

Gaia Microserver: An Extendable Mobile Middleware Platform¹

Ellick Chan, Jim Bresler, Jalal Al-Muhtadi, Roy Campbell
University of Illinois at Urbana-Champaign,
Department of Computer Science,
{emchan, jbresler, almuhtad, rhc}@uiuc.edu

Abstract

The Gaia ubiquitous computing platform currently supports mobile devices through a thin client proxy architecture. Mobile devices run a lightweight proxy client written in J2ME to join an active space. While this approach allows a wide variety of devices to interact with active spaces, it lacks the ability to use device specific functionality. This problem is addressed by combining the J2ME client with a microserver, which is a bridge from the native language to J2ME. The microserver-proxy approach enables thin clients to fully access device-specific features while respecting security through a standard interface as Gaia services.

1. Introduction

Gaia enables us to orchestrate our digital and physical environments into a unity known as an active space [1, 2]. Gaia currently operates on both traditional desktop computers as well as mobile devices. Our proxy service allows Gaia to intercommunicate with mobile devices as thin clients capable of acting as both display devices and input devices [3]. The proxy-based approach has several shortcomings due to our use of the ubiquitous J2ME programming language [4, 5]. Some of the problems encountered were an inability to leverage manufacturer-specific functionality on smart phones, difficulty in deploying applications over the air, and weak security. Our recent work on the proxy resulted in several architectural changes that allow us to partially remedy some of these weaknesses. In addition to improving existing services, we introduce a new component service on mobile devices that allows us to leverage the full power of the native programming environment. The remainder of this paper is divided as follows. Section 2 motivates our work. Section 3 explains our J2ME middleware. Section 4 gives an overview of the Gaia meta-operating system. Section 5 explains the microserver

and its architecture. Section 6 talks about the extended middleware, which integrates the microserver with Gaia. Section 7 details the implementation and results. Section 8 briefly looks into related work. Finally, Section 9 concludes.

2. Motivation

Through the Gaia platform, we are able to unify a physical space consisting of many digital servants into an active space, where participants act in collaboration to form a collective synergy. Our middleware approach works well for devices that have a generous amount of computational resources but lacks effectiveness in a technically advanced world populated by a plethora of mobile and embedded devices. Due to resource constraints, many of these nodes cannot support the complex frameworks necessary to communicate directly with Gaia. To resolve this deficiency, we created the Gaia proxy, a special service that intermediates between mobile devices and the Gaia system. Although the proxy solved some of the problems of extending Gaia, the proxy was only a transitional phase of our middleware evolution. To better support mobile services, we introduce several architectural changes to the Gaia middleware client devices to improve access to native capabilities.

3. J2ME Middleware Client

It is extremely important to choose a deployment environment that runs on as many types of devices as possible in a heterogeneous mobile environment. We designed our system on Java 2 Micro Edition (J2ME) [4, 5] because of its wide manufacturer support, open standards adoption and code mobility. In many cases, J2ME can be hardware accelerated on processors that support Java byte code as part of their instruction set [6]. The runtime environment in J2ME provides a standard set of abstractions for seamless application execution across

¹ This research is supported by grants from the National Science Foundation, NSF CCR 0086094 ITR and NSF 99-72884 EQ

different hardware environments while providing necessary security policies to prevent the spread of viruses and worms.

J2ME offers platform independence by running inside a virtual machine. To support customization, J2ME defines different capability-based profiles; however, these profiles do not fully encompass the set of functionality available on all of today's devices. Although J2ME provides the facilities necessary to target many platforms simultaneously, its standards lack the ability to access device-specific functionality. This limitation confines the ubiquitous application designer to the standard set of functionality allowed by J2ME. Our microserver approach provides device-specific services through an abstraction layer that enables the application programmer to use these services through a standardized Gaia interface.

