GaiaOS: An Infrastructure for Active Spaces *

Manuel Román Christopher K. Hess Anand Ranganathan
Pradeep Madhavarapu Bhaskar Borthakur Prashant Viswanathan
Renato Cerqueira Roy H. Campbell M. Dennis Mickunas

Department of Computer Science
University of Illinois at Urbana-Champaign
1304 West Springfield Avenue, Urbana, IL 61801-2987 USA

{*mroman1, ckhess, ranganat, madhavar borthaku, prashant, rcerq, roy, mickunas*}*@cs.uiuc.edu*
http://choices.cs.uiuc.edu/gaia

Report No. UIUCDCS-R-2001-2224, UILU-ENG-2001-1731

May, 2001

*This research is supported by grants from the National Science Foundation, NSF 0086094 and NSF 99-72884 CISE.

GaiaOS: An Infrastructure for Active Spaces

Abstract

We envision a world of mobile users in an unobtrusive ubiquitous computing environment that couples a computational model, digital media, and virtual representations of the physical world. Hundreds of embedded computers support the information and computational needs of each user. Users, applications, and computing devices move. The location of users and devices drives applications and resource management. Users have anytime/anywhere access to information, the network, and computational resources. Within this world, applications that make effective use of resources to support the activities of users must be simple and efficient to construct. Changes to the physical environment alter the computational model and information space of the users. Similarly, changes to the computational model and information space may alter the physical environment. We call this environment an *Active Space*. We propose a systems software infrastructure that functions in much the same way as a traditional operating system. However, instead of managing resources within a computer, it manages the computational resources within a physical space. In this paper, we describe this systems software architecture and the results of an experimental implementation called *GaiaOS*.

Contents

1 Introduction

Ubiquitous 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 or imagined world that is automated and enhanced by the behavioral context of its users. Large numbers of inexpensive 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. We call such environments *Active Spaces*. The power of a computer infrastructure to coordinate these digital and physical entities has three contributing factors: 1) the translation of information to and from physical properties, 2) the computers and their ability to transform data, and 3) the cooperative computational environment that results from embedding these devices in a network. Given the impact of the Internet, this last factor, the computational capabilities of an Active Space, is the likely long-term benefit of the current information technology revolution. However, the benefit cannot be achieved without devising a new form of operating system that enables applications to be built and run in Active Spaces. The operating system must manage the resources of a physical space and its devices for a user and his applications. This paper addresses the fundamental issues in the design of such a system.

The power of the computer as a tool to enhance thought and action encourages its broad application to human activities, endeavors and enterprises. Future "smart" cities, buildings, offices, homes, and vehicles will contain many hundreds of thousands of devices and services that join thousands of users through a hierarchy of ubiquitous networks, ranging from personal "body area" networks to the global Internet. In this ubiquitous network, there will be enormous opportunities for information appliance software to gather, organize, and analyze data and then provide sophisticated services that enhance and improve human activities, experience, learning, and knowledge. The Active Space must scale and must support the behavioral contexts of diverse groups of mobile users. In particular, the Active Space must support application programs written by or for these users.

In direct analogy with building a computing environment for users to run applications on a dedicated machine, we propose that 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, the reuse of software, and offer portability and flexibility. An active office, home, classroom, operating theater, building, or city application and its appropriate libraries should run on a ubiquitous infrastructure. A home or office application should be movable from one physical location to another, overlaying the actual physical devices and geometry of a home or office. Users should be able to interact with their office or home whether or not they are physically present. The Active Space should permit "virtualization" of resources so that a user may access their home from their laptop, work, or a conference. Entering an Active Space should not require a user login; yet users must be authenticated and user spaces must be secure.

The end result of this effort will be uniquely user-centric and an application-oriented computational environment. To realize this vision, we have developed an operating system for physical spaces that creates an Active Space. The foundation of the system provides network communication that allows entities to communicate. On this foundation lies the kernel, which consists of essential services that are required for applications to operate in the context of a physical space. In order to build those applications, we have developed an application model that addresses the

shortcomings of models designed for current desktop environments. The remainder of the paper continues as follows. Section 2 gives an overview of our software infrastructure and Section 3 gives a detailed description of the architecture, including the communication mechanism and core kernel services. Section 4 describes our application model and Section 5 explains how we coordinate components in the system through the use of a flexible scripting language. Finally, Section 6 covers related work and Section 7 offers some concluding remarks.

