Gaia: A Middleware Infrastructure to Enable Active Spaces

Revised Paper #20 (2nd Revision)

7/1/2002

المنارة للاستشارات

www.manaraa.com

Gaia: A Middleware Infrastructure to Enable Active Spaces[1]

Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganat, Roy H. Campbell, Klara Nahrstedt
{mroman1, ckhess, rcerq, ranganat, rhc, klara}@cs.uiuc.edu
Digital Computer Lab
University of Illinois at Urbana-Champaign

Abstract

We envision a future where people's living spaces are interactive and programmable. Users interact with their offices, homes, cars, malls and airports to request information, benefit from the resources available, and configure the habitat's behavior. Data and tasks are always accessible and are mapped dynamically to convenient resources present in the current location. Users may extend the habitat with personal devices that seamlessly integrate with the environment. Such user-oriented interactive environments may require a novel software infrastructure to operate their resources, sense context properties, and assist in the development and execution of applications. In this article, we present an experimental middleware infrastructure called Gaia that we have used to prototype the resource management of and provide the user-oriented interfaces for such physical spaces populated with network-enabled computing resources. To limit the scope of our research, we focus on physical spaces used for teaching; classrooms, offices, and lecture rooms. The system described in this paper is derived from a series of experiments starting in 1996. We show how, by applying the concepts of a conventional operating system to middleware, we can manage the resources, devices and distributed objects in a room, building, or physical space, how a distributed extension of the model–view–controller that is use in personal computers simplifies and structures practical applications for these environments, and how, by driving context-sensitivity into its data storage mechanisms, the system can help satisfy the requirements for user-centricity and mobility.

1. Introduction

Pervasive computing environments encompass a spectrum of computation and communication devices that seamlessly augment human thought and activity with digital information, processing, and analysis to provide an observed world that is automated and enhanced by the behavioral context of its users. Large numbers of heterogeneous computing devices provide new functionality, enhance user productivity, and ease everyday tasks. In home, office, and public spaces, ubiquitous computing will unobtrusively augment work or recreational activities with information technology that optimizes the environment for people's needs.

The motivation behind our research is the lack of a suitable software infrastructure to assist us in the development of applications for ubiquitous computing habitats or living spaces. Figure 1 presents our prototype ubiquitous computing environment. In this environment, users interact with a number of devices simultaneously, register their own devices as a resource of the environment, require application adaptation according to changes in the environment, access data located in remote spaces, and suspend applications and restart them later either in the same place or in a different one. Homes, offices, and meeting rooms capable of sensing user actions and equipped with a large variety of devices will assist users with different tasks. We refer to these ubiquitous computing environments as Active Spaces, an extension to physical spaces. Physical spaces (Figure 2a) are geographic regions with limited and well defined physical boundaries, containing physical objects, heterogeneous networked devices, and users performing a range of activities. We define an Active Space (Figure 2b) as a physical space coordinated by a responsive context-based software infrastructure that enhances the ability of mobile users to interact and configure their physical and digital environment seamlessly. A requirement of Active Spaces is to support the development and execution of user-centric mobile applications.

[1] This research is supported by the National Science Foundation grant NSF 98-70736, NSF 9970139, and NSF infrastructure grant NSF EIA 99-72884

المنارة للاستشارات
www.manaraa.com

15 Pentium IV at 1.2GHz
4 NEC HD Plasma Displays (61")
1 Sharp HD Projector
2 *Sound Web* Sound system
4 InfraredBeacons
2 iButton detectors
5 Touchscreen displays
4 Badge detectors
3 Compaq iPaq HandHeld devices
1 Gbit Ethernet network
1 Wavelan wireless network at 11Mbps
X10 appliance controllers

Figure 1. Experimental Active Space

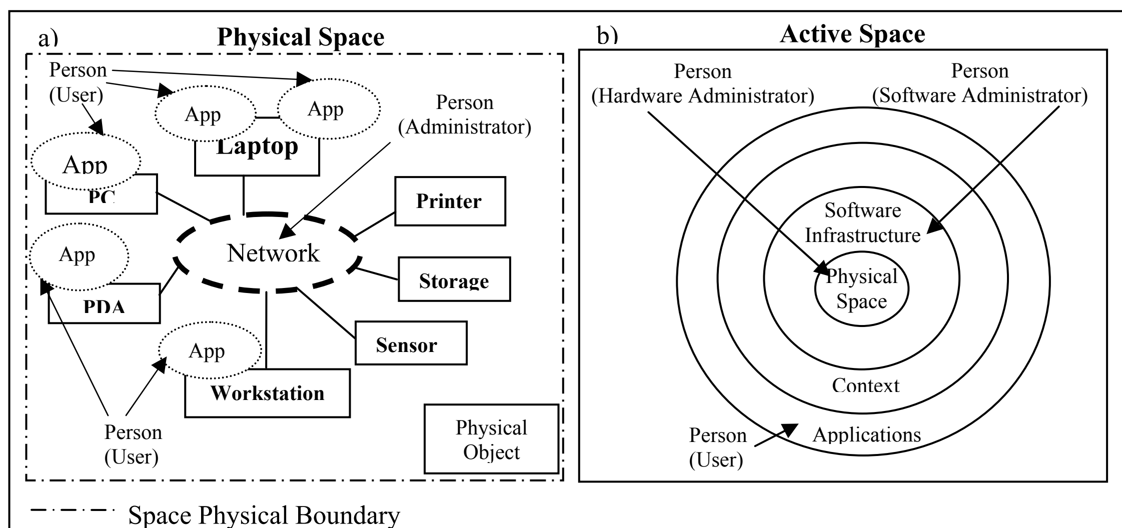

Figure 2. Physical and Active Space

1.1. Our Vision

We believe that active spaces will become commonplace. People will interact with their physical habitats to customize their behavior, gain access to a greater amount of information, and benefit from the resources contained in the space. In this scenario, user data and applications are not confined to any one active space; they are associated with users using the notion of a "session". This allows users to move across different active spaces and have their data and applications always available. When a user enters an active space, their sessions are dynamically mapped to the active space resources. Users can define different sessions and can activate and suspend them as required. We refer to the collection of sessions associated to a user and independent of particular active spaces as the *user virtual space.* This user virtual space requires support to locate resources available in the user's environment and to map the sessions to the existing resources. The user-centric mobile application requirement of active spaces creates problems in our traditional approaches to building computer software infrastructure leading to a post-PC era of system software design.