4. Gaia Overview

Gaia is a distributed meta-operating system designed to facilitate ubiquitous computing. Gaia provides the necessary infrastructure for a heterogeneous collection of devices to intercommunicate and coordinate themselves seamlessly into an active space. Gaia provides an application framework that implements the six patterns needed for typical ubiquitous computing applications [7, 8]. These patterns are multi-device support, user-centrism, run-time adaptation, mobility, context-awareness, and environment independence. The application framework automates use of the identified patterns and facilitates the creation and management of ubiquitous applications. The application layer extends the Model-View-Controller paradigm [9] and facilitates the decomposition of an application into different input elements, output elements, and a model, where the application logic resides. Furthermore, Gaia provides a context-aware security framework that supports dynamic roles and policies to protect the privacy and security of users [10-12].

4.1. Gaia Proxy

The Gaia Proxy Service allows resource-stripped mobile devices to communicate with Gaia by providing a level of indirection between mobile devices and Gaia. The proxy allows mobile devices to act as input and output devices by using a specialized client for each application. To best serve mobile devices, the proxy translates Gaia actions and state to mobile devices through a custom mobile-optimized protocol.

To operate with the proxy, devices must have an active client. Our prototype software is a simple J2ME thin client. This design supports several useful mobile applications that include viewing a presentation and

tracking a user's location in an active space. However, the client side design has several problems stemming from our use of the J2ME platform. J2ME's sandbox model restricts applications to device operations supported by J2ME profiles. We also lack the ability to componentize client software, which means that users have to manually download and install a client for each application. Our new microserver work alleviates some of these features and component issues.

5. Microserver

The microserver is a minimal software component that exports the advanced functionality provided by a device's native environment to the limited sandbox provided by J2ME [13]. A set of proxy objects provided by the microserver application framework exports the functionality of the native APIs. These proxy objects provide a standardized interface to common services. Figure 1 shows the layering of the microserver components. The microserver client library, which runs in the J2ME sandbox, communicates via remote procedure calls to the microserver, which can service privileged requests. Our middleware layer, described in Section 6, exports the machine functionality provided by the microserver platform to Gaia as components.

The microserver acts as an interoperability service in environments that lack a native method invocation facility. In a full Java implementation, the Java Native Interface (JNI) provides the glue functionality to call native services [14]. JNI operates by allowing native C++ code to interoperate with Java code by exposing the underlying virtual machine structures to the C++ environment. However, the J2ME CLDC profile does not provide JNI functionality; therefore, we must resort to the microserver.

Applications (J2ME)	
Middleware layer (J2ME)	
Microserver (C++)	Microserver client (J2ME)
Native libraries (C++)	

Figure 1: Microserver application layer

5.1. Microserver Replication and Packing

The microserver is written in the native development environment because necessary libraries are only accessible from the native domain. Due to this limitation, the microserver must be ported to each target platform. As a result there are many platform-specific microservers for each client binary. However, all the microservers provide a standardized interface to Gaia. To ease distribution in an active space, we bundle all associated resources such as

the platform-specific microserver(s) into a single platform-independent installer. This allows any client to transfer our middleware binary to any other platform and bootstrap our system through our innovative deployment mechanism. Since J2ME does not have a universal file system interface, we cannot directly write the installation binary into a temporary location for installation. Therefore, we use a web server hosted on the device itself to deliver the appropriate microserver installation file to a web browser running on the same device. The user then accepts the download and installs the generic package to bootstrap our middleware system.

When a user wishes to share a copy of our middleware to other mobile devices, the system can utilize the device's Bluetooth [15] or infrared-based transfer capability. The system only needs to send a single Java .jar binary because the middleware system can bootstrap itself using the web installation technique. Since our approach targets devices in a platform-neutral manner, our middleware can spread quickly, enabling a whole new class of functionality-rich ubiquitous applications.