2 Gaia: Enabling Active Spaces

Several approaches for interacting with ubiquitous computing environments are customized for particular scenarios or are targeted towards a specific type of application. We propose a general model, called *Gaia*, which exports and coordinates the resources contained in a physical space thereby defining a generic computational environment [Gro]. The model we present requires a revolutionary form of operating system that manages heterogeneous computational resources and exports them uniformly, facilitating the control of physical spaces, and the development of applications that run in the context of a space. *Gaia* converts physical spaces and the ubiquitous computing devices they contain into a programmable computing system or Active Space. We describe the supporting infrastructure to build applications that exploit ubiquitous communicating devices embedded within a physical space.

We argue that Active Spaces are 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. However, the heterogeneity, mobility and sheer number of devices makes the system vastly more complex. Applications may have the choice of a number of input devices, such as mouse, pen, or finger; output devices, such as monitor, PDA screen, wall-mounted display, or speakers. An operating system for such a space must be able to locate the most appropriate device, detect when new devices are spontaneously added to the system, and adapt content when data formats are not compatible with output devices. Traditional operating systems manage the tasks common to all applications; the same management is necessary for physical spaces. Our research is focused on the study of issues related to the design and implementation of such an operating system, called *GaiaOS*. Applications running on this system require the development of a novel application model that defines the rules to build applications that run in the context of a space.

To support the construction of applications for Active Spaces, we adapt the traditional Model-View-Controller (MVC) architecture to meet the requirements of ubiquitous computing. The traditional MVC model is composed of software components. We expand the notion to include hardware and physical entities, context information, and dynamic application construction. Models may be ambient room conditions, human vital signs, or a person's schedule; views may be projection screens, a phone, lights, or audio speakers; controllers may be a mouse, keyboard, hand motion, a person entering a room, or voice. Different views may be registered with the same model to render data and may be attached and detached at different times; multiple controllers may be used for applications and may choose which is most appropriate as environmental conditions change. The infrastructure must manage all these dependencies and seamlessly adapt to device types and characteristics.

3 Architecture

GaiaOS is a component based meta-operating system, or middleware operating system [2K 98], that runs on top of existing operating systems (e.g., Windows2000, WindowsCE, and Solaris). Figure 1 illustrates the five main components of *GaiaOS*. At the lowest level, the *Unified Object Bus* provides tools to homogeneously manipulate heterogeneous components running in the system. This bus is the foundation on which the remaining of *GaiaOS* components rely. The *Gaia Kernel* includes essential services that implement the core functionality of the system, including discovery, repository, events, naming, data storage and manipulation, and security. The *Gaia Services* include additional services, such as QoS [XWN00b, XWN00a], which enhance functionality to support applications. The *Gaia Application Model* defines a standard framework for creating ubiquitous applications, which run in the context of the space. Finally, the *Active Space Execution Environment* comprises the "user level" of our operating system. It consists of all services and applications that run in the context of a particular Active Space that are not part of *GaiaOS*. The following sections describe each of these components in further detail.

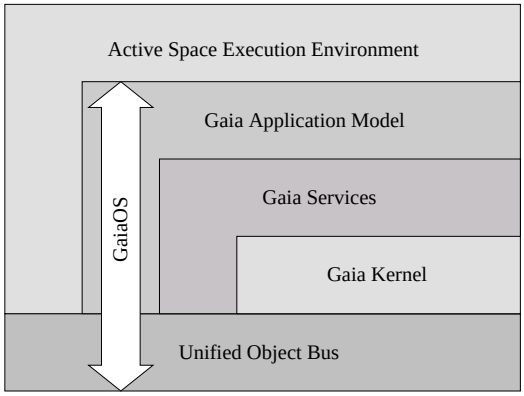

Figure 1: *Gaia* is composed of the communication substrate, the kernel, additional services, and the application model.

