Providing a Web Interface for Active Spaces

James F. Bresler, Roy H. Campbell, Jalal Al-Muhtadi
{jbresler, rhc, almuhtad}@cs.uiuc.edu
Department of Computer Science
University of Illinois at Urbana-Champaign

December 10, 2003

Abstract

Pervasive computing promises a new significance in our lives by providing seamless computing services. Much of the current research on active spaces is focused on supporting new devices and applications that provides compelling reasons for users to adopt pervasive computing. Simultaneously, there is significant research on accessing content in information spaces. It is possible to treat the Web like a device in an active space, resulting in a merged active and information space. This allows users to leverage existing information space technologies to interact with active spaces. This can be used to easily distribute content produced in an active space to a very large number of users and interact with an active space from a traditional computing environment. In this paper we describe several requirements and challenges for making an active space accessible over the Web, describe a prototype implementation, and propose a direction for future research.

1 Introduction

Pervasive computing promises a new computing environment where users seamlessly interact with a large spectrum of devices and applications. These computing environments allow users vastly greater functionality than traditional computing environments, ease everyday tasks, and allow users to be more productive. We refer to these integrated computing environments as active spaces [12]. As users embrace active spaces, they will start to produce content that needs to be distributed to people outside of their active space. Therefore, a mechanism is needed for remote users to access an active space that supports a wide variety of applications, imposes minimal requirements for end users, provides strong security, and scales well to large numbers of clients.

There are many ubiquitous computing applications that require collaboration between users who preside in different locations; some of these users may not be part of an active space. For example, a meeting for a group of distance learning students requires collaboration between several users who reside in different locations. For these situations, it is possible to form a virtual active space that can be accessed using existing communication technologies. Users could join multiple virtual active spaces, allowing users to appear to be at multiple places at the same time and interact with several environments. Therefore, a mechanism is needed to access an active space that is reliable, uses existing communication mechanism, provides strong security, accomodates large numbers of clients, and supports a wide variety of applications.

There will be a large group of users that gradually adopt pervasive computing. These users will want to access their active spaces from traditional computing environments. Many of these users will demand access to their active spaces from environments like hotel rooms or Internet cafes where limited infrastructure is available. Many of these users require technology that works over a firewall. Furthermore, a large group of users may be interested in accessing an active space from devices such as cell phones or PDAs where they can access the Web but cannot run reflective middleware services. Therefore, a mechanism is needed to remotely access an active space that is secure, supports a wide variety of applications and scenarios, requires no software installation on a computer accessing the active space, is easy to use, and uses an existing, very well supported communication mechanism like the Web.

There has been significant research and work on web applications frameworks such as Struts [6, 13] in the last few years. However, these frameworks do not integrate with active spaces, limiting interactions with the physical world. There has been significant

research on controlling ubiquitous computing environment from a web browser [11], and research has been performed to determine how users in an active space can access the Web [1]. Because of this research and the ubiquity of web accessible computers, we believe the Web is an ideal environment for remotely accessing active spaces.

1.1 Problem

We believe that component based active spaces will become commonplace. Some of these systems, such as those based on Gaia [12], use reflexive middleware that cannot be used behind firewalls and require special software installation. Furthermore, although we expect most active spaces to support several components viewing or providing input to an application, some active spaces may support only a limited number of these components efficiently.

The Web is an ideal mechanism to access an active space because of its ubiquity. However, the Web is designed to pull data. Many applications need the ability to push data so an effective user interface can be designed. Furthermore, many applications require the ability to instantly respond to user events and intelligently encode data. This can be accomplished by embedding application logic in a web page using technology like Java [7]. However, this logic runs in a severely restricted environment; as a result of this, all network communication from a web client must be to the machine containing the web server [3].

Several features have been proposed for active spaces that include having a computer respond when a user walks next to it [10] and using badges to track users [9]. These features are impossible to support over the Web because users may not have the necessary hardware.

Many of these issues can be addressed by inserting a proxy inside an active space that interacts with Web clients. This allows one component in an active space to interact with several clients, clients to interact with an active space when they cannot use reflexive middleware, client software that is simple to implement, and client software that can under restricted but common environments like the Java VM [3].

2 Active Space Web Accessibility Challenges and Requirements

In this section, we discuss the challenges and requirements for making an active space web accessible.

2.1 Challenges

Providing web accessibility support for an active space creates several unique challenges.

