Major Research Activities of the Project

1. Introduction

Several existing approaches for interacting with ubiquitous computing environments are customized for particular scenarios or are targeted towards a specific type of application. We have proposed a general model, called Gaia, which exports and coordinates the resources contained in a physical space thereby defining a generic computational environment. Gaia converts physical spaces and the ubiquitous computing devices they contain into a programmable computing system or Active Space.

Active Spaces encompass a wide range of research topics. We have focused our research in the follow areas:

- Infrastructure
- Multimedia
- Networking
- QoS
- Graphics
- Recognition

In the following sections, we describe the major research activities that we have performed in these areas during the past year.

2. Infrastructure

The previous year has involved the identification and design of the necessary components and initial prototypes to support an infrastructure for Active Spaces, which we call GaiaOS. The middleware infrastructure is comprised of the *Unified Object Bus*, which provides tools to homogeneously manipulate heterogeneous components running in the system. This bus is the foundation on which the remaining parts of GaiaOS 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, 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 *Gaia 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.

1.1. 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. We have developed the Unified Object Bus (UOB), which 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 and can be dynamically manipulated (i.e., created, deleted and inspected) regardless of their execution location. 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.

1.2. Gaia Kernel

The Gaia Kernel is the core of GaiaOS. It contains the minimum required services to bootstrap GaiaOS in any arbitrary space. Following is a discussion of the components that were determined to be necessary for a wide range of applications.

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 discovered and used by other applications. 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. We use an implementation of the CORBA Naming Service for this component.

Event Manager

The Event Manager is used to distribute information among software 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 with 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. The Event Manager is based on the CORBA event service, with additions to facilitate the creation and management of event channels.

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. In order to track software components, people, and physical entities present in a space, we have developed a component called the Discovery Service. 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. 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.

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, the 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 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.

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.

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 could involve diverse mechanisms including voice recognition, retinal scan, fingerprint matching, active badges, iButtons, camera recognition 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. The Authentication Service using a key, which it shares with the Access Control Service, to sign credentials.

Gaia provides a mechanism to enforce different kinds of access control. 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.

Data Object Service

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. 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 a sequence of data transformations. Applications simply specify the type of a source 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 the user is 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.

The current implementation allows applications to pull data objects. Future work will include pushing data and investigate caching and fault tolerance.

1.3. Application Model

We are currently developing the Gaia Application Model, which provides a standard mechanism to build applications for ubiquitous computing scenarios. 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.

The concepts defined by the traditional MVC appear 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 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 "software pattern" into Active Space environments. This new model takes into account issues such as the variety of interaction device, contextual properties associated with 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 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 only rendering it graphically. The model can be externalized in any possible way that affects the senses of a user. Therefore, the user can not 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.

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.

1.4. Active Space Coordination

Gaia abstracts the entities contained in the space as distributed objects. GaiaOS uses a high-level scripting tool, called LuaOrb, to coordinate these entities and to easily program and configure the space. LuaOrb is based on the interpreted language Lua, 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 assemble dynamically 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. In addition, we take advantage of the interpretative character of Lua to send scripts around the space for reconfiguration purposes.

3. Multimedia

Active Spaces may incorporate multimedia content including audio and video. Spaces may interact with users through audio or users may view video feeds on a variety of display devices. Our on-going work includes high definition video transmissions, multimedia on small devices, and location sensitive video.

We have developed an HDTV streaming server using equipment provided to the department through an NSF CISE grant. The streaming software is integrated with the DCS HDTV capture and processing hardware that includes:

- HDCam Camera
- HDCam Tape Deck
- HDTV Non-linear hard-disk editing system
- HDTV Broadcast MPEG encoder
- HDTV DVB/ASI capture stations

The streaming system captures the DVB/ASI stream from the broadcast encoder and transmits the MPEG transport stream packets over IP/UDP or IP multicast. Experiments to determine latency and loss for a variety of transport stream coalescing strategies have been performed. An architecture for management and control of standalone server components deployed within the network was devised and is being implemented. More recently, we have established a research relationship with EchoStar networks that permits our Active Space to include IP streamed live satellite HD feeds.

We developed an HDTV streaming client application that displays the IP based HDTV streams that are generated by the streaming server. The streaming client manages an HDFocus hardware card to drive the display with data received from the IP network. This client has been used to drive a number of displays including a 64" Philips, HD projector, and Flat Panel displays simultaneously within a lab setting. In addition, we have developed an architecture for deploying large numbers of HD streaming servers and streaming clients under network control.