3.1 Unified Object Bus

Active Spaces are highly heterogeneous by definition. They include a variety of hardware devices and diverse software protocols. However, in order to export a space as a programmable entity, such heterogeneity must to be hidden. Programmers must be offered a common interface to manipulate components in the system, regardless of their specific properties and details of the hosting device. The Unified Object Bus (UOB) is responsible for providing tools to manipulate the lifecycle of components running in an Active Space, such as component instantiation, component inspection, component deletion, and component naming. The UOB can manipulate different component types (e.g., CORBA, Scripts, and Java Beans) and provides an open architecture to seamlessly incorporate new component models. Components integrated into the UOB are called *Unified Components* and programmers manipulate them using a common interface, regardless of the specific type of component. The UOB defines four basic abstractions, which include the *Unified Component*, *UOBHost*, *Component Container*, and *Component Manager*.

Unified Components are the basic elements of the UOB. These components follow a common

naming scheme, can be dynamically manipulated (i.e., created, deleted and inspected) regardless of their execution location, and can be shipped across the system. However, these components do not assume a particular implementation. Instead, they define a specific component lifecycle. Therefore, it is possible to use existing component models and add additional functionality so their components follow the lifecycle protocol defined by the UOB. As a result, these components can be manipulated as unified components. The Component Manager exports the interface to manipulate the lifecycle of components (i.e., creation, activation, naming, and destruction). It also 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. It exports functionality to manage the dependencies of the components it contains, and has by default an instance of a Component Manager, which allows manipulating the components that belong to the container. 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 creating components in particular component containers.

3.2 Gaia Kernel

The *Gaia* Kernel is the core of the *GaiaOS*. It contains the minimum required services to bootstrap *GaiaOS* in any arbitrary space.

3.2.1 Naming Service

GaiaOS is a component-based system where every entity is abstracted as a distributed object. This requires a mechanism to name objects and export their names so they can be browsed and resolved by any interested party. Components are registered in the *Naming Service*, which stores <*name, value*> tuples. Every Active Space has an associated naming space, where all kernel services and user level objects specific to the space are registered. Although every space runs its own instance of the Naming Service, it is possible to federate them and therefore export a common namespace containing object references from different Active Spaces, enabling inter-space collaborations.

3.2.2 Event Manager

The *Event Manager* is used to distribute information among components, while maintaining loose coupling. The Event Manager is based on the channel abstraction; channels simplify the management of events, by classifying them into different categories, and therefore allow the creation of event hierarchies. For example, *GaiaOS* uses the "Discovery" channel to notify the space about entities entering and leaving the space. Another channel is the "Error" channel, which is used by components to post events with information about specific error conditions. Applications can register to specific event channels to be notified of information or changes in the environment. The dynamic nature of Active Spaces makes this communications model beneficial in situations where components may not have explicit knowledge about each other. Moreover, this model can insulate applications against component failures.

However, events are not the only means by which applications can communicate. *GaiaOS* relies on a dual communication model that combines both events and peer-to-peer interaction. While events are helpful for some specific situations, such as distributing information when consumers and suppliers are not known *a priori*, other scenarios require a direct communication mechanism. For example, consider a group of components that are part of an application and explicitly know

about each other. In this situation, it is more efficient to use a peer-to-peer mechanism in such a way that components do not need to go through a third party to communicate. However, the dual model makes it possible to use hybrid solutions where components can combine both models according to the specific requirements.

3.2.3 Discovery Service

Active Spaces are very dynamic and constantly in flux. Services, applications, devices, and entities are frequently introduced and removed from a space. The dynamic nature is further increased by failures, which are inevitable in any distributed system, and it is therefore necessary to keep track of currently active entities. The *Discovery Service* is responsible for tracking software components, people, and physical entities present in a space. Information about currently active components is required for diagnostics and "self-healing" activities that stabilize the system by handling component failures. Information about people and physical entities present in an Active Space is used as context information and affects the behavior of active applications and services.

The Discovery Service is composed of two sub-services: the *Presence Service* and *Informer Service*, as shown in Figure 2. The Presence Service keeps track of software entities currently active in a space, while the Informer Service is responsible for user and physical entity detection. Both services employ a leasing mechanism to determine whether or not a particular entity is still active. Software components are responsible for sending heartbeats at a pre-established rate to renew their leases. The Presence Service utilizes the Event Manager to receive information regarding the presence of entities. This information is distilled by the Informer Service to decide when an entity enters and leaves a space and can notify components of this occurrence.

