Gaia: A Development Infrastructure for Active Spaces*

Renato Cerqueira† Christopher K. Hess Manuel Román Roy H. Campbell

Department of Computer Science
University of Illinois at Urbana-Champaign, USA
{rcerq,ckhess,mroman1,rhc}@cs.uiuc.edu

Abstract

In this paper, we present an overview of our research project with Gaia, a development infrastructure for ubiquitous applications. This infrastructure is based on three main elements: a component-based middleware operating system that provides a generic computational environment for ubiquitous computing, an application model that defines a standard mechanism to build ubiquitous applications, and a scripting language that we use to assemble component-based applications and to coordinate activities in ubiquitous computing scenarios.

Keywords: *Ubiquitous Computing, Active Spaces, Application Models, Scripting Languages, Component-based Development.*

1. Introduction

Ubiquitous computing environments encompass a spectrum of computation and communication devices that seamlessly augment human perception and activity with digital information, processing, and analysis. Large numbers of computing devices provide new functionality, enhance user productivity, and ease everyday tasks. In home, office, and public spaces, ubiquitous computers will unobtrusively augment work or recreational activities with information technology that optimizes the environment for people's needs. Users will have anytime/anywhere access to information, the network, and computational resources.

Within this ubiquitous computing world, applications should make effective use of the available resources to support the activities of users. Because users and computing devices move, a ubiquitous application should be adaptable from one physical location to another, overlayed on the actual physical devices and geometry of the physical space.

*This research is supported by a grant from the National Science Foundation, NSF CCR 0086094 ITR and NSF 99-72884 EQ.

†This author is supported by a grant from CAPES-Brazil.

However, the heterogeneity, mobility, and sheer number of devices make the application development vastly more complex. Applications may have the choice of a number of input devices, such as mouse, pen, or finger; and output devices, such as monitor, PDA screen, wall-mounted display, or speakers. Devices and software components can be added to or removed from the system at anytime, and thus applications might need to adapt dynamically to new execution environments. Many aspects of the computational, physical and behavioral contexts that surround the user can impact on the behavior of her applications, due to resource availability, security, or privacy requirements.

Several approaches have been proposed for interacting with ubiquitous computing environments that are customized for particular scenarios or targeted toward a specific type of application. We argue for a general operating system for ubiquitous computing environments, which exports and coordinates the resources contained in a physical space, and therefore facilitates the development of ubiquitous applications. A computing environment for a physical room, operating theater, building, or city must have some form of operating system that organizes that environment to simplify the management of resources, the coding of applications, the identification and authentication of users, the provision of services, and the reuse of software. In the *Gaia* project, we define a generic computational environment that converts physical spaces and their ubiquitous computing devices into a programmable computing system, called an *active space*. An active space is analogous to traditional computing systems; just as a computer is viewed as one object, composed of input/output devices, resources and peripherals, so is an active space.

Traditional operating systems manage the tasks common to all applications; the same management is necessary for physical spaces. However, instead of managing resources within a single computer, an operating system for active spaces manages the computational resources within a physical space.

Our research with *Gaia* is focused on the study of issues related to the design and implementation of such an operat-

ing system that facilitates the development of applications that run in the context of a space. Section 2 presents an overview of *GaiaOS*, an operating system for active spaces that we have developed. Section 3 describes our application model that defines a standard framework for creating ubiquitous applications that run in the context of an active space. Section 4 explains how we coordinate components and tasks in the system through the use of a flexible scripting language. Finally, Section 5 offers some concluding remarks.

2. *GaiaOS*

GaiaOS is a component based meta-operating system, or middleware operating system [1], that runs on top of existing systems, such as Windows2000, WindowsCE, and Solaris. The communication infrastructure that we are using in *GaiaOS* to integrate different platforms is based on CORBA. Currently, we are using the TAO ORB [14] in Unix workstations and Windows desktop machines, and the UIC ORB in handheld devices [13].

