Gaia: Enabling Active Spaces

Manuel Roman, Roy H. Campbell
{mroman1,rhc}@cs.uiuc.edu
Department of Computer Science.
University of Illinois at Urbana-Champaign
1304 West Springfield Avenue
Urbana, IL, 61801

Abstract

Ubiquitous computing promotes physical spaces with hundreds of specialized embedded devices that increase our productivity, alleviate some specific everyday tasks and provide new ways of interacting with the computational environment. Because the computational environment is spread across the physical space, personal computers lose the focus of attention. Therefore, the users' view of the computational environment is finally extended beyond the physical limits of the computer. Physical spaces become computer systems, or in other terms, Active Spaces. However, these Active Spaces require novel system software capable of seamlessly coordinating their hidden complexity. Our goal is to extend the model provided by current computer systems to allow interaction with physical spaces and their contained entities (physical and virtual) by means of a single abstraction called Active Space.

1. Introduction

Ubiquitous computing promotes the proliferation of embedded devices specialized in specific tasks. Devices will be everywhere: providing new functionality, enhancing user productivity and simplifying everyday tasks. One of the main aspects of ubiquitous computing is invisibility, which means that users perceive the functionality but not the devices that provide this functionality.

Physical spaces have an associated geographical location, define some physical boundaries and have some semantic meaning associated (an office and a kitchen are both rooms but the tasks performed in each of them are different). All these properties confer a particular personality to every physical space, which in turn dictates the type of entities expected in the space (e.g. physical devices, computational services and people), the way in which those entities interact and finally the services that the physical space hosts. These physical spaces, and not the particular entities they contain, are perceived as computing systems; however this perception is purely abstract since it is not possible to interact with particular physical spaces as a single entity at any level (programmatically or interactively). Only individual devices and services contained in a physical space can be addressed, therefore losing the perception of a single computing environment, which implies the impossibility of accessing the context (both physical and virtual) associated to the physical space.

There are several approaches that provide models for interacting with active spaces [1] [2] [3]. However, the solutions they provide are customized for particular scenarios or are targeted towards a specific type of functionality.

In this paper we introduce the Gaia OS, a system software infrastructure designed to provide support for Active Spaces. An active space is a model that maps the abstract perception of a physical space as a computing system, into a first class software entity. An active space encapsulates contextual information related to the physical space, both physical and virtual, and allows interaction with the physical space, both interactive and programmatically. An active space blends virtual and physical properties therefore presenting the physical space as a "living entity" with well defined behavior and explicitly defined functionality. Physical properties associated to the space (e.g. location of entities and sound and light conditions of the space) affect the execution of applications, in the same way that the execution of applications affects the physical properties of its associated physical space.

An active space provides a generic model that hides the complexity associated to its internal computational status and exports a standard programming interface independent of the underlying physical space. Applications can then be developed in the context of generic active spaces, because the Gaia OS adapts the application requirements to the particular properties of its associated physical space. Therefore, applications do not have to “reinvent the wheel” to deal with the particular characteristics of every possible physical space where they can be executed. The Gaia OS abstracts the particular properties of arbitrary physical spaces in the same way that a traditional OS abstracts the particular hardware of every machine, therefore exporting a generic computational environment.

2. Computer Systems and Active Spaces

In the introduction we have defined an active space as a generic computing system. This means that traditional computing systems such as personal computers are particular cases of active spaces. In this section we map the standard components defined for a traditional computing system to their active space counterparts. We assume a personal computer as an example of a standard computing system and an active office as an example of an active space. We also assume that the active office contains some personal computers and workstations, video cameras, an overhead projector, devices to detect the location of users in the office and devices to configure the lights and temperature of the room.

Traditional computing systems can be divided into four components: hardware, operating system, application programs and users [4]. The **hardware** component of a personal computer comprises one or several processing units with their associated volatile memory, permanent storage and several peripherals. The active office exports, at least, as many processing units as personal computers and workstations are contained in the office. The active office coordinates all the existing processing units exporting them uniformly by providing, for example, a multicomputer model composed of weakly coupled processors. The active office also contains persistent storage, which equals the sum of available persistent storage exported by each computing system contained in the active office (fixed and mobile devices). Finally, and still dealing with hardware, come the peripherals. In the case of the personal computer, there are standard protocols to interact with those peripherals. For those attached to internal cards, some of the standards are: PCI, AGP and ISA. They define the electrical signals required to enable interaction between the personal computer and the cards by means of the internal bus. The active office also contains, coordinates and exports peripherals (e.g. temperature and location sensors and light and temperature controllers). The active office defines a software bus to enable data distribution among peripherals. This is equivalent to the internal bus contained in a personal computer. The software bus provides a well-defined protocol to enable communication among entities contained in the active office. Examples of protocols for the software bus are IIOP, DCOM and Java RMI. So far we have already introduced the hardware mapping from a computing system (a personal computer) to an active space (an active office). Thereby we have a physical space modeled as a computing system. However, it is not an active space yet. It is simply a physical space with computational capabilities that can host software to control the resources exported by the physical space. At this point the active office can be compared to a distributed object system and most of the existing solutions stop at this point.