2.1.1 Authentication

Authentication and access control are necessary features for practical web accessibility. It is possible that several users accessing an application over the Web will share one set of credentials, especially if an active space cannot support a large number of components bound to an application efficiently. A mechanism is needed to determine the appropriate set of credentials to issue for a set of users accessing a common application. In addition, an algorithm is needed to determine whether a new user has sufficient privileges to share a set of existing credentials.

2.1.2 Context Awareness

An active space may support features to determine the context of an application. However, clients accessing an application over the Web have very limited context information, and several clients sharing one component may confuse the active space. Future work is needed to address these problems.

2.2 Requirements

There are several requirements for providing web accessibility support for an active space.

2.2.1 Application Support

Users in an active space are accustomed to using a wide variety of applications on a routine basis. These users will demand web accessibility for a wide variety of applications to approximate native remote access of an active space.

To minimize the number of applications that must have web accessibility support manually written, it is possible to use active space features to automatically support some applications. For example, active spaces running Gaia use a context file system [5] capable of converting data to a format desired by an application on the fly. This can be used to write one web accessibility component that supports viewing PowerPoint presentations and PDF documents [5].

2.2.2 Adaptable

Users accessing active spaces on the web are using a transitional technology to approximate native remote access of an active space. Therefore, it is critical that web accessibility works in a large variety of settings. Many environments, such as airport terminals, do not permit users accessing the Web to install software. To

avoid this, logic can be embedded within a web page using technology such as Java [7].

It is also possible that several users wishing to access an active space are behind a firewall. A large percentage of firewalls allow outgoing connections to certain TCP ports, like 443. This implies that it is possible to support clients behind a large percentage of firewalls by only communicating over a limited number of well known ports.

2.2.3 Scalable

Users in an active space may desire support for allowing a large number of clients to view content simultaneously. Some active spaces do not handle a large number of components efficiently. Therefore, several web clients must share a single component to support a large number of users.

The network I/O for certain types of content can be significant when data is transferred to a large number of clients. Many of these applications can achieve a significant reduction in bandwidth usage by encoding data in an intelligent manner.

2.2.4 Interopable

The web accessibility support for an active space must not affect the design of the active space. This suggests that clients accessing an active space over the Web must appear as standard components to the active space, possibly after going through a proxy.

3 Case Study: Gaia OS Web Proxy

Gaia is a component based meta operating system that provides an infrastructure designed to facilitate ubiquitous computing. It supports a wide variety of applications running on a large selection of devices, context awareness, user authentication, and access control [2, 12]. Gaia uses a component model that partitions applications similar to the Model-View-Controller [8] by dividing applications into model, input sensor, presentation, and coordinator components. A model component implements the logic of an application, an input sensor component is responsible for informing the model component when an input event occurs, a presentation component exports the application's data to a viewable form, and a coordinator components is responsible for the management of other components [12]. The components are implemented using CORBA; CORBA uses nonstandard ports that many firewalls block. A web proxy has been implemented for Gaia that provides a generic infrastructure for web accessibility. Functionality has been implemented to allow clients over the Internet to view or control presentations and monitor the status of an active space.

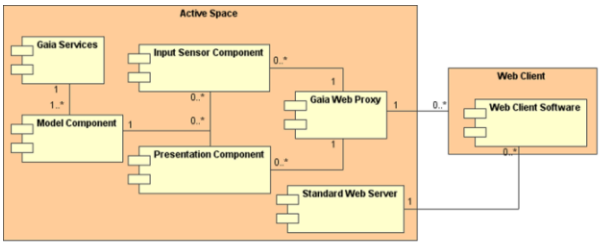

Figure 1: Gaia Web Accessibility Architecture

3.1 Proxy Architecture

The Gaia Web Proxy is based on the proxy design pattern [4] and controls access to Gaia from web clients. The Gaia Web Proxy resides in an active space and contains a mix of presentation and input sensor components bound to model components. The Gaia Web Proxy will encode and send clients information based on events the component receives, and, for input sensor components, will change the state of the application based on client messages.

Clients utilize the Gaia Web Proxy by connecting to a standard web server in the active space which resides at a well known location. The content received from the web server contains logic that allows users to interact with an application by connecting to the web proxy.

3.2 Proxy Design

The Gaia Web Proxy must perform bidirectional forwarding of information between web clients and Gaia. This information can be obtained by having the proxy contain a set of components bound to applications. To minimize risk of security compromise due to different application privilege levels and the risk of instability due to application logic bugs, each component will reside in a separate process.

Each component process will directly communicate with web clients accessing the application the component is bound to. The component process is responsible for encoding idempotent messages which determine a client's state when an input event occurs. These messages will be broadcasted to all the component's web clients.