GaiaOS is composed of two main parts. At the lowest level, the *Unified Object Bus* provides tools to manipulate uniformly heterogeneous components running in the system [11]. This bus is the foundation on which the remaining of *GaiaOS* components rely. The *GaiaOS Kernel* includes essential services that implement the core functionality of the system, including entity discovery, a component repository, event distribution, naming, data storage and manipulation, and security. The following sections describe briefly these two main parts of *GaiaOS*. A more detailed description of *GaiaOS* can be found in [12].

2.1 Unified Object Bus

Active spaces are highly heterogeneous by definition. They include a variety of hardware devices and software protocols. However, to export a space as a programmable entity, such heterogeneity must be hidden. The *Unified Object Bus* (UOB) [11] is responsible for providing common tools to manipulate the life cycle of components running in an active space, such as component creation, inspection, deletion, and naming. The UOB can manipulate different component types (e.g., CORBA, native program files, and scripts) and provides an open architecture to incorporate new component models.

The UOB defines four basic abstractions, which include the *Unified Component*, *Component Manager*, *Component Container*, and *UOBHost*. Unified Components are the basic elements of the UOB. These components follow a common naming scheme and their life cycle can be dynamically managed regardless of their component model and location. The Component Manager exports the interface to manipulate the life cycle of components, and encapsulates the functionality to integrate different component models; therefore, there is one Component Manager for each integrated component model. The Component Container provides the execution environment for components, and exports functionality to manage the dependencies of the components it contains. Finally, a UOBHost is any device capable of hosting the execution of components. These UOBHosts export functionality to create and delete component containers, as well as to create components in particular component containers.

2.2 The Kernel Services

The kernel contains the minimum required services to bootstrap *GaiaOS* in any arbitrary space. A fundamental service is the *Event Manager*, which is used to distribute information among components, while maintaining loose coupling. Applications can register to specific event channels to be notified of information or changes in the environment. We use this service to support context-aware applications in Gaia. The *Discovery Service* uses the event manager to track software components, people, and physical entities present in a space.

The devices, services, applications, and users currently active in a space are stored in the *Space Repository*. The Space Repository exports an interface to search for specific entities based on constraints such as location, service type, and resource availability.

Applications access data through the *Data Object Service* [4], a dynamically typed file system that supports content adaptation, customized data access, and location awareness. Personal storage may reside on remote desktop machines or mobile special-purpose devices. A user may define personal storage that can be incorporated into a space when she is physically present.

Security concerns in *GaiaOS* include authentication, access control, secure dynamic loading of components, secure tracking, and location privacy. *GaiaOS* employs an *Authentication Service*, which issues credentials for user identity verification. These credentials enable the *Access Control Service* to provide discretionary, mandatory and role-based access control. More details about the security support of *GaiaOS* can be found in [16].

3. The *Gaia* Application Model

The *Gaia Application Model* [10] provides a standard framework to build applications for ubiquitous computing scenarios, and proposes a new way of using applications, based on the ubiquitous computing paradigm. This application model is based on the traditional Model-View-

Controller (MVC) paradigm, augmented with extensions to account for the characteristics of ubiquitous computing environments.

3.1. Traditional MVC

MVC [8] defines a modular design that models the behavior of interactive applications by clearly encapsulating the model of the application domain (model), the visualization of the model (view), and the mechanisms to interact with the model (controller). This modular architecture simplifies the task of modifying and extending applications, as well as reusing specific components.

A model has one or more views attached, which are responsible for displaying the data in some particular way. The benefit of this separation between model and view is that the same model can be rendered in different ways. The model explicitly knows about the assigned views and is responsible for updating them whenever a change in the model state is detected. The final element required is the controller, which allows users to interact with the application model through any of the assigned views. A controller stores a model-view pair as well as a reference to the input sensor (e.g., mouse, keyboard and pen) that captures input events generated by users. The result of an input event depends on the associated view and the actions derived from the control mechanism are automatically sent to the views associated with the model.