Initially, computing systems did not have an **operating system**. Therefore, one of the many problems was that each application had to implement the code to control the hardware devices it required [4]. Modeling a physical space as a computing system without an operating system poses exactly the same problems. We have a collection of uncoordinated entities that each application has to configure appropriately. However, this is not a major problem. Assume now that we have more than one application running in the same space, or more than one user managing the resources exported by the space. This is obviously a common situation that will arise in almost every active space. The problem of the previous situation is how to decide which entity can use what devices. The solution is to provide a generic software infrastructure to manage the resources contained in a physical space. In other words, the physical space requires an operating system. This operating system converts the physical space into an active space, or in other words, into a “living entity” aware of itself, with reactive behavior and capable of interacting and coordinating the entities it contains. An important aspect of this operating system is that it combines physical and virtual properties of the active space.

The last two components of a computing system, **Application Programs** and **Users**, fit naturally in the model presented. However, the traditional way of understanding applications and users changes once the model is extended. In the context of active space, a user, from the point of view of an entity that “uses” the operating system is not restricted to a person. It can be any entity in the active space. Consider for example the case of an active car that checks the traffic status to update the route presented by the navigational system. The car is a user of a particular active city and there is no human user involved. Application programs benefit from the new computational model we define because for their computational purposes they can use any of the entities exported by the active space, included users. According to our new model users are exported as resources of the active space. Applications use the standard interface of the active space model to request specific functionality therefore becoming independent of the particular properties of the underlying physical space.

An active space is thus a physical space with its associated entities (physical and virtual) plus an operating system that standardizes the mechanisms required to coordinate the computational capabilities of the active space. The infrastructure we present, the Gaia OS, enables active spaces and models them as generic computing systems. Therefore it is possible to reuse most of the existing principles associated to standard computing systems.

3. Gaia

In the 1970s James Lovelock formulated and popularized the idea of the Earth - the collective atmosphere, oceans, lithosphere, and biosphere - as a single, cybernetically self-functioning superorganism. This global entity he called "Gaia", after the old Greek Earth-Goddess.

The Gaia project conceives physical spaces, their semantic and physical properties and their contained entities as active spaces. As such, active spaces are aware of all contained resources (physical and computational) and their status, export a contextual model and provide a two-way interaction mechanism. The Gaia project defines an OS (Gaia OS), an application model (Gaia Application Model) and the active space in itself; these components are depicted in figure 1. Gaia does not intend to define high-level policies or rules regarding the behavior of the entities running in the space. It simply provides the tools and a minimum set of low-level policies that allow modeling arbitrary active spaces.

The Gaia OS is in itself a component based distributed metaoperating system that runs on existing operating systems and is implemented on top of existing middleware platforms. It includes the Unified Object Bus and the Gaia OS services (check section 3.1 for more details).

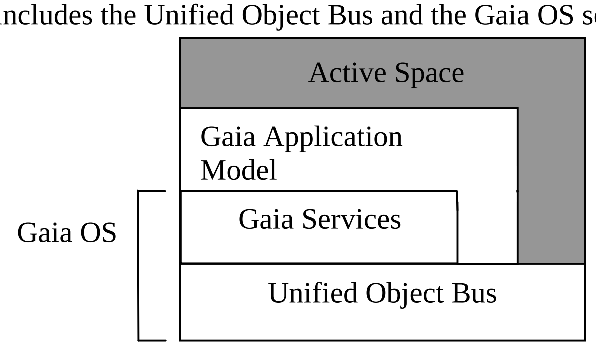

Figure 1. Gaia Architecture

The Gaia Application Model provides support for building, registering and managing applications that run in the context of the active space (section 3.2).

The active space models a particular physical space and the entities it contains. It uses the Gaia OS to perform most of his tasks. The state of the active space consists on physical and semantic properties of the physical space as well as entities contained in the active space. The active space exports functionality to manipulate the active space in different ways; section 3.3 provides information about the exported functionality.

3.1 The Gaia OS

The Gaia OS provides the infrastructure on top of which active spaces are implemented. The two components of the Gaia OS are: the Unified Object Bus and the Gaia Services.