Application specific logic in the component process runs in a dedicated thread. This logic calls a function when a new message needs to be broadcasted

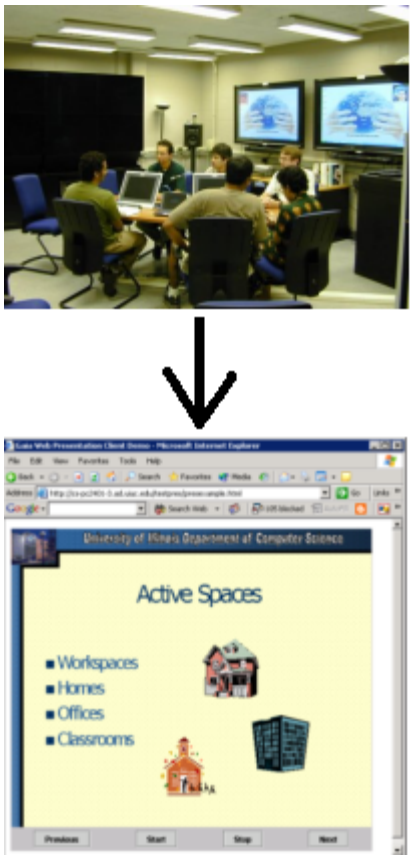

Figure 2: Screenshot of a web client viewing a presentation syncronized with a meeting

and processes incoming messages from web clients. It is possible to use technologies such as COM [14] to allow application support to be implemented without modifying the component process's logic.

The Gaia Web Proxy component processes will be managed by a master process. The master process is responsible for accepting new connections from clients, finding out which component they wish to access, and then moving control of the client to the appropriate component process. The master process allows one network port to be used for all web clients and still put each component in a separate process.

3.3 Proxy Implementation

The Gaia Web Proxy performs bidirectional forwarding of information needed to interact with an application between web clients and Gaia. This information can be obtained by maintaining a set of components that are bound to applications. These components are created by treating component processes as standard Gaia components and sending messages to Gaia requesting their creation.

Communication between the master process and component processes is implemented with a bidirectional named pipe. The master process initiates communication with a component process when a new client wants to access the component's application. When this occurs, the socket API is used to transfer ownership of the client's socket to the component process. The named pipe is also used by the component processes to inform the master process when a web client disconnects, a component is about to terminated, or any serious error occurs.

The proxy's design suggests a method is needed to determine which component a web client should associate with. This is implemented by requiring each component process to contain a unique, user specified application identifier. When a client connects to the proxy, he sends an application identifier which the master process attempts to match to a component process. This mechanism allows a web client to pick a specific application component to associate with when several different components are available. Because Gaia implements access control by assigning credentials to components [2], this design allows different privilege levels for users accessing the same application.

This results in an implementation of the Gaia Web Proxy where every client initially connects to a standard web server located inside an active space that resides at a well known location. The user receives a web page containing logic to connect to the Gaia Web Proxy and request a specific application component. When a web client connects, the master process uses non blocking I/O to obtain the application identifier from the client. Then, the proxy attempts to find a component process by looking up the desired application type in the component process directory. If an appropriate component process is available, the client's socket will move to the component process.

This implementation gives the component processes the responsibility of encoding and sending messages to clients. This is implemented by providing basic component process functionality and a simple API for writing application support. The core functionality contains logic to broadcast messages to several users, forward input messages to application logic, and communicate with the master process. The core component process functionality requires messages to be idempotent and determine the state of a client. This allows the component process to drop messages if a client cannot receive them at a sufficiently fast rate.

3.4 Benefits

The Gaia Web Proxy satisfies the requirements for web accessibility support.

3.4.1 Extensible

The Gaia Web Proxy component process can be modified to support an additional application by implementing the application specific logic in the component process.

3.4.2 Simple Client

The Gaia Web Proxy is designed to make clients as simple as possible. A client is responsible for connecting to the proxy, sending the desired application type, and waiting for messages. When application input is supported by the proxy, clients trigger events by sending messages over the socket. This functionality can be implemented in a Java applet. Therefore, no client software installation is necessary. The client code for viewing and controlling a presentation contains less then 225 lines of Java.

3.4.3 Scalable

The Gaia Web Proxy supports sharing components between several web clients accessing the same application identifier, minimizing the load on Gaia. The only significant marginal resource utilization for a client requesting use of an existing application component is network I/O.

3.4.4 Works Over Most Firewalls

The Gaia Web Proxy only uses port 443 to communicate with clients. This is allowed by most firewalls.