3.2. MVC for Active Spaces

The concepts defined by the traditional MVC are valid for any interactive application, regardless of the specific environment where those applications run. An application has a model, externalizes a representation of the model so users can perceive it, and has mechanisms to modify the state of the model. However, most of the existing implementations of MVC are customized for traditional application environments, and therefore it is difficult to reuse them in the context of active spaces. In *Gaia*, we adopted an application model that maps the MVC pattern into active space environments. This new model takes into account many issues related to ubiquitous computing: the variety of interaction devices; contextual properties associated to the user and the space where the application runs; automatic model-view data type adaptation; mobility of the view, model and controller; and applications running on behalf of a user or a space, instead of in the context of a particular device. Our application model standardizes the way in which applications for active space environments are designed, built, and assembled. From the user perspective, this application model defines how to use and customize applications. These applications run in the physical space inhabited by users, and a user interacts with the space as a single entity and not as a collection of individual unrelated digital devices and services.

The traditional MVC presents the state of the model to users by means of views. Views are responsible for rendering the state in some visual form. In ubiquitous applications, presenting the model's state to the user does not necessarily imply rendering it. The model can be externalized in any possible way that affects the senses of a user, such as sounds, smells, and vibrations.

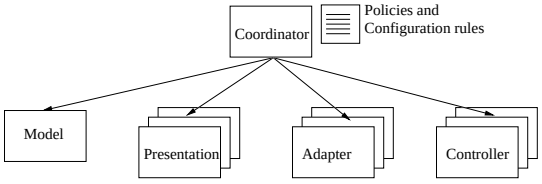

Figure 1. Our application framework consists of the Model, Presentation, Adapter, Controller, and Coordinator.

Based on our application model, we implemented a framework to support the development of ubiquitous applications, called Model-Presentation-Adapter-Controller-Coordinator (MPACC) [10]. This framework defines five main components: (1) *Model*, (2) *Presentation*, (3) *Adapter*, (4) *Controller*, and (5) *Coordinator*. The Model is the implementation of the application's central structure, which normally consists of data and a programmable interface to manipulate the data. The Presentation is the physical externalization of the model that allows users to perceive it through one or more senses. The Controller exports mechanisms to modify the state of the model. However, unlike the standard MVC controller, the controller defined by MPACC coordinates not only input devices, but any source of physical and digital context that can affect the application. The Adapter is the component responsible for adapting the format of the model data to the characteristics of an arbitrary output device, in essence providing "impedance matching". The Coordinator is a meta-level component that manages the application composition and applies adaptation policies, based on functional and non-functional application aspects. The Coordinator stores references to the components that compose the application, as well as policies regarding adaptation, customization and mobility of the application. Figure 1 illustrates a schematic diagram of our application framework.

4. Active Space Coordination

In addition to its services and application framework, we provided *GaiaOS* with support to configure and coordinate applications and OS components in an easy manner. To provide that kind of support, we chose a mechanism based on a high-level scripting language. As in other operating systems, like Unix with Bourne Shell and MS-Windows with the Windows Script Host facilities, our scripting language is able to automate management and configuration tasks. For instance, scripts can coordinate the bootstrap process, configure the *GaiaOS* services, and reconfigure the system in response to changes in its context.

However, a suitable scripting language can play different roles in a system like *GaiaOS*, in addition to coordinating tasks and describing service's configurations. Because *GaiaOS* is a component-based operating system, a language with a suitable set of abstractions and mechanisms must be used to connect the available components in order to compose new services and applications, similar to a *composition language* [15].

A scripting language also brings an interesting design alternative to the development process of ubiquitous applications: It greatly improves the support for testing, rapid prototyping, and dynamic configuration, as we can load and test new design alternatives for an application in a quick and simple way. For instance, a component can be tested by a set of scripts, while other components that are not implemented yet, but are required by this component, can be provided by prototypes implemented with the scripting language. Components implemented with a scripting language can be dynamically modified and extended without compiling or linking phases, and so, without interrupting their services. With an interpreted language, it is easy to send code across a network, which allows the system to do remote or interactive modifications and extensions to distributed components and services.