Active spaces such as the one depicted in Figure 1 challenge existing assumptions for traditional PC applications. As observed by Marc Weiser [1], the problem raised by ubiquitous computing is to develop systems that vanish into the background. In an active space there is no longer the notion of a one-to-one relationship between a user and the interfaces of keyboard, mouse and display. Indeed, the complexity of ubiquitous applications encourages a relationship between a user and an active space. Active space system

المنارة للاستشارات
www.manaraa.com

software support should simplify application programming and execution. In a similar manner to the role that operating systems play in supporting traditional PC applications, active space applications need support to access and operate the resources contained in the space that hosts their execution.

1.2. Our Approach

We present in this article Gaia OS, a meta-operating system [2, 3] that aims at supporting the development and execution of portable applications for active spaces. Gaia is a distributed middleware infrastructure that coordinates software entities and heterogeneous networked devices contained in a physical space. Gaia exports services to query and utilize existing resources, to access and use current context, and provides a framework to develop user-centric, resource-aware, multi-device, context-sensitive, and mobile applications.

We believe that extending the concepts of traditional operating systems to ubiquitous computing spaces simplifies the management of these spaces and the development of applications. The main contribution of Gaia is not in the individual services, but instead, in the interaction of these services. This interaction allows users and developers to abstract ubiquitous computing environments as a single reactive and programmable entity instead of a collection of heterogeneous individual devices. The description we present in this article is an overview of the overall Gaia architecture. This description focuses on Gaia as a complete system instead of trying to give all details about Gaia individual services.

Sidebar 1 – Gaia OS Evolution

Gaia OS is the result of six years of research on reflective middleware and meta-operating systems, middleware for handheld and embedded devices, and ubiquitous computing. Previous to the Gaia project, our group developed the 2K meta-operating system [2, 4], a reflective middleware operating system built on top of traditional operating system (e.g. Windows, Linux, Solaris, and PalmOS). 2K was strongly influenced by previous research on reflective middleware [5-7] and was built on top of a modified version of TAO [8], the pattern-based CORBA ORB from Douglas Schmidt et al. [9]. 2K hides device and operating system heterogeneity and can adapt dynamically to changes in the environment while maintaining the integrity of the overall system. Users in 2K interact with the system using different devices, therefore eliminating the one-to-one user-to-device traditional mapping. Individual devices in 2K become portals to the system. Following this approach, we started a new line of research to study how to integrate resource-limited mobile devices such as handheld and embedded devices into distributed computing environments [10]. These devices use middleware to interoperate with 2K and leverage the functionality provided by 2K services. As part of this research we developed an adaptable middleware prototype customized for handheld devices [11], which evolved into a fully reconfigurable middleware infrastructure [12]. This middleware allows bi-directional interaction between the handheld devices and the meta-operating system. As a result of our previous research, and influenced by the research of Georgia Tech [13-15], MIT [16], and Berkeley [17] on ubiquitous computing, we created Gaia OS [3], a meta-operating system customized for physical spaces that supports the development of applications customized to these environments. Gaia OS provides a standard API that abstracts the complexity and heterogeneity associated to ubiquitous computing environments. The explicit binding between Gaia OS and physical spaces requires new services (not present in 2K) to take into account issues such as the context of the space and the detection of resources added to and removed from the space.

3. Gaia: an Active Space System Software Infrastructure

Gaia provides support for mobile user-centric active space applications. It manages the resources and services of an active space. It provides services for location, context, and events, and repositories with information about the active space. The system is built as a distributed object system. Figure 3 shows the three major building blocks of Gaia: the Gaia Kernel, the Gaia Application Framework, and the Applications

The Gaia Kernel contains a management and deployment system for distributed objects and an interrelated set of basic services that are used by all applications. The Component Management Core dynamically loads, unloads, transfers, creates, and destroys all the components and applications of Gaia. Gaia components are distributed objects and require communication middleware to support remote interaction. The current

المنارة للاستشارات
www.manaraa.com

implementation of Gaia uses CORBA [18]; however, it is possible to port Gaia to other communication middleware architectures including SOAP, RMI, or customized implementations. CORBA provides a stable infrastructure for distributed object interaction. However, the dynamism and heterogeneity of active spaces require extensions to deal with issues such as soft-state to handle crashing components and resources added to and removed from the space. Gaia's five basic services are the Event Manager Service, Presence Service, Context Service, Space Repository Service, and Context File System. The Gaia software infrastructure implements a bootstrap mechanism that initiates the execution of the Gaia Kernel in any arbitrary physical space. Some of the services are built on top of existing middleware services (e.g. Event Manager), while others are extensions to the communication middleware (e.g. Presence Service).

Gaia's applications use a set of component building blocks, organized as the Gaia Application Framework, to support applications that execute within an active space. The framework provides mobility, adaptation, and dynamic binding. The functionality permits commercial off the shelf as well as new applications to run in the active space. Active Space Application layer contains applications and provides functionality to register, manage, and control these applications through the Gaia Kernel services.

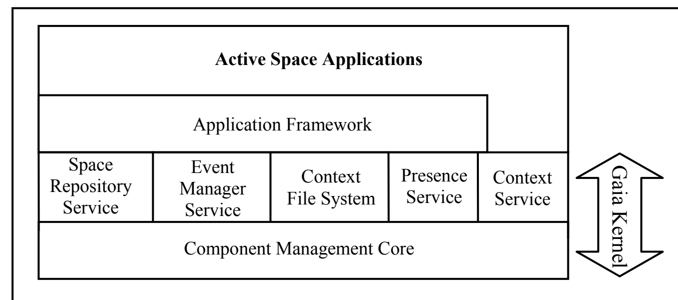

Figure 3. Gaia Architecture

3.1 Gaia Kernel

Gaia applications are component-based, distributed and mobile, and therefore require support for remote component execution and management. The Component Management Core (CMC) provides Gaia with functionality for component creation, destruction, and uploading. Remote execution nodes register with the active space and host the execution of Gaia components. The Gaia Kernel is composed of five services built as Gaia components which are described below in more detail.

a) Event Manager

Active spaces are highly dynamic execution environments that require a flexible mechanism to expose changes in their current state. These changes include components starting, applications moving, users entering and leaving, component beaconing updates, and contextual changes. The Event Manager Service is responsible for event distribution in the active space and implements a decoupled communication model based on suppliers, consumers, and channels. Each channel has one or more suppliers that provide information to the channel and one or more consumers that receive the information. The event manager has a single entry point and one or more event channel factories. Each event factory is responsible for creating event channels with a specific behavior, e.g. high speed events, or persistent events. Gaia defines a default set of event channels to notify interested Gaia components about new services, applications, people, errors, and component heartbeats. Applications can also define their own event channels for application state changes. The event service allows applications to tap into event channels to learn about changes in the system.