The HDTV transmission feeds have been integrated with the Active Space infrastructure to allow the remote control of IP streamed video feeds into example offices or conference rooms. For example, we have built an application that allows multiple video streams to be selected and controlled from a handheld wireless device. In one smart room demonstration, two decoding processors and HDTVs allow the coordinated simultaneous display of coordinated HDTV IP video streams as part of a presentation. The Gaia infrastructure manages the HDTV software components, switching streams, and control functions. Once the components are loaded and registered by the UOB, the handheld is able to discover the objects and interact with them through the Gaia middleware. We have developed a small communication library that is able to interact with our infrastructure and is small enough to run on a variety of small handheld devices. Supported functions are video stop and start, HDTV selection, and video feed selection.

4. Networking

Users in an Active Space are mobile and may be carrying wireless handheld devices that interact with a space. Different spaces may have varying degrees of network connectivity and reliability. We are currently building a system that will enable seamless use of all available wireless communication media based on a location-based database of previous and expected wireless connectivity. As the user moves, the database can be queried as to what wireless connectivity is expected in that area. The database is updated with current connectivity information (i.e., signal strength, channel characteristics) from mobile devices.

Indoor locations are determined by a diffuse infrared beacon, while outdoor locations are determined by GPS. The mobile device monitors its location and can then use locally cached information or query the database for expected connectivity. This design allows the mobile device to determine where it is without divulging such information to others, providing anonymity, while still allowing the mobile to determine its location.

The next step of this research is to use such location-based connectivity information to determine how to configure a mobile's connections. Connectivity decisions will be based on cost, quality and energy consumption.

5. Quality of Service

An Active Space consists of different types of devices with different available resources interacting together to form the intelligent interactions via the running distributed applications corresponding to different user requirements.

Hence, the quality of service in a smart space can be provided via the appropriate management of distributed applications and devices' resource availability. Corresponding to this, we have developed a component-based QoS management framework, called $2K^Q$, as a core service for QoS provision in the Active Space. $2K^Q$ is based on the integrated approach of QoS compilation and reconfiguration, component-based runtime middleware. Since QoS provisions should start from the development phase, we have

developed an innovative approach of QoS compilation and translations and a visual QoS-aware application programming environment, which allows the application developer to develop a QoS-aware application systematically and easily. In the execution phase, we have developed innovative algorithms for distributed service component configuration and contention-aware multi-resource reservation. In addition to the $2K^Q$ framework, we have designed a scalable middleware called SMART for ubiquitous multimedia delivery in Active Space environments. To experiment with this middleware, we have constructed example applications, such as voice mail, video preview, a universal remote control, and media session handoffs on handheld devices. We also introduce the security authentication as part of media session handoffs. In addition to the $2K^Q$ and SMART, we are working on the resource coordination among distributed displays in an Active Space based on the distributed dynamic soft real-time CPU scheduler (DSRT). Further, we are studying the addition of adaptation specifications including adaptations corresponding to the dynamic nature of Active Spaces, such as user mobility and interest and resource availability in the $2K^Q$, based on QoS compilation.

6. Graphics

One of our primary goals is to be able to efficiently transport and display complex geometric content within the Active Space environment. Our specific focus over the past year has been in creating high-quality surface simplification methods, which can analyze raw polygonal surface models, producing a hierarchy of surface regions that can be used to facilitate view-dependent refinement, progressive transmission, and compression.

In order to achieve these goals, we have developed the permission grid, a spatial occupancy grid used to guide almost any standard polygonal surface simplification algorithm into generating an approximation with a guaranteed geometric error bound. In particular, the distance between any point on the approximation and the original surface is bounded by a user-specified tolerance. Such bounds are notably absent from most current simplification methods, and are becoming increasingly important for applications such as collision detection and scientific computing. Conceptually simple, the permission grid defines a volume in which the approximation must lie, and does not permit the underlying simplification algorithm to generate approximations outside of this volume.

The permission grid makes three important, practical improvements over current error-bounded simplification methods. First, it works on arbitrary triangular models, handling all manners of mesh degeneracy's gracefully. Further, the error tolerance may be expanded as simplification proceeds, allowing the construction of an error-bounded level-of-detail hierarchy with vertex correspondences among all levels of detail. And finally, the permission grid has a representation complexity independent of the size of the input model, and a small running time overhead, making it more practical and efficient than current methods with similar guarantees.