The Unified Object Bus defines the distributed object model for the OS. It consists on: (1) a core (the Component Management Core) which provides low level functionality to implement components, but without enforcing any particular policy or specific distributed object model; (2) a set of components that define how to integrate different component models (Integrators); (3) a collection of ORBs (Object Request Brokers) that implement the remote communication mechanisms for each component model that has been integrated and (4) the execution environments for the components (Component Containers). The unified object bus allows dynamic instantiation and reconfiguration of entities, management of inter-entity dependencies [5] and integration of different component models. In our case, we provide integrators and ORBs for CORBA, because all the Gaia services are implemented as CORBA services. We are currently using two CORBA ORBs' implementations: dynamicTAO [6] for PCs and workstations and LegORB [7] for handheld and palm-size computers. From the point of view of handheld, palm-size and in general any embedded device, the Gaia OS architecture expands their functional capabilities. By implementing the component management core they can be integrated in the active space and therefore have access to any of the exported services. If required, Gaia can dynamically adapt some of the services to the capabilities of those devices. Current version of the unified object bus runs on Solaris, Windows NT and Windows CE.

The Gaia Services are implemented as Unified Components (CORBA objects in our case), run in the context of the active space and can be accessed by any entity with functionality to interact with the unified object bus. Gaia Services are based on the 2k operating system [8]. 2k is a component-based, reflective, adaptable and user-centric metaoperating system. It is based on the premise "What You Need Is What You Get" (WYNIWYG) meaning that the system dynamically reconfigures itself to provide the functionality required by each particular entity.

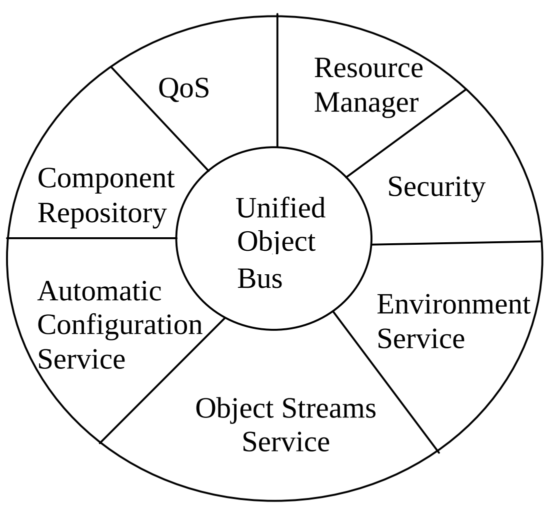

Figure 2. Gaia Services

As depicted in figure 2, there are seven Gaia Services, which run on top of the unified object bus: QoS [9], ResourceManager [10], Security, Environment Service [11], Object Stream Service, Automatic Configuration Service[12] and the Component Repository.

3.2 Gaia Application Model

Many of the applications in Active Spaces are interactive. These applications involve dependencies between sensing, computation, and physical space, views or displays. In Gaia OS, we provide an application management service that supports the ability to create, register, modify, list, delete, and schedule interactive applications and deliver their events. The application management service is inspired by the Model-View-Controller [13]. In a MVC, the model contains application data; the view is a visual representation of the model, the controller provides information to update the view and the model, and the system maintains consistency between the view and the model as they change. In Gaia, this framework is reinvented to support adaptive interactive applications in the distributed, dynamic, and heterogeneous nature of Active Spaces. The framework introduces a new element called the *model adapter* which is an instance of an Object Stream that transforms, at runtime, the type of data exported by the model to the type of data expected by a view. Creating an interactive application in the context of an Active Space requires providing a model, attaching one or more views, and associating one or more controllers to each of the views. Views are attached to the model using model adapters that format the data according to the requirements of the view. Finally, controllers, reacting to sensing services, modify views and their corresponding models. In an Active Space, the view can broadly be interpreted to include not only the displays in the physical space but also the effects of any embedded actuator devices in the physical space. For example, activating a microphone, switching on a set of lights and projecting and image on a screen is one of the views associated to a videoconferencing model that uses the position of the user inside the active space as the controller. The Gaia Application model provides functionality to create and register the different components of applications (model, view or controller), manipulate the application, schedule the application according to several configurable parameters (physical and virtual) and resolve conflicts between different applications.

3.3 Active Space

It implements the active space model according to the particular properties of its associated physical space. The implementation is responsible for exporting the abstraction of the physical space and all its associated properties by using the functionality exported by the Gaia OS.

In general we use the term *entity* to denote any physical or computational object contained in the space. The active space provides mechanisms to manage the entities, but it does not impose any policy. All policies used by the active space are customizable.