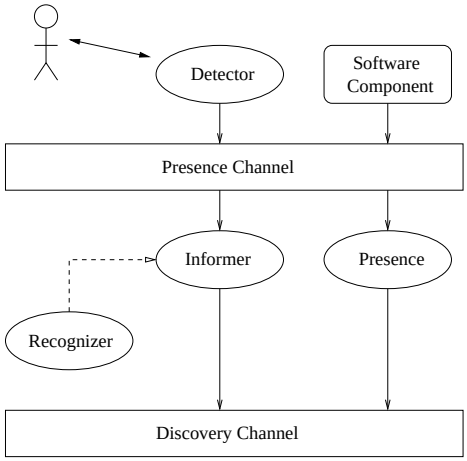

Figure 2: Channels are used to discover entity presence and inform applications of the status of entities.

Users and physical entities are not direct members of the digital world, which implies they cannot negotiate leases on their own behalf. These physical entities are sensed by specialized devices, which provide the required information to decide whether or not they are still located in a particular space. The Presence Service interprets the data obtained from different sensors and decides whether or not the physical entity is still present in the space. While the physical entity is still present, the Presence Service automatically renews its lease on behalf of the entity by sending

heartbeats. Due to the diversity of sensing technologies, the Presence Service must be able to incorporate new sensing technologies. To support future inclusion of new technologies, service is constructed as a framework that allows the incorporation of new sensor interpreters. In our current implementation, we have components to interpret data from active badges. We are working on the creation of components that incorporate data received from iButtons [iHP] and cameras. Note that the ability to seamlessly incorporate new sensing technologies makes the system more robust and accurate. It is possible to combine the information from different sensors to locate a single physical entity.

3.2.4 Space Repository

The *Space Repository* is responsible for storing information about entities (i.e., devices, services, applications, and users) currently active in the context of the space. The Space Repository exports an interface that allows searching for specific entities, based on constraints using a specific query language.

Each entity has an associated description, specified in XML, which is registered in the Space Repository. Entity descriptions include fields such as the type of component, a generic description, a name, the type of data that the entity can manipulate, and the reference of the object. The Space Repository listens to the events generated by the Discovery Service in order to keep the information it stores up-to-date. On learning about the presence of a new entity, the Space Repository contacts that entity, requests its associated XML description, and stores all relevant information. In the same way, when an entity becomes inactive, the Space Repository receives an event and automatically removes the component. This mechanism allows applications to discover what types of devices are currently available in a space.

3.2.5 Security Service

Security concerns in *Gaia* include authentication, access control, secure dynamic loading of components, secure tracking, and location privacy. In addition to general access control and authentication issues, a user may desire their location and activity be kept private.

Apart from a traditional login/password mechanism, Active Spaces can authenticate and recognize users in different ways. These involve voice recognition, retinal scan, fingerprint matching, active badges, and swipe cards. *Gaia* 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.

Credentials are used for authentication and delegation of authority to trusted and untrusted services. We employ three types of credentials; 1) *generic credentials* for delegation of authority to trusted programs, 2) *restricted credentials* for delegation of authority to untrusted programs and 3) *non-delegable credentials* for simple authentication. The credential stores information about a user or space identity, users or space roles and attributes, delegation restrictions if present, and a timestamp. Credentials are signed by the Authentication Service using a key which it shares with the Access Control Service.

Gaia provides a mechanism to enforce different kinds of access control [Gil99]. It provides administrative role-based access control and discretionary access control. Role-based access control is applied to all resources in the system. Discretionary access control is provided to users so that they can secure their private resources.

3.2.6 Data Object Service

Traditional distributed file systems are generally designed for homogeneous environments and simply transfer contiguous bytes of data to the local node. However, the heterogeneous and dynamic nature of Active Space environments deems the static configurations of traditional distributed file systems inappropriate. *Gaia* applications access data through the *Data Object Service* (DOS), a dynamically typed file system that supports content adaptation, customized data access, and location awareness.

Active Space applications cannot make assumptions regarding the devices and preferences of users [BBG+00]. Therefore, the system must be able to adapt to these dynamic conditions. Adaptation and data grouping is achieved through the use of *containers*. Containers encapsulate data type-specific knowledge and parse data sources into indexed components. Type-aware containers may be linked together to create chains, which may perform some sequence of data transformation. Therefore, an application simply specifies the type it wishes to access the source as and the system transparently converts the data source to the desired type. In addition, indexing facilitates customized data access and random access, by using the type mechanism to signify alternate data groupings. For example, a device may wish to access pages, rather than text characters. Containers provide higher-level semantics by presenting data in a form that is more meaningful to applications.