The event manager is particularly convenient to decouple information suppliers from information consumers, therefore increasing system reliability. Suppliers and consumers can fail without disrupting the system. A crashing supplier can be automatically replaced with a replica that continues delivering messages to its assigned channel without the consumers being aware of the failure. Our current implementation of the event manager uses the CORBA Event Service [18] as the default event factory. More details of the event manager service can be found in [19].

المنارة للاستشارات
www.manaraa.com

b) Context Service

Gaia applications may use context information to adapt their behavior to accommodate user behaviors and activities[20]. Incorporating context into the infrastructure facilitates the adaptation of the computing environment to the needs of human users [1][2][3][4]. Our context service allows applications to query and register for particular context information so that they may adapt to their environment. The context infrastructure consists of a number of components, called context providers that provide information about the current context. These include sensors that track the location of people, the conditions within a room (e.g. temperature and sound) and other external conditions, such as weather and current stock prices. In addition, we also have components that can infer certain higher-level contexts based on sensed information. For example, we have a component that deduces the kind of activity going on in a room (i.e., meeting, presentation, or movie screening) based on who is in the room, which applications are running, and other cues. There exists a registry that maintains a list of the different context providers available and allows applications to find the providers that supply the contexts in which they are interested.

We use a model for context that is based on first order logic and boolean algebra, which allows us to easily write various rules to describe context information. These rules may be a combination of lower level context information. We represent context through a 4-ary predicate, whose structure is borrowed from a simple clause in the English language of the form <subject><verb><object>. An atomic context predicate is defined in the following way: Context(<ContextType>, <Subject>, <Relater>, <Object>). The Context Type refers to the type of context the predicate is describing, the Subject is the person, place or thing with which the context is concerned, and the Object is a value associated with the Subject. In our implementation, the ContextType is mapped to an event channel. The Relater relates the Subject and the Object such as a comparison operator (=, >, or <), a verb, or preposition. Some example context predicates are: Context(location, chris, entering, room 3231); Context(temperature, room 3231, is, 98 F); Context(social relationship, venus, sister, serena); Context(stock quote, msft, >, $60); Context(printer status, srgalw1 printer queue, is, empty); Context(time, , Is, 12:00 01/01/01). In some cases, one or more elements of a predicate may be empty (e.g., the Time context). It is possible to construct more complex contexts by performing first order logic operations such as quantification, implication, conjunction, disjunction, and negation of context predicates. One example of a rule is Context(Number of people, Room 2401, >, 4) AND Context(Application, Powerpoint, is, Running) => Context(Social Activity, Room 2401, Is, Presentation).

Ideas behind our context infrastructure have been inspired from the Context Toolkit [21]. We structure the expressive power of contexts with first order logic to frame rules and queries and to infer properties involving context using mechanisms that are similar to those of Prolog and other automated theorem provers. High-level context information may be determined from context information, similar to the aggregators of the Context Toolkit. Our system also formalizes the ways in which context information is exchanged between different components in the system and allows us to describe the properties of various components [5].

c) Presence Service

As a resource-aware infrastructure, Gaia needs to maintain updated information about resources present in the active space. The presence service is responsible for detecting digital (e.g., service and application) and physical entities (e.g., furniture and people) present in an Active Space. Our current implementation of Gaia defines four basic types of entities: Application, Service, Device, and Person.

The presence service implements a beaconing mechanism to maintain soft-state about entities present in the space and it is divided into two main subsystems: Digital Entity Presence and Physical Entity Presence. The Digital Entity Presence subsystem detects digital entities; these entities periodically send heartbeats to notify the presence service that they are in the active space. When a digital entity fails to send the heartbeat, the digital entity presence subsystem assumes that the entity is no longer available – either it was stopped or it crashed – and therefore notifies the rest of the space that the entity left.

The Physical Entity Presence subsystem is responsible for detecting physical entities present in the active space. This subsystem uses different types of sensors to proactively detect the presence and, if possible, the location of physical entities. The physical entity presence subsystem implements the beaconing mechanism

المنارة للاستشارات
www.manaraa.com

on behalf of the physical entities, acting as a proxy. The physical entity presence subsystem is implemented as an open infrastructure where different sensor device drivers and algorithms such as [21] can be incorporated. Our presence service differs from other existing context infrastructures (e.g. [21]) in that it provides functionality to detect and maintain soft-state information about software components, devices, as well as people.

d) Space Repository

Active space entities require functionality to learn about the properties of the resources available in the active space. For example, when an application starts executing, it uses the space repository to find appropriate resources (e.g., execution nodes, displays, and speakers). The space repository stores information about all software and hardware entities contained in the space (e.g., name, type, and owner) and provides functionality to browse and retrieve entities based on specific attributes. The space repository learns about entities entering and leaving the active space by subscribing to the channels defined by the presence service. Applications use the space repository during their instantiation to find suitable resources. This level of indirection allows us to describe applications in a generic manner (active space independent) and map them to the resources contained in different active spaces. All active space resources have an XML description associated that lists their properties (e.g., type and location). When new resources are introduced in the active space, the space repository contacts them to obtain the XML description and stores the information. The current version of the space repository uses a CORBA Trader [18] to store the data about entities. We are currently using the constraint query language defined by the trader, although we plan to extend this language in the future with a generic language that could be mapped either to the CORBA Trader constraint language or to standard SQL. For example, the following query: *Category = = 'Device' and Type = = 'Display'* returns a list of all displays present in the active space.

e) Context File System

Active spaces are often designated for specific tasks. The context of these tasks can determine the information that is meaningful and can be used to prune out irrelevant material. Long running processes may not have the luxury of human intervention to locate required data, which may vary over time due to changes in context. If relevant information is known to exist in a particular location, the application is relieved from performing costly searches over the entire collection of data. In addition, users are highly mobile in active spaces and should not be burdened with manually transferring files or data, be it configurations, preferences, or application data from one environment to another. The environment should assist in making personal storage automatically available in the users' present location.

To address the foregoing issues, we have developed a context-aware file system (CFS) that uses context to alleviate many of the tasks that are traditionally performed manually or require additional programming effort. More specifically, context is used to 1) automatically make personal data available to applications, conditioned on user presence, 2) organize data to simplify the location of data important for applications and users, and 3) retrieve data in a format based on the context of user preferences or device characteristics through dynamic data types. CFS constructs a virtual directory hierarchy [22] based on the types of context that have been associated with particular files and aggregates related material. The layout of the directory hierarchy is implemented using a mounting mechanism, where mount points are owned by users and contain context tags. Users may merge their personal mount points into a space to make their data available to applications and other users. CFS is aware of different types of context, which are defined by the context service as well as by the users and applications.