The interface exported by the active space allows interaction with the space as a single element and not as a collection of heterogeneous entities. Therefore, the active space encapsulates the complexity associated to the space by providing functionality to: (1) store and retrieve information related to the active space; (2) to manipulate the active space; (3) discover and check the status of the entities; (4) partition the space into regions with well defined properties; (5) locate entities; (6) manipulate events and (7) bootstrap the active space.

Providing an abstraction for active spaces allows different levels granularity. The active space can be approached as an entity in itself that has a state and exports well defined functionality. From this perspective it is possible to develop applications that exploit inter-active space relations by interacting with instances that abstract particular physical spaces. On the other hand, it is possible to reason about the active space as a container of entities and policies, therefore emphasizing on the individuality of the contained entities.

4. Related Work

There are several projects related to digitally enhancing physical spaces. They provide solutions for particular scenarios and specific types of applications.

The Microsoft Easy Living project [1] 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.

HP is working on CoolTown [2], 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.

Finally Georgia Tech's Classroom 2000 [3] preserves and augments electronic notes taken by students and teachers by attaching and synchronizing audio, video and annotations to class notes.

5. Conclusion

Gaia offers a generic infrastructure to manage arbitrary physical spaces with large collections of entities. The result is what we call an active space that allows developing applications without a strict knowledge of the underlying infrastructure provided by the physical space. Gaia is responsible for mapping those applications to particular physical spaces. We have extended existing concepts for traditional computing systems to model active spaces. We believe this strategy facilitates the understanding of the concept.

The Gaia OS is similar in some aspects to standard distributed object systems; however, what it makes it unique is the tight integration with the physical environment and the explicit knowledge of each of the entities contained in an active space. The tight integration implies that changes in either the physical or virtual environment affect the other counterpart. The knowledge of each of the entities allows the active space to adapt the global execution environment to each of the particular entities.

When considering a traditional computing system and an active space, it is not correct to consider the active space as a particular case of the traditional computing system. It has to be understood the other way around. That is, a traditional computing system is a particular case of an active space. It can be defined as a micro-active space.

Gaia extends the computational environment beyond the PC, but not only to rooms or buildings but also to campuses, cities and in general any arbitrary physical space. Active spaces will not exist in isolation but instead will interoperate among them to provide a distributed computational environment.

References

1. Shafer, S., et al., *The new EasyLiving Project at Microsoft Research*. 1999, Microsoft Research.
2. Caswell, D., *Creating a Web Representation for Places*. 1999, Internet Systems and Applications Laboratory.
3. Abowd, G.D., *Classroom2000: An experiment with the instrumentation of a living educational environment.* IBM Systems Journal. **38**(4 - Pervasive Computing).
4. Silbershatz and Galvin, *Operating System Concepts.* 1994: Addison Wesley.
5. Kon, F. and R.H. Campbell, *Dependence Management in Component-Based Distributed Systems.* IEEE Concurrency, 2000. **8**(1): p. 26-36.
6. Roman, M., F. Kon, and R.H. Campbell, *Design and Implementation of Runtime Reflection in Communication Middleware: the dynamicTAO Case*, in *Proc. ICDCS'99 Workshop on Middleware*. 1999: Austin, TX.
7. Roman, M., et al. *LegORB and Ubiquitous CORBA*. in *IFIP/ACM Middleware'2000 Workshop on Reflective Middleware*. 2000. Palisades, NY.
8. Kon, F., et al., *2K: A Reflective, Component-Based Operating System for Rapidly Changing Environments*, in *ECOOP'98 Workshop on Reflective Object-Oriented Programming and Systems*. 1998: Brussels, Belgium.
9. Nahrstedt, K., D. Wichadakul, and D. Xu. *Distributed QoS Compilation and Runtime Instantiation*. in *Proceedings of IEEE Internation Conference on Multimedia and Expo (ICME2000)*. 2000. New York.
10. Yamane, T., *The Design and Implementation of the 2K Resource Management Service*, in *Master Thesis*. 2000, University of Illinois at Urbana-Champaign.
11. Carvalho, D., et al. *Management of Environments in 2K.* in *7th International Conference on Parallel and Distributed Systems (ICPADS-2000)*. 2000. Iwate, Japan: IEEE.
12. Kon, F. and R.H. Campbell, *Supporting Automatic Configuration of Component-Based Distributed Systems*, in *Proc. 5th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'99)*. 1999: San Diego, CA. p. 175-187.
13. Krasner, G.E. and S.T. Pope, *A Cookbook for using the Model-View-Controller User Interface Paradigm in Smalltalk-80.* Object-Oriented Programming, 1988: p. 27-49.