4.1. Lua and LuaOrb

To provide scripting facilities, *GaiaOS* uses the LuaOrb scripting tool [2, 3]. LuaOrb is programming tool for component-based systems, that supports dynamic gluing of components from different component infrastructures, such as CORBA, COM and Java. LuaOrb extends the interpreted language Lua [6, 7] with a set of facilities to access component infrastructures, and it can perform the following tasks at run-time: (1) the identification of new component types and the integration of their instances into a dynamically assembled application, (2) the implementation of new components using Lua, and (3) the extension and adaptation of the available components also using Lua.

Currently, LuaOrb implements language bindings between Lua and CORBA, COM, and Java. Instead of using the traditional mechanism based on statically generated stubs, the design of these language bindings are strongly based on reflective mechanisms provided by Lua and the component infrastructure: interface introspection, dynamic call assembly, and dynamic implementation definition [5]. The LuaOrb's bindings use a mechanism called *generic proxy* to offer access to the available components, and use another mechanism, called *generic adapter*, to support dynamic implementation of components using Lua. By connecting generic proxies and generic adapters, we can create dynamic bridges between different component systems [2]. Using these dynamic bridges, components of different component systems can seamlessly interoperate. Therefore, Lua acts as a unifying glue language, wherein we can write code at run-time that freely use and mix components from different component systems.

In addition to the interoperability and component-based programming facilities that LuaOrb provides, the use of a language like Lua is very suitable for the scenarios in which we are interested in applying *GaiaOS*. Because of its fast language engine with a small memory footprint and completely implemented in ANSI C, Lua has been used in many different platforms, including embedded systems and handheld devices [9]. Therefore, *GaiaOS* can provide scripting facilities even for resource-constrained devices.

4.2. Integrating LuaOrb with Gaia

LuaOrb performs different tasks in *GaiaOS*. We started using it to test some *GaiaOS* components and to implement prototypes of new services. However, we are using it now to implement some more specific features for active spaces.

We are using the data description facilities of Lua to provide a declarative mechanism, instead of a procedural one, that specifies a set of components that should be created in an active space. We have been applying this facility to coordinate the *GaiaOS* bootstrap, to start demonstration scenarios, to perform application tests, and to implement login scripts for users. Due to the unifying character of LuaOrb, we can create and use in these declarative scripts components from different component systems.

We are also using LuaOrb to implement two important types of components in *Gaia*. The first one is a component type that we call *context controller*. This kind of component captures events related to context information from the *Gaia* Event Manager, and triggers specific actions when certain conditions are met. Using these components, we can easily implement context translators that listen for specific events and translate, aggregate, or distill them to generate new events. The second type of component is the Coordinator component of our application framework. Using the composition and scripting facilities of LuaOrb, we can eas-

ily implement this kind of component, which glues other components, coordinates their activities, and defines configuration policies.

LuaOrb works together with the *Gaia* application framework and the UOB to support dynamic composition and adaptation of ubiquitous applications: While LuaOrb offers a mechanism to command dynamic reconfigurations of component connections, the UOB infrastructure and the MPACC framework offer the means that allow the reconfiguration of component connections (*re-wiring*).

5. Final Remarks

In this paper, we presented our research project with *Gaia*. With the infrastructure that *Gaia* currently provides, we can coordinate the resources contained in a physical space and create ubiquitous applications by dynamically connecting available components. *GaiaOS* converts physical spaces and their ubiquitous computing devices into a programmable computing system. The *Gaia* application framework allows us to partition applications, dynamically place application components, adapt to different devices characteristics, react to contextual changes, and attach/detach application components. The LuaOrb scripting tool allow us to automate management and configuration tasks, describe and create ubiquitous computing scenarios, test components, and prototype new applications.

To validate our infrastructure, we have developed a presentation application that operates on multiple displays and coordinates all the resources available in the space to conduct presentations. In this application, we used *Gaia* to integrate infra-red devices, X10 devices, different user identification devices, and software components from different component systems, such as CORBA and COM.

Although we still have many issues to investigate related to application development in ubiquitous computing environments, our preliminary results show us that offering a common infrastructure can significantly reduce the complexity of this task.

Resources and further information about *Gaia* are available at `http://choices.cs.uiuc.edu/gaia/`.