Context is presented as directories, where path components represent context types and values. The file system path syntax uses the 4-ary predicate structure from the context service, where the relater defaults to equality. Context may be attached to files and directories by copying data to a context directory, which associates the particular context to the data. The virtual directory hierarchy forms a simple query language to determine what types of contexts are attached to files. For example, to determine which files have the context of `location == RM2401 && situation == meeting` associated with them, one may enter the `/location:/RM2401/situation:/meeting` directory.

المنارة للاستشارات
www.manaraa.com

CFS uses the current context properties of the environment (e.g., location, time, situation, weather) together with user specified properties to display the correct application data. For example, a seminar application may require all papers that are to be discussed during a seminar. This application may be automatically started when the seminar is started, triggered from a calendar or when the moderator arrives, and therefore must be able to locate the correct files to display. The application simply opens the directory for the current papers, e.g., `/type:/papers/current:`. The file system will use the current location, situation, and time information along with the fact that "papers" are requested to find the correct files for the application. The contents of this directory may automatically change every week, as papers are added and old papers time out. However, from the application point of view, it simply opens the same directory every week and finds the relevant material.

The queries that are performed are not simply a combination of the current context and the application requested material; the space may define a context that is irrelevant to the current task. For example, the context "the weather is sunny" may be meaningless to the seminar application, but may make sense for a tele-presence application. The system resolves this issue by ignoring any context that is valid in the environment, but that is not explicitly associated with the data. Although the context directory structure is viewed as a hierarchy, context directories impose no fixed ordering, resulting in a forest rather than a tree structure; context paths can be traversed in any order.

The CFS architecture is composed of mount and file servers. Each active space has a namespace that is maintained by one mount server. The namespace changes as users physically move in and out of the space. File servers may be located locally or remotely to export storage to the local space. Our servers are implemented at application-level and leverage existing native file systems to access files and directories. More details about the CFS can be found in [23].

3.2 Application Framework

Active spaces entail a user-centric, resource-aware, multi-device, context-sensitive, mobile application model. Applications are partitioned among a group of coordinated devices [24], receive input events from different devices, present their state using different types of devices (e.g., sound system, display, and device to control temperature), and adapt to changes in the environment. Active spaces allow users to decide how to interact with applications using a number of inputs, outputs, and processing devices. However, development of applications for active spaces becomes a challenge. For example, an application may require moving the output data from one display to another, duplicating the output to different displays, transforming a visual representation into speech, and switching from a mouse to a voice input sensor, all of which must be performed providing application consistency guarantees.

We have developed an application framework that provides mechanisms to construct, run or adapt existing applications to active spaces. The framework is composed of a distributed component-based infrastructure, a mapping mechanism, and a group of policies to customize different aspects of the applications. The application framework infrastructure reuses the application partitioning proposed by the traditional Model–View–Controller [25] and introduces new functionality to export and manipulate the bindings of the application components; the mapping mechanism customizes applications to different active spaces; finally, the policies define different sets of rules to customize several aspects of applications including instantiation, mobility, reliability, and composition (number of components and their bindings).

The application infrastructure is composed of four components: model, presentation (generalization of view), controller, and coordinator. The first three components constitute the building blocks for any application. The last component, the coordinator, is responsible for the management of the other three components. The model implements the application logic, the presentation exports the application data, and the controller maps input events (e.g. touch screen events and context changes) into method requests for the model. These input events are generated by input sensors, which are the hardware and software entities that trigger changes in the application. The model, presentation, and controller are strictly related to application domain functionality (e.g., music jukebox functionality), while the coordinator provides the meta-level functionality of the application. Figure 4 illustrates the framework infrastructure.

المنارة للاستشارات
www.manaraa.com

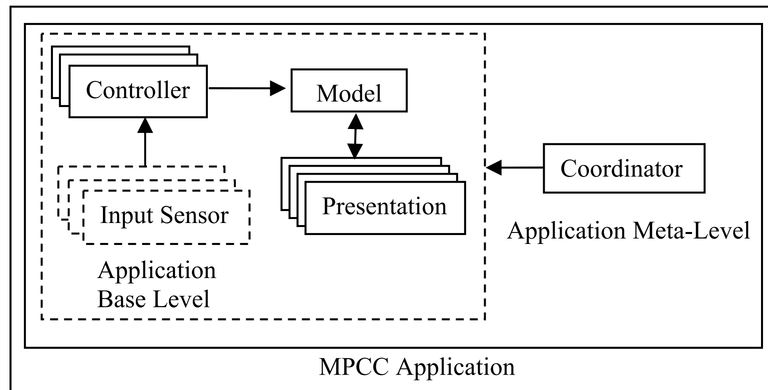

Figure 4. Application Model Infrastructure.

The heterogeneity of active spaces requires a mechanism to customize applications to different scenarios. For example, a calendar application running in an active office may use simultaneously a plasma display to present the appointments for the week, the handheld to display the appointments for the day, and an input sensor running in the desktop PC to enter data. However, the same calendar running in an active car may use the sound system of the car to broadcast information about the next appointment, and an input sensor based on speech recognition to query the calendar as well as to enter and delete data. The mapping mechanism defines two application description files: Application Generic Description (AGD) and Application Customized Description (ACD). The AGD is an active space independent description created by the developer that lists the application components, the minimum and maximum number of instances allowed, and their requirements (e.g., audio output, and Windows OS). The ACD consists of a list of application components, including their associated execution nodes (chosen according to the component requirements) and initialization parameters. The ACD is implemented as a script that coordinates the instantiation and assembly of the different components. The specialization mechanism generates ACDs using an AGD and the space repository service of the target active space. The mapping mechanism uses the space repository to find the appropriate devices and services according to the requirements stored in the AGD.

The application framework infrastructure and the mapping mechanism provide the tools to build and instantiate applications. The policies customize different aspects of the application including reliability, how to react to changes in the environment (e.g., context changes), and how to implement mobility. The application framework relies on policies to address all these issues. Users can define their own policies or can use default policies provided by the framework.

