Event Desktop concentrates on creating an open platform for event client dissemination and management. To achieve these goals, the Event Desktop provides dynamic code loading and automatic state persistence capabilities on top of the Siena event notification system. These capabilities combine with a secure end-user interface for managing and manipulating event clients to form a platform on which to experiment with the next generation of Web services and applications. An initial implementation of the Event Desktop is available including a small set of eventlets to demonstrate its capabilities.

7 References

[1] S. Aggarwal, C. Benson, J. Stracke, and C. Vermeulen. "Transport Protocol for Presence Information/Instant Messaging." IETF, Internet Draft. March 9, 2000.

[2] America Online, Inc. AOL Instant Messenger. Version 4.0. 2000. http://www.aol.com/aim.
[3] At Home Corp. Excite Assistant. Version 1.1.3.3. 2000. http://excite.com.
[4] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. "Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service." Proc. of the 19th ACM Symposium on Principles of Distributed Computing. July 2000.
[5] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. "Interfaces and Algorithms for a Wide-Area Event Notification Service." University of Colorado Technical Report. CU-CS-888-99. October 1999.
[6] EntryPoint, Inc. Acquired PointCast in 1999. http://www.entrypoint.com.
[7] D. Flanagan. "Java in a Nutshell." O'Reilly & Associates, Inc. 1997.
[8] D. Garlan and D. Notkin. "Formalizing Design Spaces: Implicit Invocation Mechanisms." Proc. of VDM '91: 4th International Symposium of VDM Europe on Formal Software Development Methods. October 1991.
[9] ICQ, Inc. ICQ. Version 2000a. 2000. http://web.icq.com.
[10] Marimba. "The Castanet Product Family." Marimba, Incorporated, 1997. http://www.marimba.com/doc/general/current/introducing/introducing.html.
[11] Microsoft Corp. MSN Messenger. December 7, 1999. http://messenger.msn.com.
[12] Sun Microsystems. "Java Network Launcher Protocol & API Specification." Version 1.0 [Public Draft 1]. June 1, 2000.
[13] Sun Microsystems. "JAR File Specification." 1999.
[14] Sun Microsystems. "JavaBeans." Version 1.01. July 24, 1997.
[15] Yahoo, Inc. Yahoo Messenger. Version 3.0.0.768. 2000. http://messenger.yahoo.com.
[16] Ziff Davis Media Inc. "Chat Goes to Work." PC Magazine. January 12, 2000.