A further consideration is the physical location of a user. A user may define personal storage that can be incorporated into a space when they are physically present. In essence, a user's personal file system may "follow" them as they move between spaces. Personal storage may reside on remote desktop machines or mobile special-purpose devices. Storage is automatically made available to applications running in a space, without the need to manually transfer files. Implicitly linking storage to a user simplifies data management and the way in which applications access data. We envision a variety of small mobile devices (e.g., mp3 players, cameras, etc.) will export data in a generic way to allow their storage to be available to other devices by placing storage servers on them.

4 Application Model

The *Gaia Application Model* provides a standard mechanism 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 in the traditional Model-View-Controller (MVC) paradigm, augmented with extensions to account for the dynamic nature of ubiquitous computing environments.

4.1 Traditional MVC

MVC [KP88] defines a modular design that models the behavior of interactive applications by clearly encapsulating 1) the model of the application domain (model), 2) the visualization of the model (view), and 3) 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 an input sensor (e.g., mouse, keyboard and pen), which captures input events generated by users. The result of the input event (e.g., left mouse key pressed) depends on the associated view and the actions derived from the control mechanism are automatically sent to the views associated with the model.

4.2 Model-Presentation-Adapter-Controller-Coordinator

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. The Model-Presentation-Adapter-Controller-Coordinator (MPACC) is an application model that maps the MVC pattern into Active Space environments. This new model takes into account issues such as the variety of interaction device, 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. The MPACC defines a model that standardizes the way in which applications for Active Space environments are built. From the user perspective, the ubiquitous application model defines a model of using and customizing applications. These applications run in the physical space inhabited by users (e.g., car and office) and therefore users interact 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 the model for 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. Therefore, the user cannot only see the model of an application, but also possibly hear it, smell it, and touch it. One of the goals of this application model is to define a framework that standardizes the way in which applications for ubiquitous computing environments are designed, built, and assembled.

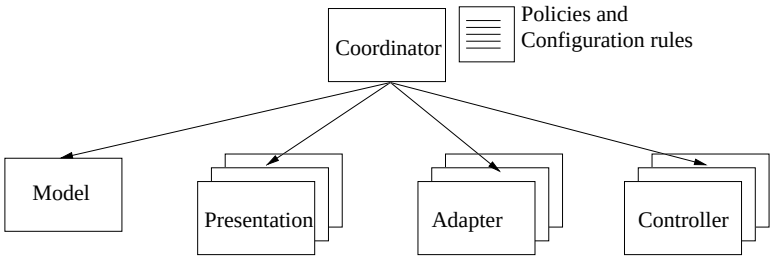

Figure 3: The application model consists of the Model, Presentation, Adaptor, Controller, and Coordinator.

The model for ubiquitous applications defines five elements: 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 programmatic 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 not only coordinates input devices, but it includes 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 meta-level application manager that handles all non-application domain specific issues, such as application structure, and adaptation policies. The Coordinator stores references to the components that compose the application, as well as policies regarding adaptation, customization and mobility of the application. Figure 3 illustrates a schematic diagram of the application model.

5 Active Space Coordination

Gaia abstracts the entities contained in the space as distributed objects. *GaiaOS* uses a high level scripting tool, called LuaOrb [CCI99], to coordinate these entities and to easily program and configure the space. LuaOrb is based on the interpreted language Lua [IFC96, IFC99], which provides a set of abstractions that can be used to connect the available components in order to compose new services and applications.

Lua facilitates automating 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 ideal for resource-constrained devices. LuaOrb implements language bindings between Lua and CORBA, COM, and Java. These language bindings were designed to support dynamic component composition; that is, they allow identification of new component types and the integration of their instances into a dynamically assembled application. LuaOrb's ability to directly communicate with various component models allows it to easily interact with the components in our system.