We have introduced four changes to the original MVC to accommodate the requirements for environmental-awareness, application partitioning, context-sensitivity, user-centrism, and mobility. First, we define a new component called presentation that models any output representation, not only graphical as proposed by the MVC view. Second, we generalize the definition of the MVC's input sensor (hardware device) to incorporate software components (e.g., context input sensor). Third, we introduce a new component called the coordinator to manage the composition of the application components (meta-level). Finally, we generalize the input sensor time-sharing model defined by the MVC into a space-time-sharing model. According to MVC, all applications' views and controllers share the same input sensors (e.g., mouse and keyboard) and therefore the input sensors must be scheduled. Graspable interfaces [26] introduce the concept of space-sharing, where different input sensors are assigned to different functional aspects of the application, therefore avoiding the need for scheduling them. We combine both approaches into space-time-sharing to model the type of applications we consider. For example, a music application running in an active space uses a PDA to control the current song, and speech recognition to control the sound level (space-sharing), however the same space may host a calendar application that uses the PDA to browse appointments, and speech recognition to control the calendar's functionality (time-sharing). Space-time-sharing allows more than one controller and presentation to be active at the same time, which contrasts with MVC where only

المنارة للاستشارات
www.manaraa.com

one controller-view pair can be active at anytime. More details of the application framework can be found in [27].

4. Gaia Management Tools

In this section we describe the scripting language we use in Gaia and the bootstrap mechanism.

4.1 Lua: Gaia's scripting language

Gaia uses a high level scripting language, called LuaOrb [28], to program and configure active spaces and to coordinate the active entities they contain. LuaOrb is based on the interpreted language Lua [29], which simplifies management and configuration tasks and allows for rapid prototyping and testing. The interpreter for Lua is fast and has a small memory footprint, which makes it suitable for resource-constrained devices. LuaOrb implements language bindings between Lua and CORBA, COM, and Java. The ability of LuaOrb to communicate with various component models directly allows it to easily interact with the components in our system. We use Lua to implement the bootstrap algorithm, to instantiate applications, to interact with execution nodes to create components and easily glue them together, and to quickly test components and applications.

Table 1 presents an example script that instantiates and assembles an MP3 application. The script uses the Gaia space repository to obtain a handle to an audio output device (line 1), an execution node for the model (line 2), an execution node for the coordinator (line 3), and a touch screen called plasma 1 for the input sensor. Then, it uses the component management core functionality to create the coordinator (line 5), the model (line 6), the presentation (line 7), and the input sensor (line 8). Finally it assigns the model to the coordinator (line 9) and registers the presentation (line 10) and the input sensor (line 11) with the application using the interface exported by the coordinator.

```
1. local presentationExNode = Gaia.getEntity("Category == 'Device' and Type == 'AudioOut' ")
2. local modelExNode =          Gaia.getEntity("Category == 'Device'  and Type == 'ExecutionNode'
                                               and Name == 'aspc1.uiuc.edu'")
3. local coordinatorExNode =   Gaia.getEntity("Category == 'Device'  and Type == 'ExecutionNode'
                                               and Name == 'aspc2.uiuc.edu'")
4. local inputSensorExNode =  Gaia.getEntity("Category == 'Device'  and Type == 'Touchscreen'
                                               and Name == 'plasma1'")
5. local coordinator =   coordinatorExNode:createComponent("Coordinator", "-name MP3Coordinator")
6. local model =          modelExNode:createComponent("MP3Model", "-name MP3Model")
7. local presentation = presentationExNode:createComponent("MP3Presentation", "-name MP3Player")
8. local inputsensor = inputSensorExNode:createComponent("VCRInputSensor","-name MP3InputSensor")
9. coordinator:setModel(model)
10. coordinator:registerPresentation(presentation)
11. coordinator:registerInputSensor(inputsensor)
```

Table 1. Lua Script Example: Application instantiation and assembly.

Although the same result can be accomplished with languages such as C++ and Java, it requires more code, time, and user effort. Lua effectively simplifies the manipulation and coordination of entities.

4.2 Bootstrap

Gaia implements a bootstrap protocol that interprets a configuration file (Lua script) and starts the kernel services accordingly. The configuration file contains information about the Gaia Kernel services, such as the name of the service, the name of the interface of the service, the Gaia node or nodes that will host the service, the service instantiation policy (i.e., instantiate the service in all Gaia nodes or only in the first available Gaia node), and start parameters. Individual Gaia Kernel services can also specify additional configuration parameters. Currently, the active space administrator is responsible for providing the list of devices contained in the space. However, in the future we expect automatic device discovery by means of different sensor technologies.

المنارة للاستشارات
www.manaraa.com

Figure 5 illustrates a state transition diagram with the instantiation order of the Gaia Kernel services. Solid circles denote Gaia Kernel services, while dotted circles denote middleware specific services (CORBA in our current Gaia version).

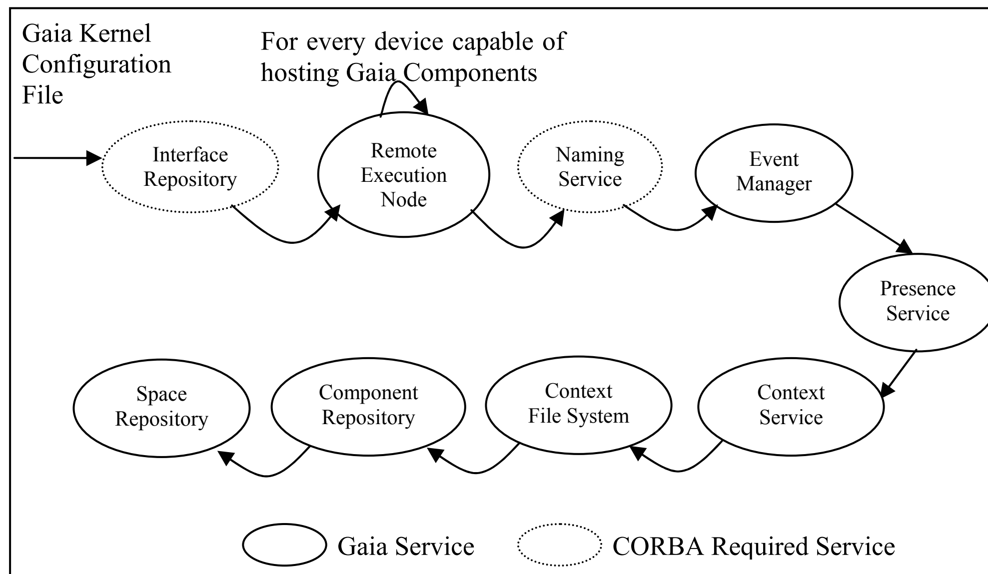

Figure 5. Gaia Kernel Bootstrap sequence.