3.4.5 Interopable

The Gaia Web Proxy does not require any changes to Gaia to support web applications.

3.5 Future Work

The Gaia Web Proxy needs future work to support authentication and context awareness.

3.5.1 Authentication

The current implementation of the Gaia Web Proxy does not support authenticating users. It is possible to address this by requiring web clients to send authentication credentials along with the application identifier to the master process. The master process would use the Gaia Authentication Service to verify the credentials are valid and sufficient.

3.5.2 Context Awareness

The Gaia Web Proxy provides limited context awareness by allowing applications to provide or receive context information. For example, an application has been implemented that receives events from the Gaia Event Service [12] when users enter or leave the active space. The application uses the proxy to allow web clients to receieve a constantly updated list of users in the active space.

Future work is needed to determine how the proxy's core logic can use context information to improve the user experience on a non application specific level.

3.6 Evaluation

The Gaia Web Proxy provides a practical mechanism for a large number of web clients to interact with an active space. The web accessibility support has been demonstrated to guests visiting the active space laboratory at the University of Illinois at Urbana-Champaign. Furthermore, the proxy is often used internally to view presentations and monitor active space parameters.

The current version of the Gaia web accessibility support does not authenticate users properly and does not support context awareness. Authentication can easily be implemented in the future. Future work is needed to determine how to use context information obtained from web clients.

Several active space features cannot be supported because web clients do not have sufficient hardware. For example, determining a web client's location cannot be supported. However, most users do not expect ubiquitous computing to be available in every environment. Many of these users will be content with using the web as a transition technology until pervasive computing is more widely adopted.

4 Conclusion and Future Work

In this paper, we documented how an active space and an information space can be merged, resulting in a scenario where users can interact with an active space from an information space. Because of the ubiquity of the web and previous research findings, we believe the Web is in ideal transition mechanism for accessing active spaces remotely. We described why web accessibility support for an active space must support a wide variety of applications, cannot require a client to install software, must work over firewalls, and must not require changes to the active space's component model. Then, we described an implementation for web accessibility support using a proxy and

demonstrated that a proxy can satisfy the web accessibility requirements.

In the future, we plan to develop web accessibility support for additional Gaia applications. In particular, we plan to focus on supporting applications that have frequent, incremental updates such as drawing vector graphics. Finally, we will investigate techniques to authenticate web clients accessing an active space and using a web client's context to uniquely tailor content.

References

[1] Sven Buchholz and Alexander Schill. Web Caching in a Pervasive Computing World. *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (July 2002)*, 2002.

[2] Roy H. Campbell, Jalal Al-Muhtadi, Prasad Naldurg, Geetanjali Sampemane, and M. Dennis Mickunas. Towards Security and Privacy for Pervasive Computing. *Theories and Systems, Mext-NSF-JSPS Internation Sympsoium (IIS 2002)*, 2002.

[3] J. Steven Fritzinger and Marianne Mueller. Java Security , 1996. Sun Microsystems White Paper.

[4] Eric Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Massachusetts, 1995.

[5] Christopher K. Hess and Roy H. Campbell. A Context-Aware Data Management System for Ubiquitous Computing Application. *International Conference of Distributed Computing Systems (ICDCS 2003)*, 2003.

[6] Ted Husted, Ed Burns Craig, and R. McClanahan. *Struts Manual*. Apache Group, 2003.

[7] Douglas Kramer. The Java Platform: A White Paper , 1996. Sun Microsystems White Paper.

[8] G. Krasner and S. Pope. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of Object Oriented Programming*, 1(3):26–49, 1988.

[9] Mike Addlesee and Rupert Curwen and Steve Hodges and Joe Newman and Pete Steggles and Andy Ward and Andy Hopper". Implementing a sentient computing system. *Computer*, 34(8):50–56, 2001.

[10] Gopal Pingali, Claudio Pinhanez, Tony Levas, Rick Kjeldsen, and Mark Podlaseck. User-following Displays, 2002.

[11] Shankar R. Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd. ICrafter: A Service Framework for Ubiquitous Computing Environments. *Lecture Notes in Computer Science*, 2201, 2001.

[12] Manuel Roman, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces . *IEEE Pervasive Computing*, Oct-Dec:74–83, 2002.

[13] Steven Sweeting, Clive Jones, , and Aaron Rustad. Integrating and Mapping a Web Application MVC Pattern. *Java Developers Journal*, 2002.

[14] Sara Williams and Charlie Kindel. The Component Object Model: A Technical Overview , 1994. Published on Microsoft Developer Network.