We use Lua as the glue to dynamically assemble components in *Gaia*. This allows us to easily create configuration scripts, customize the behavior of spaces, and prototype applications. Furthermore, Lua is extensively used to capture events from the Event Manager (e.g., discovery, errors, and context information) and trigger specific actions when certain conditions are met. The richness of the Lua syntax allows us to dynamically create arbitrary complex triggers. We can easily create context translators that listen for specific events and translate, aggregate, or distill them to generate new events. Scripts can define how to process particular events in order to package a collection of events to give them semantic meaning for applications [SDA99, DAS99]. In addition, we take advantage of the interpretative character of Lua to send scripts around the space for reconfiguration purposes.

6 Related Work

In recent years, ubiquitous computing has become a serious research topic. Some of the seminal work occurred at Xerox Parc, which introduced a Tab [SAG+95], a low-capacity networked terminal, and Active Badges [WHFG92], a mechanism for tracking users.

The Microsoft Easy Living project [Mic] 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 detect user location, and recognize gestures and users, and 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 allow applications to run on arbitrary devices.

Hewlett Packard has developed CoolTown [Hew], which proposes adding web presence to physical spaces. The Web paradigm is extended to physical spaces, which contain infrared beacons that provide URLs related to the physical space. The web browser is used as the standard application GUI and one of the key aspects is the dynamic creation of customized web pages according to contextual parameters. The project associates data with physical locations, which helps to present information that is useful to the current location of a user. In contrast, our project is attempting to create a generic computing environment.

MIT's Oxygen project [Der99] envisions a world where embedded computers will be so pervasive that it becomes a part of everyday life. Speech, vision, and movement drive interaction with these machines. Oxygen defines three key components: N21, E21 and H21. N21s are networks that interconnect devices, enabling collaborative regions. These networks support different types of protocols and provide naming and security mechanisms. E21s are devices embedded in physical spaces, responsible for turning such spaces into intelligent environments. E21s provide large amounts of computation as well as access to different types of sensors and displays. Finally, H21s are the mobile points of interaction, capable of reconfiguring themselves. H21s communicate over E21 networks though do not explicitly require an E21. However, if E21s are present, the H21 can offload computation and communication to them. While our projects have several similarities, Oxygen focuses on human-computer interactions, whereas *Gaia* is investigating the application model and data abstractions.

The Endeavour project [Kat99] from Berkeley focuses on integrating sensors, actuators, position locators, displays, and mobile handheld devices. A key component is the processing and management of information as it flows through the system. Portolano [EHAB99] from the University of Washington strives to provide invisible computing through transparent user interfaces, universal connectivity, and intelligent services. The project is investigating determining user interfaces based on user location and movement, and data-centric networking, and novel models for distributed services.

The i-Land [SGH+99] and Roomware [SGH98] 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 Workspaces [FJHW00] from Stanford presents an augmented meeting room that promotes group work. The room contains wall-sized touch screen, 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. The main software infrastructure is an event heap, which allows different devices to interoperate without the strictness of a peer-to-peer model. This infrastructure is robust and easy to maintain. Both of these projects are interested in the interaction with physical spaces and collaborative work groups. Our work is similar to the Interactive Workspaces, in that we believe there is a need for a supporting infrastructure or operating system. However, we focus on the construction of applications, in contrast to their work, which focuses on making existing applications operate in workspaces to support collaborative groups.

The Ninja project from UC Berkeley [GWBC99, GWvB+00] defines an architecture consisting of four main components: bases, units, active proxies, and paths. Bases are manifested as a cluster of workstations that provide scalability, fault tolerance, and concurrency. Units comprise the myriad of devices that may be connected to the infrastructure. The active proxies provide adaptation

of content (similar to our containers), and are the result of previous research in data distillation using the TACC [Fox] model to perform on-the-fly data transformations [FGBA96, FGG+98]. The last component, paths, constructs flows of data that may be transformed while passing through different components, using their active proxies [CMI]. The Ninja architecture is investigating scalable and robust ways to build scalable Internet services. We are focusing on physical spaces and the services required to support applications in such spaces.

The Iceberg project uses the Ninja framework to provide an infrastructure for integrated network communications [RKJ99]. The infrastructure supports call establishment and signaling. One of the main focuses is support for user device modality. Users can switch between devices and the system can adapt to the properties of the device. Iceberg is an infrastructure for messaging, in contrast to *Gaia*, which is developed as a general computational environment.