The configuration file contains a list of primary and backup Gaia Nodes for each Gaia Kernel service; the bootstrap process uses the dependencies diagram and the configuration file to decide whether or not to start a service in a particular Gaia Node. Gaia uses a timeout mechanism and a probing protocol to ensure that each Gaia Service is started in at most one machine (as specified in the configuration file).

Sidebar 2 – Gaia OS: Extending Traditional Operating Systems to Physical Spaces

The motivation behind Gaia OS is to abstract a space and all the resources it contains as a single programmable entity. However, this motivation is not new; it is the same motivation behind traditional operating systems. According to Silberschatz et al, "*the purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner*" [30]. Gaia OS provides such environment at the space level (i.e. room, car, and home).

Silberschatz et al. define a group of seven services that are common for every operating system, namely: (1) program execution, (2) I/O operations, (3) File-system manipulation, (4) Communications, (5) Error detection, (6) Resource allocation, (7) Accounting and Protection.

Gaia OS provides functionality that covers the first six services defined by Silberschatz, and we are currently finishing a security prototype that provides functionality for accounting and protection. We provide next a comparison between the six traditional services and their Gaia OS counterpart.

Program Execution

The Gaia OS Component Management Core (CMC) provides functionality to create, destroy, and upload components in any execution node present in the a ctive space. The CMC uses the program execution facilities of the execution node's OS, which includes memory, thread, and process management.

I/O Communications

Gaia OS leverages the low-level OS I/O functionality and provides device drivers (implemented as distributed objects) that export this functionality to the rest of the active space. Gaia OS also defines default

المنارة للاستشارات
www.manaraa.com

I/O channels (i.e. input, output, and error), which are mapped to event channels. This allows creating a default "console" for the space.

File-System Manipulation

CFS provides functionality to manipulate files in active spaces. CFS interacts with the devices' low-level operating systems' file-systems to access and export the data to the active space. The specific location of the files is hidden from users and files can be accessed from any device in the active space. CFS extends traditional operating systems with functionality to transform data to different formats dynamically and to use context to organize the data based on different properties.

Communications

Gaia OS supports both direct and indirect communication mechanisms. Direct communication is similar to synchronous low-level OS IPC mechanisms, while indirect communication is the counterpart to asynchronous low-level OS IPC. Gaia OS provides RPC support for direct communication, and events (i.e. suppliers and consumers) for indirect communication. In both cases, Gaia OS leverages standard communication middleware. Events are similar to Unix signals and we use them in Gaia to notify entities in the active space about new resources added and removed, error conditions, changes in the file-system, and application state changes. The use of asynchronous events improves the reliability of the system by decoupling event producers from event listeners. While Gaia events are used mostly for signaling purposes, Gaia entities use other mechanisms such as RPC –for remote method invocations– and non-blocking streaming –for audio and video transmission.

Error Detection

Error detection includes both software and hardware errors that affect the execution of applications. Gaia OS uses the event service to report errors. Users register services that receive the error notifications and react accordingly (e.g. notify users, restart components, and suspend applications).

Resource Allocation

In traditional operating systems, resource allocation is related to the functionality required to manage hardware resources including memory, CPU, and disk. Gaia OS leverages this functionality and extends the notion of resource allocation to resources (i.e. devices, services, and applications) present in the active space. The Gaia SR stores information about resources present in the space, their owner, status (i.e. free, used, available, and malfunctioning), and properties specific to each resource. Applications use the SR to obtain the resources they require to execute. Gaia OS implements the presence service that provides functionality that is conceptually similar to "plug and play" mechanisms offered by most modern operating systems. The heterogeneity and large number of resources contained in an active space require the presence services to maintain soft-state about existing resources for reliability reasons.

5. Gaia Utilization Example: The Presentation Manager

We present in this section an application based on Gaia (Presentation Manager [31]) that we use regularly to present slideshows in our prototype active meeting room (Figure 1). The application exports functionality to present slides in multiple displays simultaneously, supports moving and duplicating slides to different displays during the presentation, and allows moving and duplicating the input sensor that controls the presentation to different devices. The presentation manager is based on the Gaia application framework and uses Microsoft's Power Point to manipulate the slides (using the COM interface).

According to our experience using the application, most of the users edit the presentation in their offices and use CFS to automatically import the data in the active meeting room. The most usual interaction mechanism is a wireless enabled handheld device running a software input sensor consisting of start, stop, next, and previous push buttons. The rest of the section consists of three subsections that summarize how the slideshow presenter enters and registers with the space, starts the presentation, and interacts with the application. We will focus primarily on the interaction between the application and the Gaia services.

المنارة للاستشارات
www.manaraa.com

Speaker Registering with the Active Space

The speaker enters the room carrying a handheld device and an RF active badge. The presence service detects the badge and sends an event to the "person/presence" channel. This event contains information about the user, including a reference to his or her profile. The space repository receives the event, retrieves an XML description for the user entity and stores the information. The context file system also receives the event about the new user, accesses the user profile, obtains the user's mount points, and mounts the data in the space. The slideshow file stored in the speaker's active office is now accessible from the active meeting room.

Next, the user registers the handheld with the space, so he or she can use it to control the presentation. The room is equipped with infrared beacons that broadcast the name of the space and a handle to a directory service. This directory service (a CORBA naming context) contains references to the Gaia kernel services. The handheld device picks up the infrared beacon, resolves the event manager from the directory, and initiates the beaconing mechanism that periodically sends a heartbeat event to the "device/heartbeat" channel. The presence service receives the event and sends a new event to the "device/presence" channel to notify the rest of the space about the new device. The space repository receives the event, contacts the device to retrieve the XML description, and stores the information. Both the user and the handheld device are now entities of the active space; they are stored in the space repository, can be contacted by other entities, and can use the resources present in the space.

Starting the Application

The active meeting room runs an application that triggers specific actions according to user specified conditions. We configure this application so it automatically starts the presentation manager application when the speaker is present in the room. This application registers with the context service to be notified when the room context meets the previously mentioned condition. The entry in the trigger service is a Lua script that gets the name of the presentation file from the "/type:/presentation/current:" context directory and starts the presentation manager. The context associated to the slideshow presentation file includes the following entry: "location=2401". Therefore, when the user data is mounted in the Active Meeting Room 2401, the file is visible from /type:/presentation/current:. The Lua script stored in the trigger service requires a valid ACD to start the application. The script uses the file system and accesses "/type:/lua/acd:/gpm/current:". This directory contains presentation manager ACDs specifically customized for 2401. The Lua script chooses an ACD named "default".