6. Virtual Machine Middleware Extension

The proxy-microserver approach allows us to create a powerful platform for mobile ubiquitous computing systems. Our platform combines the ubiquitous abilities of Gaia with the extended reach of the microserver on many mobile devices in a generic and portable manner. In addition to providing application transparency to Gaia and Java, we offer several services that help weave our system into the fabric of an active space. First, we provide a control service through the microserver proxy that allows Gaia to interact with the advanced capabilities of mobile devices such as imaging, contact manipulation, application launching, etc. Second, we leverage Gaia to apply security policies to operations invoked on mobile devices. As part of an active space, a mobile device may run Gaia-enabled mobile applications that are context and location-aware [24]. Finally, Gaia uses the microserver's application installation facility to deliver dynamic software components and multimedia content to users. This allows Gaia to push content or customized applications onto the mobile device, making dynamic adaptation possible. Native and sandboxed applications benefit from this approach by accessing higher-level Gaia functionality through the microserver middleware framework.

In the following subsections we give more details about the services provided by the extended virtual machine.

6.1. Mobile Device Services as Gaia Components

In addition to providing a framework for application programmers to access Gaia and local hardware, we also provide several basic services necessary for allowing mobile nodes to act as addressable input and/or output elements for Gaia applications. This functionality is implemented as an interoperability component that accepts Gaia requests and executes them on behalf of the middleware system. Our architecture provides a uniform interface to access local and remote resources present in an active space.

Our mobile middleware services for Gaia are implemented as a library. This approach allows existing native applications, such as games, to utilize ubiquitous computing resources such as context and location services. We rely on Gaia to enforce component policies and restrictions. For each functionality, we create a proxy object through the Gaia proxy service. This allows us to offload application logic and policies into the proxy server while allowing the mobile devices to remain a thin client. We successfully tested the new framework by building a prototype application that uses the new framework to allow active spaces to invoke multimedia operations on the mobile device. Furthermore, the application registers itself as a generic Gaia service that can be discovered and invoked by any Gaia application in the space.

The smart device running our middleware client communicates with Gaia using the proxy. The proxy presents each device as a proxy object component that uses the full set of Gaia descriptions and policies. To support dynamic adaptation of mobile device software, the microserver provides Gaia with component management abilities through the installation facility.

6.2. Security and Ubiquitous Services

Currently, J2ME uses a security model that defines different application domains [16]. On each application invocation, checks are made against the permission domain. Sometimes, the user is prompted to grant access to a capability. Eventually, a user may "click through" the security warnings emitted by the runtime system. Our middleware avoids this problem by allowing Gaia to handle security through a rich set of context-sensitive policies. The use of these security policies in the microserver allows applications to assume the minimum required set of permissions.

To provide enhanced security and access control, we utilize Gaia's authentication and access control services [10, 11]. Gaia provides a security framework that supports the creation of dynamic security policies which are

location-based and context-aware. Policies are written in a Prolog style, allowing users to specify access restrictions in an easy format. In effect, users can specify which entities have access to which functionality in the mobile device under what situations. The policy language facilitates the creation of highly adaptable and flexible policies that can provide finer-grain control over resources. Without using Gaia core services, the security policies would be static and context-insensitive limiting the overall effectiveness in a dynamically changing active space.

6.3. Dynamic Component Installation

A major weakness of the previous proxy design was the dependence on specialized clients for each application type. This design decision was made to limit the client size and allow for application-specific optimization. Such an approach does not work well in a highly dynamic ubiquitous environment, where many types of applications are available. It is not practical to ask a user to manually install a client for every application. Our new approach uses the microserver's ability to place executable components on a device. Through the microserver, our middleware can automatically install and remove components as necessary to optimize for space usage and application functionality as necessary. We do not currently address issues of code authenticity, but a simple code signing procedure is ample to ensure safety.

6.4. Integration Benefits

The proxy-microserver architecture allows devices to act as first-class members in an active space. This allows users to apply fine-grained Gaia security and privacy policies to mobile and native components. Simultaneously, it allows Gaia to integrate the mobile's services into its own environment. Services such as the event channel mechanisms are used in Gaia to propagate state change information such as entity presence and context information [17]. Our unified middleware solution is capable of providing Gaia with environmental information such as Bluetooth presence and propagating location information to the mobile device. By combining these informational sources, both Gaia and middleware applications operating on the mobile device can obtain more accurate contextual information.