7 Conclusions

In this paper, we present our vision of ubiquitous computing as the technology that drives the interaction between human activities and their associated physical spaces. We believe ubiquitous computing requires the creation of a generic digital infrastructure to assist users in their everyday tasks. The result of combining a physical space with the digital infrastructure is what we call an Active Space. It is a programmable entity, populated by a dynamic group of mobile users, consisting of physical devices and digital services, and coordinated by a standard software infrastructure. In addition to being programmable, they also can react to physical stimulus to affect applications.

We have describe *GaiaOS*, our software infrastructure implementation to enable Active Spaces. This infrastructure resembles a traditional operating system in that it manages heterogeneous computational resources and exports them uniformly, facilitating the control of physical spaces, and the development of applications that run in the context of a space

We have finished a prototype of *GaiaOS* and are concentrating on building applications that take benefit of the exported infrastructure. Initial results show that the design of traditional desktop applications must change to accommodate the dynamic nature of Active Spaces and the wide variety of computing devices. We are studying how to build and use applications that take full benefit of the surroundings, by leveraging the available services, context, and computing devices that exist in such a space.

We have constructed an application builder that allows us to create distributed applications by dynamically connecting available components. We have built some demonstrations of ubiquitous applications that take advantage of the *GaiaOS* infrastructure, such as an MP3 player, a slide presentation application, and a high-bandwidth HDTV video controller that operates on multiple displays. The *Gaia* application model allows us to partition applications, dynamically place application components, adapt to different device characteristics, react to contextual changes, and attach/detach application components. Although we still have many issues to investigate related to application development in Active Space environments, our preliminary results show us that offering a common infrastructure can significantly reduce the complexity of this task.

8 Acknowledgments

The authors would like to thank Fabio Kon for discussions during the early design stages of our system. We would also like to thank Cristina Ururahy for providing Lua scripts for our system.

9 Resources

Resources and further documentation for *Gaia* is available at *http://choices.cs.uiuc.edu/gaia*.

References

[2K 98] 2K Research Team. 2K: A Component-Based Network-Centric Operating System for the Next Millennium. `http://choices.cs.uiuc.edu/2k`, 1998.

[BBG+00] Guruduth Banavar, James Beck, Eugene Gluzberg, Jonathan Munson, Jeremy Sussman, and Deborra Zukowski. Challenges: An Application Model for Pervasive Computing. In *Proceedings of the Sixth ACM/IEEE Int. Conf. on Mobile Networking and Computing*, pages pp. 266–274, 2000.

[CCI99] Renato Cerqueira, Carlos Cassino, and Roberto Ierusalimschy. Dynamic component gluing across different componentware systems. In *International Symposium on Distributed Objects and Applications (DOA'99)*, pages 362–371, Edinburgh, 1999. IEEE Press.

[CMI] Sirish Chandrasekaran, Samuel Madden, and Mihut Ionescu. Ninja Paths: An Architecture for Composing Services Over Wide Area Networks. `http://ninja.cs.berkeley.edu/dist/papers/paths.ps.gz`.

[DAS99] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A Context-based Infrastructure for Smart Environments. In *Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE '99)*, pages pp. 114–128, 1999.

[Der99] Michael L. Dertouzos. The Future of Computing. *Scientific American*, 1999.

[EHAB99] M. Esler, J. Hightower, T. Anderson, and G. Borriello. Next Century Challenges: Data-Centric Networking for Invisible Computing: The Portolano Project at the University of Washington. In *Proceedings of the Mobicom'99*, 1999.

[FGBA96] Armando Fox, Steven D. Gribble, Eric A. Brewer, and Elan Amir. Adapting to Network and Client Variation via On-Demand Dynamic Distillation. In *ASPLOS-VII*, Boston, MA, October 1996.

[FGG+98] Armando Fox, Ian Goldberg, Steven D. Gribble, David C. Lee, Anthony Polito, and Eric A. Brewer. Experience With Top Gun Wingman: A Proxy-Based Graphical Web Browser for the 3Com PalmPilot. In *IFIP International Conference on Distributed Systems Platformas and Open Distributed Processing*, Lake District, UK, September 1998.

[FJHW00] Armando Fox, Brad Johanson, Pat Hanrahan, and Terry Winograd. Integrating Information Appliances into an Interactive Workspace. *IEEE Computer Graphics and Applications*, 20(3), May/June 2000.