The ACD is also a Lua script that interacts with the component management core to instantiate the components. Next, the ACD contacts the coordinator of the application and registers the model, the presentations, the controllers, and the input sensors. The model interacts with the event manager to create a channel that it uses to send the update messages to the presentations and registers the presentations with the channel. The coordinator assigns the model's reference to the presentations, and the controller's reference to the input sensors. All the application components initiate the beaconing mechanism and therefore are detected by the presence service, are introduced to the space using an "entered" event, and are registered in the space repository.

Interacting with the Application

The default presentation manager ACD creates the application input sensor in one of the room's touch screens. However, the speaker decides to move the input sensor to his or her handheld device. In order to move application components we provide a library that interacts with the space repository to locate the handheld and create a new instance of the input sensor using the component management core. Next, it locates the coordinator of the application in the space repository, registers the new input sensor and unregisters the original one. Finally, the library uses the component management core to delete the original input sensor. The presence service stops receiving heartbeats from the original input sensor and therefore sends an event to the "service/presence" channel to notify that the entity left. The space repository automatically removes the information about the entity. Using the functionality provided by the library, the

المنارة للاستشارات
www.manaraa.com

speaker can move the presentations (i.e. the slides) to new displays in the middle of a presentation. Examples of new displays are the PDAs of people entering the meeting room.

The presentation manager uses four plasma displays simultaneously. When the user selects start on the input sensor running on the handheld, the input sensor sends a request to the application controller, which sends a "startPresentation" request to the model. The model sends a "start" event to the application updates channel. As a result, all presentations contact the model to obtain a handle to the presentation file. The model leverages the context file system to open the file and returns a handle to the file. The presentations get the file and display the first slide of the presentation. When the users selects "next", the input sensor sends a request to the controller, which sends a "nextSlide" request to the model. The model sends a "next" event to the application channel, which instructs presentations to display the next slide.

6. Related Work

There are several projects that address issues similar to Gaia. In this section we present some of these projects and explain how they compare to Gaia.

The Microsoft Easy Living project [32] focuses on home and work environments and states that computing must be as natural as lighting. The main properties of their environments are self-awareness, casual access, and extensibility. The infrastructure allows interfaces to move, according to user location. The project uses computer vision to recognize gestures and users, and to detect user location. The system uses this information to customize the room accordingly. We differ in that we fundamentally change the way in which applications are built, moving away from the desktop paradigm, and allowing application partitioning on different devices.

The i-Land [33] and Roomware [34] research projects present an infrastructure that digitally augments meeting rooms. The goal is to offer an environment where it is easy to exchange ideas, digitally record the results of the meetings, search in knowledge bases, and provide tools for group collaboration focusing on multimedia data exchange. The Interactive Wokspaces [35] from Stanford presents an augmented meeting room that promotes group work. The room contains wall-sized touch screens, several projectors, arrays of microphones, speakers, laptops and PDAs. The project identifies the importance of a high level operating system to coordinate the entities contained in the room. Roomware, i-Land, and Interactive Workspaces are interested in the interaction with physical spaces (mostly meeting rooms) and collaborative work groups. Our work is similar to Roomware and Interactive Workspaces in that we believe there is a need for a supporting infrastructure. However, we focus on generic spaces (e.g. office and house) which may or may not imply collaborative work. We consider that while some active spaces define a collaborative environment (e.g. meeting room and classroom), other active spaces are mostly single-user based (e.g. office and car). Furthermore, Gaia defines the notion of mobile users that can move their applications and data across different active spaces.

Aura [36] shares several common design goals with Gaia. Aura emphasizes the notion of mobile users moving around different environments. Their definition of environment is similar to our proposed notion of Active Space. Aura uses the term *task* to identify applications associated to users capable of migrating from one environment to another. Aura defines a software infrastructure to support the execution of these tasks, which maximizes the use of available resources, and minimizes user distraction. The main difference between Gaia and Aura is that Gaia emphasizes the notion of space programmability. Gaia provides mechanisms to allow users configure their applications to benefit from the resources contained in their current space. Users can interact with multiple devices simultaneously, can reconfigure applications dynamically, can suspend and resume groups of applications, and can program the behavior of applications based on context attributes. Gaia emphasizes the interaction between users and active spaces.

7. Conclusions, Contributions, and future work.

We present in this paper Gaia, a middleware infrastructure capable of managing resources contained in physical spaces. The functionality exported by Gaia simplifies the development of portable applications that

المنارة للاستشارات
www.manaraa.com

can be dynamically partitioned and mapped to a variety of devices, can be customized based on the space context, are bound to users, and can move across different spaces.

Gaia encapsulates the heterogeneity of active spaces, and presents them as a programmable environment, instead of a collection of individual and disconnected heterogeneous devices. Gaia's application framework provides functionality to build applications that exploit the resources of active spaces. Furthermore, Gaia emphasizes the interaction between users and active spaces by providing functionality to customize applications in a variety of ways. User data and applications are abstracted into a user virtual space and can be mapped dynamically to the resources located in the current environment. Users can move across different active spaces and have their virtual space always available.

Gaia contributes to ubiquitous computing in four aspects:

1. It extends the concepts of traditional operating systems to ubiquitous computing environments.
2. It provides an application framework that supports the development of applications for ubiquitous computing environments.
3. It implements a file system that uses context to organize the data according to the user activities.
4. It abstracts users' data and applications into the *user virtual space* that can be moved across and mapped to different active spaces.

As part of our future work we plan to develop new applications to validate different aspects of Gaia. We also plan to extend the infrastructure with a security service that is currently under development, and expand our current implementation of the services that support the user virtual space abstraction. Finally, we are also studying how to federate Gaia services in order to aggregate different active spaces.

References

[1] M. Weiser, "The Computer for the Twenty-First Century," in *Scientific American*, 1991, pp. 94-101.

[2] F. Kon, A. Singhai, R. H. Campbell, D. Carvalho, R. Moore, and F. J. Ballesteros, "2K: A Reflective, Component-Based Operating System for Rapidly Changing Environments," presented at ECOOP'98 Workshop on Reflective Object-Oriented Programming and Systems, Brussels, Belgium, 1998.

[3] M. Roman and R. H. Campbell, "GAIA: Enabling Active Spaces," presented at 9th SIGOPS European Workshop, Kolding, Denmark, 2000.