To integrate with existing platform applications, our microserver is built as an application library. Potential applications include games, which need the high performance environment of C++ and assembly. Through our microserver framework, native applications can access Gaia components on any mobile device. Although

there is currently a CORBA client capable of running the full Gaia framework on devices such as Pocket PCs, such functionality does not currently exist on popular smartphone operating systems such as Symbian and Palm OS. Through our services, we can present a universal Gaia client written in J2ME to interact with a native client written in any other language that supports sockets. In essence, our J2ME middleware framework acts as a proxy from any software environment to Gaia.

7. Implementation

Our current implementation allows mobile clients running the microserver proxy to export components to the Gaia system. These components are proxy objects that may be directly addressed by Gaia applications. Proxy objects must implement several functionalities: input, output, and the ability to receive events through Gaia event channels. We found that operations on proxy objects are limited by the latencies and transmission rates of the underlying connection. We tested several functionalities in the system based on a Sony Ericsson P900 smartphone running Symbian OS and connected via Bluetooth and GPRS (T-Mobile). Figure 2 shows the call sequence diagram for a one-way call to access the smartphone's camera. Table 1 measures the performance estimate at each link shown in Figure 2.

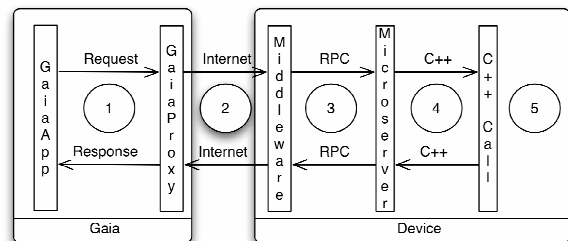


Figure 2: Call sequence for an operation

Table 1: Performance Numbers

Stage	Description	Latency	Notes
1	Gaia proxy request	50 ms	LAN
2 (GPRS)	Proxy->Mobile	6000 ms	GPRS
2 (BT)	Proxy->Mobile	1500 ms	BT
3	J2ME-> Microserver	< 1 ms	Local
4	Microserver C++ call	< 1 ms	Library
5	Camera snapshot	2500 ms	Camera
Total	Over GPRS	8550 ms	GPRS
Total	Over Bluetooth	4050 ms	BT

8. Related Work

Our proxy-based middleware framework approach is

similar to Berkeley's Ninja project [19], where mobile devices connect as thin clients to powerful servers. In general, we differ from most other projects because our client provides a hardware compatibility layer that optimizes access to device-specific functionality through a standardized interface. The microserver is essentially an RPC wrapper around a set of device functionalities. CORBA [20], Java RMI [21], XMLRPC [22] and SOAP [23] are fairly heavy to use on mobile devices with limited resources.

9. Conclusion

Our middleware framework extends the functionality of the virtual machine in a ubiquitous environment. We provide a facility to export native device functionality into active spaces. Services are addressed through Gaia rather than through direct manipulation. As a result, we inherit the benefits of an active space application, such as rich security policies and contextual information. Custom applications and services can access our components through the proxy to securely collect private information and infer situational information to adapt the system to serve users better.

To allow dissemination of our middleware to the widest array of devices, we used the portable J2ME programming language with our microserver extensions. Through our packaging mechanisms, we can automatically target any platform through a single unified executable. Finally, we no longer require static client applications to access Gaia. With the microserver approach, we can increase the extensibility by adding components dynamically at runtime. This additional flexibility will enable a whole new class of rich multimedia ubiquitous applications to emerge.