7. Recognition

Active Spaces are populated by human users and physical objects. These physical entities must be integrated into the digital realm that directly controls the applications and devices in a space. Users many move within a space, which may trigger particular events, such as switching application focus or passing control of an application to a different user. Users may also interact with applications through the use of gestures. For example, a finger may be used as input device, replacing a mouse if it is not available or not convenient. Physical objects can be integrated into the geometric model of a space to determine the relationship between users and objects. Each of these situations are handled through the use of video analysis, through hand gesture recognition, people tracking, and image-based modeling and rendering of objects with complex reflectance functions.

To use of hand gestures for human-computer interaction, we have developed in collaboration with others at UIUC a real-time approach to the spotting, representation and recognition of hand gestures from a video stream. The approach exploits multiple cues including skin color, hand motion and shape. Skin color analysis and coarse image motion detection are joined to perform reliable hand gesture spotting. At a higher level, a compact spatio-temporal representation is proposed for modeling appearance changes in image sequences containing hand gestures. The representation is extracted by combining robust parameterized image motion regression and shape features of a segmented hand. For efficient recognition of gestures made at varying rates, a linear re-sampling technique was developed to eliminate temporal variation while maintaining the essential information of the original gesture representations. The gesture is then classified according to a training set of gestures.

We have begun implementing a person tracking system using multiple binocular and monocular video streams. To cover the area of the active (smart) room, four IEEE-1394 (firewire) binocular video cameras will be interfaced to four PC's, each of which will be running dense binocular stereopsis code to segment the image into candidate regions based on disparity and color. Using motion information and long term tracking of these regions across all four video streams, we expect robust tracking of all occupants of the space. In the longer term, this information will be augmented by multiple monocular color digital sources.

We have also looked at the problem of how to reconstruct the geometry of objects with an arbitrary and possibly anisotropic bidirectional reflectance distribution function (BRDF). Present reconstruction techniques, whether stereo vision, structure from motion, laser range finding, etc. make explicit or implicit assumptions about the BRDF. We introduced two methods that were developed by re-examining the underlying image formation process; they make no assumptions about the object's shape, the presence or absence of shadowing, or the nature of the BRDF which may vary over the surface. The first method takes advantage of Helmholtz reciprocity while the second method exploits the fact that the radiance along a ray of light is constant. In particular, the first method uses stereo pairs of images in which point light sources are co-located at the centers of

projection of the stereo cameras. The second method is based on double covering a scene's incident light field; the depths of surface points are estimated using a large collection of images in which the viewpoint remains fixed, and point light source illuminate the object. We have demonstrated the use of these techniques for both reconstruction and also for image-based rendering and compositing.

8. Conclusions

Through the work performed on the infrastructure during the last year, we have been able to determine key services that are required for applications running in an Active Space. We have implemented prototypes of the services described above. Initial appl$_{ication}$ prototypes have validated our need for generic services. To facilitate the construction of appl$_{ications}$, we have begun development of an appl$_{ication}$ model that defines a framework that specifies the way in which application components are assembled and communicate. Future work will involve the construction of several applications that will help us refine the current implementations, add need functionality to services and possibly the construction of new services, experiment with the integration of resident devices and physical entities into software appl$_{ications}$, and create a stable execution platform.

We are exploring the use of IA64/Linux cluster computers for use in parallel rendering of HDTV images. We are finalizing a cooperative research relationship with Avid/SoftImage. We will experiment with various network and distributed system algorithms for partitioning and reassembling the workload, load balancing, media processing, etc. This work will leverage the IA64/Linux Renaissance project at UI funded by HP.

Incorporating graphic appl$_{ications}$ will involve making the appropriate extensions to the existing middleware components to support these applications. Specifically, the event delivery service must be extended to support the low latency requirements of continuous event streams from devices such as a mouse. We also need to put in place the rendering architecture required to drive the several display devices in the Active Space. After evaluating the available technologies, we expect to adapt the WireGL system developed at the Stanford Graphics Lab.

Our work in video analysis will allow us to integrate physical entities into the digital infrastructure. One of our goals is to make computers easier to use. Gesture recognition is a step in achieving this goal by allowing users to interact with the computers without electronic input devices.

Our first year has laid the groundwork for future research. We are currently equipping a room that will be used as a test bed for our Active Space environment. In our second year, we will start to integrate the separately developed parts and develop appl$_{ications}$ to validate our approach. Through further experience with our environment, we will be able to further enhance the support needed for appl$_{ications}$ running in the context of a physical space.