[4] F. Kon, R. H. Campbell, M. D. Mickunas, K. Nahrstedt, and F. J. Ballesteros, "2K: A Distributed Operating System for Dynamic Heterogeneous Environments," presented at 9th IEEE International Symposium on High Performance Distributed Computing, Pittsburgh, 2000.

[5] A. Singhai, A. Sane, and R. H. Campbell, "Quarterware for Middleware," presented at 18th IEEE International Conference on Distributed Computing Systems (ICDCS 1998), Amsterdam, The Netherlands, 1998.

[6] G. Coulson, G. Blair, N. Davies, P. Robin, and T. Fitzpatrick, "Supporting Mobile Multimedia Applications through Adaptive Middleware," *IEEE Journal on selected areas in Communications*, vol. 17, pp. 1651-1659, 1999.

[7] F. Kon, F. Costa, R. Campbell, and G. Blair, "The Case for Reflective Middleware," *Communications of the ACM*, vol. 45, pp. 33-38, 2002.

[8] F. Kon, M. Roman, P. Liu, J. Mao, T. Yamane, L. C. Magalhaes, and R. H. Campbell, "Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB," presented at IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000), New York, 2000.

[9] D. C. Schmidt, D. L. Levine, and S. Mungee, "The Design of the TAO Real-Time Object Request Broker," *Computer Communications. Elsevier Science*, vol. 21, 1998.

[10] M. Roman, A. Singhai, D. Carvalho, C. Hess, and R. H. Campbell, "Integrating PDAs into Distributed Systems: 2K and PalmORB," presented at International Symposium on Handheld and Ubiquitous Computing (HUC'99), Karlsruhe, Germany, 1999.

المنارة للاستشارات

www.manaraa.com

[11] M. Roman, D. Mickunas, F. Kon, and R. H. Campbell, "LegORB and Ubiquitous CORBA," presented at IFIP/ACM Middleware'2000 Workshop on Reflective Middleware, IBM Palisades Executive Conference Center, NY, 2000.
[12] M. Roman, F. Kon, and R. H. Campbell, "Reflective Middleware: From Your Desktop to Your Hand," *IEEE Distributed Systems Online. Special Issue on Reflective Middleware*, 2001.
[13] G. D. Abowd, "Classroom 2000: An experiment with the instrumentation of a living educational environment," *IBM Systems Journal*, vol. 38, pp. 508-530, 1999.
[14] A. K. Dey, "CyberDesk: A Framework for Providing Self-Integrated Context-Aware Services," *Knowledge-Based Systems*, vol. 11, pp. 3-13, 1998.
[15] A. K. Dey, G. D. Abowd, and D. Salber, "A Context-Based Infrastructure for Smart Environments," presented at Workshop on Managing Interactions in Smart Environments (MANSE), 1999.
[16] M. L. Dertouzos, "The Future of Computing," in *Scientific American*, 1999.
[17] A. Fox, "Building Scalable, Composable, Adaptive Internet Services with TACC," in *PhD Thesis in EECS*. Berkeley: University of Berkeley, 1998.
[18] M. Henning and S. Vinosky, *Advanced CORBA Programming with C++*: Addison-Wesley, 1999.
[19] B. Borthakur, "Distributed and Persistend Event System For Active Spaces," in *Master Thesis in Computer Science*. Urbana-Champaign: University of Illinois at Urbana-Champaign, 2002, pp. 67.
[20] B. N. Schilit, N. Adams, and R. Want, "Context-Aware Computing Applications," presented at IEEE Workshop on Mobile Computing Systems and Applications, 1994.
[21] A. K. Dey and G. D. Abowd, "The Context Toolkit: Aiding the Development of Context-Aware Applications," presented at Workshop on Software Engineering for Wearable and Pervasive Computing, Limerick, Ireland, 2000.
[22] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O. T. Jr., "Semantic File Systems," presented at SOSP13, 1991.
[23] C. K. Hess, "A Context File System for Ubiquitous Computing Environments," University of Illinois at Urbana-Champaign, Urbana-Champaign, CS Technical Report UIUCDCS-R-2002-2285 UILU-ENG-2002-1729, July 2002 2002.
[24] B. A. Myers, "Using Hand-Held Devices and PCs Together," in *Communications of the ACM*, vol. 44, 2001, pp. 34-41.
[25] G. E. Krasner and S. T. Pope, "A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System," ParcPlace Systems, Inc., Mountain View 1988.
[26] G. W. Fitzmaurice, "Graspable User Interfaces," in *PhD Thesis in Computer Science*. Toronto: University of Toronto, 1996.
[27] M. Roman and R. H. Campbell, "A User-Centric, Resource-Aware, Context-Sensitive, Multi-Device Application Framework for Ubiquitous Computing Environments," University of Illinois at Urbana-Champaign, Urbana, CS Technical Report UIUCDCS-R-2002-2284 UILU-ENG-2002-1728, July 2002 2002.
[28] R. Cerqueira, C. Cassino, and R. Ierusalimschy, "Dynamic component gluing across different componentware systems," presented at Internationa Symposium on Distributed Objects and Applications (DOA'99), Edinburgh, 1999.
[29] R. Ierusalimschy, L. Figuereido, and W. Celes, "Lua: An Extensible extension language," presented at Software: Practice and Experience, 1996.
[30] A. Silbershatz and P. Galvin, *Operating System Concepts*, 5 ed: Addison Wesley, 1998.
[31] C. K. Hess, M. Roman, and R. H. Campbell, "Building Applications for Ubiquitous Computing Environments," presented at Pervasive 2002 - International Conference of Pervasive Computing, Zurich, Switzerland, 2002.
[32] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer, "EasyLiving: Technologies for Intelligent Environments," presented at Handheld and Ubiquitous Computing (HUC), Bristol, England, 2000.
[33] P. Tandler, "Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices," presented at Ubicomp 2001: Ubiquitous Computing, Atlanta, Georgia, 2001.
[34] N. Streitz, J. Geissler, and T. Holmer, "Roomware for Cooperative Buildings: Integrated Design of Architectural Spaces and Information Spaces," presented at Workshop on Cooperative Buildings (CoBuild'98), Darmstad, Germany, 1998.

المنارة للاستشارات
www.manaraa.com

[35] A. Fox, B. Johanson, P. Hanrahan, and T. Winograd, "Integrating Information Appliances into an Interactive Workspace," *IEEE Computer Graphics & Applications*, vol. 20, 2000.
[36] J. P. Sousa and D. Garlan, "Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments," presented at IEEE Conference on Software Architecture, Montreal, 2002.

المنارة للاستشارات
www.manaraa.com