10. References

- [1] M. Roman and R. H. Campbell, "GAIA: Enabling Active spaces," presented at 9th SIGOPS European Workshop, Kolding, Denmark, 2000.
- [2] M. Roman, C. K. Hess, R. Cerqueira, R. H. Campbell, and K. Narhstedt, "Gaia: A Middleware Infrastructure to Enable Active spaces," *IEEE Pervasive Computing Magazine*, vol. 1, pp. 74-83, 2002.
- [3] J. Bresler, J. Al-Muhtadi, and R. Campbell, "Gaia Mobility: Extending Active space Boundaries to Everyday Devices," presented at The International Workshop on Smart Appliances and Wearable Computing 2005 (IWSAWC 2005) in conjunction with ICDCS 2005, 2004.
- [4] D. Kramer, "The Java Platform," Sun Microsystems White Paper 1996.
- [5] Sun Microsystems, "MIDP APIs for Wireless Applications," Sun Microsystems White Paper 2003.
- [6] S. Steele, "Accelerating to Meet the Challenge of Embedded Java," ARM Limited White Paper 2001.
- [7] M. Roman and R. H. Campbell, "Providing Middleware Support for Active space Applications," presented at ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil, 2003.
- [8] M. Roman, H. Ho, and R. H. Campbell, "Application Mobility in Active spaces," presented at 1st International Conference on Mobile and Ubiquitous Multimedia, Oulu, Finland, 2002.
- [9] G. E. Krasner and S. T. Pope, "A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System," *Journal of Object Oriented Programming*, vol. 1, pp. 26-49, 1988.
- [10] J. Al-Muhtadi, A. Ranganathan, R. Campbell, and M. D. Mickunas, "Cerberus: A Context-Aware Security Scheme for Smart Spaces," presented at the First IEEE Annual Conference on Pervasive Computing and Communications (PerCom 2003), Fort Worth, Texas, 2003.
- [11] G. Sampemane, P. Naldurg, and R. Campbell, "Access Control for Active spaces," presented at the Annual Computer Security Applications Conference (ACSAC), Las Vegas, NV, 2002.
- [12] J. Al-Muhtadi, R. Campbell, A. Kapadia, D. Mickunas, and S. Yi, "Routing Through the Mist: Privacy Preserving Communication in Ubiquitous Computing Environments," presented at International Conference of Distributed Computing Systems (ICDCS 2002), Vienna, Austria, 2002.
- [13] E. Chan, "A Microserver Approach to Mobile Application Design," University of Illinois at Urbana-Champaign, 2004.
- [14] Sun Microsystems, "Java Native Interface Specification. Release 1.1," Sun Microsystems White Paper 1997.
- [15] The Official Bluetooth Website, "Bluetooth." <http://www.bluetooth.com/>.
- [16] J. S. Fritzinger and M. Mueller, "Java Security," Sun Microsystems White Paper 1996.
- [17] B. Borthakur, "Distributed and Persistent Event System For Active spaces," in *Master Thesis in Computer Science*. Urbana-Champaign: University of Illinois at Urbana-Champaign, 2002, pp. 67.
- [18] S. Gribble, M. Welsh, R. Behren, and e. al., "The Ninja Architecture for Robust Internet-Scale Systems and Services," *Computer Networks*, vol. 35, pp. 473-497, 2001.
- [19] S. Gribble, M. Welsh, R. Behren, and e. al., "The Ninja Architecture for Robust Internet-Scale Systems and Services," *Computer Networks*, vol. 35, pp. 473-497, 2001.
- [20] OMG, "CORBA, Architecture and Specification," Common Object Request Broker Architecture (CORBA) 1998.
- [21] Sun Microsystems Inc., "Java Remote Method Invocation (Java RMI)." available at <http://java.sun.com/products/jdk/rmi/>.
- [22] XML-RPC Home Page. <http://www.xmlrpc.com>.
- [23] SOAP Version 1.2 Part 1: Messaging Framework. W3C, 2003.
- [24] A. Ranganathan, J. Al-Muhtadi, S. Chetan, R. Campbell, and M. D. Mickunas, "MiddleWhere: A Middleware for Location Awareness in Ubiquitous Computing Applications," presented at 5th International Middleware Conference 2004.