[Fox] Armando Fox. The Case for TACC: Scalable Servers for Transformation, Aggregation, Caching, and Customization. Qualifying Exam Proposal.

[Gil99] Binny Sher Gill. Dynamic Policy-Driven Role-Based Access Control for Active Spaces. Master's thesis, University of Illinois at Urbana-Champaign, 1999.

[Gro] Software Research Group. Gaia: Enabling Active Spaces. `http://choices.cs.uiuc.edu/gaia`.

[GWBC99] Steven D. Gribble, Matt Welsh, Eric A. Brewer, and David Culler. The MultiSpace: an Evolutionary Platform for Infrastructural Services. In *Proceedings of the 1999 Usenix Annual Technical Conference*, Monterey, CA, June 1999.

[GWvB+00] Steven D. Gribble, Matt Welsh, Rob von Behren, Eric A. Brewer, David Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Josheph, R. H. Katz, Z. M. Mao, S. Ross, and B. Zhao. The Ninja Architecture for Robust Internet-Scale Systems and Services. *Special Issue of Computer Networks on Pervasive Computing*, 2000.

[Hew] Hewlett Packard Company. Cooltown. `http://www.cooltown.hp.com`.

[IFC96] Roberto Ierusalimschy, Luiz Figueiredo, and Waldemar Celes. Lua—an extensible extension language. *Software: Practice & Experience*, 26(6):635–652, 1996.

[IFC99] Roberto Ierusalimschy, Luiz Figueiredo, and Waldemar Celes. *Reference Manual of the Programming Language Lua version 4.0.* Rio de Janeiro, Brazil, 1999. (available at `http://www.lua.org/ftp/refman.ps.gz`).

[iHP] iButton Home Page. iButton: Armored steel computer chip for everyday wear and tear. `http://www.ibutton.com`.

[Kat99] Randy H. Katz. The Endeavour Expedition: Charting the Fluid Information Utility. `http://endeavour.cs.berkeley.edu/proposal`, 1999.

[KP88] Glenn E. Krasner and Stephen T. Pope. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System, 1988. Parc Place Systems Inc, Mountain View.

[Mic] Microsoft Corp. Easyliving. `http://www.research.microsoft.com/easyliving`.

[RKJ99] Bhaskaran Raman, Randy H. Katz, and Anthony D. Joseph. Personal Mobility in the ICEBERG Integrated Communications Network. Technical Report UCB/CSD-99-1048, UCB EECS, May 1999.

[SAG+95] Bill N. Schilit, Norman I. Adams, Rich Gold, Karin Petersen, David Goldberg, John R. Ellis, and Mark Weiser. The PARCTAB Ubiquitous Computing Experiment Roy Want. Technical report, Xerox Palo Alto Research Center, March 1995. Technical Report CSL-95-1.

[SDA99] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proceeding of CHI'99*, Pittsburgh, PA, May 15-20 1999. ACM Press.

[SGH98] Norbert Streitz, Jörg Geissler, and Torsten Holmer. Roomware for Cooperative Buildings: Integrated Design of Architectural Spaces and Information Spaces. In *Proceedings of the First International Workshop on Cooperative Buildings (CoBuild'98)*, pages pp. 4–21, Darmstadt, Germany, February 25-26 1998.

[SGH+99] Norbert Streitz, Jörg Geissler, Torsten Holmer, Shin'ichi Konomi, Christian Müller-Tomfelde, Wolfgang Reischl, Petra Rexroth, Peter Seitz, and Ralf Steinmetz. i-Land: An interactive Landscape for Creativity and Innovation. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'99)*, pages pp. 120–127, Pittsburg, Pennsylvania, May 15-20 1999.

[WHFG92] Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1):91–102, 1992.

[XWN00a] Dongyan Xu, Duangdao Wichadakal, and Klara Narhstedt. Multimedia Service Configuration and Reservation in Heterogeneous Environments. In *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS2000)*, April 10-13 2000.

[XWN00b] Dongyan Xu, Duangdao Wichadakal, and Klara Narhstedt. Resource-Aware Configuration of Ubiquitous Multimedia Service. In *Proceedings of IEEE International Conference on Multimedia and Expo 2000 (ICME2000)*, New York, New York, July 31-August 2 2000.