References

[1] 2K Research Team. 2K: A Component-Based Network-Centric Operating System for the Next Millennium. `http://choices.cs.uiuc.edu/2k`, 1998.

[2] R. Cerqueira, C. Cassino, and R. Ierusalimschy. Dynamic component gluing across different componentware systems. In *International Symposium on Distributed Objects and Applications (DOA'99)*, pages 362–371, Edinburgh, 1999. OMG, IEEE Press.

[3] R. Cerqueira, R. Ierusalimschy, and N. Rodriguez. The LuaOrb Project. Project home page: `http://www.tecgraf.puc-rio.br/luaorb`, 1999.

[4] C. Hess, R. Campbell, and D. Mickunas. The role of users and devices in ubiquitous data access. Technical Report UIUCDCS-R-2001-2226 UILU-ENG-2001-1733, University of Illinois at Urbana-Champaign, May 2001. Avaliable at `http://devius.cs.uiuc.edu/gaia/papers/dos-tech01.pdf`.

[5] R. Ierusalimschy, R. Cerqueira, and N. Rodriguez. Using reflexivity to interface with CORBA. In *International Conference on Computer Languages 1998*, pages 39–46, Chicago, 1998. IEEE, IEEE Press.

[6] R. Ierusalimschy, L. Figueiredo, and W. Celes. Lua—an extensible extension language. *Software: Practice & Experience*, 26(6):635–652, 1996.

[7] R. Ierusalimschy, L. Figueiredo, and W. Celes. *Reference Manual of the Programming Language Lua version 4.0*. Rio de Janeiro, Brazil, 2000. (available at `http://www.lua.org/ftp/refman.ps.gz`).

[8] G. E. Krasner and S. T. Pope. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System, 1988. Parc Place Systems Inc, Mountain View.

[9] Projects using Lua. http://www.lua.org/uses.html.

[10] M. Román and R. H. Campbell. A model for ubiquitous applications. Technical Report UIUCDCS-R-2001-2223 UILU-ENG-2001-1730, University of Illinois at Urbana-Champaign, May 2001. Avaliable at `http://devius.cs.uiuc.edu/gaia/papers/appmodel-tech01.pdf`.

[11] M. Román and R. H. Campbell. Unified Object Bus: Providing support for dynamic management of heterogeneous components. Technical Report UIUCDCS-R-2001-2222 UILU-ENG-2001-1729, University of Illinois at Urbana-Champaign, May 2001. Avaliable at `http://devius.cs.uiuc.edu/gaia/papers/uob-tech01.pdf`.

[12] M. Román, C. K. Hess, A. Ranganathan, P. Madhavarapu, B. Borthakur, P. Viswanathan, R. Cerqueira, R. H. Campbell, and M. D. Mickunas. GaiaOS: An infrastructure for active spaces. Technical Report UIUCDCS-R-2001-2224 UILU-ENG-2001-1731, University of Illinois at Urbana-Champaign, May 2001. Avaliable at `http://devius.cs.uiuc.edu/gaia/papers/gaia-tech01.pdf`.

[13] M. Román, F. Kon, and R. Campbell. Reflective middleware: From your desk to your hand. *IEEE Distributed Systems Online*, 2(5), July 2001. Avaliable at `http://computer.org/dsonline/0105/features/rom0105_1.htm`.

[14] D. C. Schmidt. *Real-time CORBA with TAO (The ACE ORB)*, 1999. (`http://www.cs.wustl.edu/~schmidt/TAO.html`).

[15] J.-G. Schneider and O. Nierstrasz. Components, scripts and glue. In L. Barroca, J. Hall, and P. Hall, editors, *Software Architectures – Advances and Applications*, pages 13–25. Springer, 1999.

[16] P. Viswanathan, B. Gill, and R. H. Campbell. Security architecture in gaia. Technical Report UIUCDCS-R-2001-2215 UILU-ENG-2001-1720, University of Illinois at Urbana-Champaign, May 2001. Avaliable at `http://devius.cs.uiuc.edu/gaia/papers/llncs.pdf`.