Building Applications for Ubiquitous Computing Environments [*]

Christopher K. Hess, Manuel Román, and Roy H. Campbell

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Abstract. Ubiquitous computing embodies a fundamental change from traditional desktop computing. The computational environment is augmented with heterogeneous devices, choice of input and output devices, mobile users, and contextual information. The design of systems and applications needs to accommodate this new operating environment. In this paper, we present our vision of future computing environments we term *User Virtual Spaces*, the challenges facing developers, and how they motivate the need for new application design. We present our approach for developing applications that are portable across ubiquitous computing environments and describe how we use contextual information to store and organize application data and user preferences. We present an application we have implemented that illustrates the advantages of our techniques in this new computing environment.

1 Introduction

Ubiquitous computing challenges the conventional notion of a user logged into a personal computing device, whether it is a desktop, a laptop, or a digital assistant. When the physical environment of a user contains hundreds of networked computer devices each of which may be used to support one or more user applications, the notion of personal computing becomes inadequate. Further, when a group of users share such a physical environment, new forms of sharing, cooperation and collaboration are possible and mobile users may constantly change the computers with which they interact. We believe this requires a new abstraction for computing which we term a *User Virtual Space*. The User Virtual Space is composed of data, tasks and devices associated to users, it is permanently active and independent of specific devices, moves with the users, and proactively maps data and tasks into the users' ubiquitous computing environment according to current context.

User Virtual Spaces pose several challenges regarding data availability, application mobility and adaptability, context management, and resource coordination. As a result, applications must provide functionality to be partitioned

[*] This research is supported by a grant from the National Science Foundation, NSF 0086094, NSF 98-70736, and NSF 99-72884 CISE.

among different devices, move from device to device, and adapt as demanded by users and their environment. We believe that this environment requires support from an external software middleware infrastructure that provides a standard set of mechanisms and interfaces to coordinate the contained resources. We use the term *active space* to refer to any ubiquitous computing environment orchestrated by such a middleware infrastructure.

In this paper, we discuss our proposed solution for constructing user-centric, space-aware, multi-device applications, which meet the requirements imposed by User Virtual Spaces and describe an example application that we have implemented in our prototype active space. The remainder of the paper continues as follows: Section 2 discusses the motivation for our application framework. Section 3 details the framework and Section 4 discusses how data is managed in our environment. Section 5 gives an overview of our development platform and Section 6 describes an application developed with our framework, which includes how the application is created, instantiated, and used. Sections 7 and 8 discuss related work and conclusions, respectively.

2 Motivation

Aspects unique to User Virtual Spaces motivate the need for a different approach to application construction. Below we discuss some of the inherent difficulties in the environment that affect applications. In later sections, we present our model for building applications that accommodates these challenges.

Context. A distinguishing feature of ubiquitous computing is context-driven application adaptation. Context can trigger changes in applications to adapt to the surroundings of a user in order to facilitate the use of the computational environment [10, 2]. User Virtual Spaces maintain a collection of data for each user that may be remotely distributed and constitutes user's application data, configurations, and preferences. For any given task, only a portion of this personal data may be required; context can be used to organize data such that the information pertinent to the current context is easily accessible. Limiting the amount of data is even more important when a space is populated by a number of users that have come together to perform a specific task, where each user may contribute some amount of information to the shared space, and the amount of aggregate information can become large. By pruning irrelevant material not necessary for the current task, locating information may be simplified. In addition, continually running applications may not have the luxury of human intervention to manually search for data. If relevant information, which may change over time, is known to exist in a particular location, the application is relieved from performing costly searches over the entire collection of data.

Binding. Ubiquitous environments are often task-driven, which often precludes applications from being permanently associated to a single device, or to a collection of well-defined devices. Applications must be constructed so they can be partitioned and mapped into the most appropriate resources according to physical context, personal configurations, and other attributes, thereby transferring the application binding from particular devices to users and spaces. Consider a music jukebox application, in which the application automatically finds and uses the most appropriate resources to continue playing and controlling the music as context changes (e.g., as the user moves from office to car) The emphasis is on the application itself, instead of the devices used to execute and control the application.

Mobility. Mobility affects the application architecture by requiring the ability to migrate parts of their functional components at runtime. User mobility also has implications on data availability; users should not be burdened with manually transferring files or data, be it configurations, preferences, or application data. Personal data should be available to them regardless of their physical location. Data becomes implicitly linked to a user and can follow them around, becoming available whenever they enter a new active space.

Adaptability. Application developers should not be concerned with the complexities of data format conversions; they should gain access to data in a particular format by simply opening the data source as the specific desired type and the system should be responsible for automatically adapting content to the desired format.

User preferences or influences from the environment may also affect the internal structure of an application. Applications must expose a means to configure this structure in order modify the way in which a user interacts with an application or the way in which the application data is presented. For example, control of an application running in an active space may be passed between group members who have different devices, new controllers may be attached to enable parallel manipulation of the application, or a user may switch to a new type of controller, such as from pen-based to voice-activated.

3 Application Framework

Ubiquitous environments require a user-centric multi-device application model where applications are partitioned across a group of coordinated devices, receive input events from different devices, and present their state using different types of devices. Applications must be designed such that their composition may be changed based on the context of the current situation. Therefore, the structure of an application must be described by a generic set of components that can be customized based on the available resources or preferences. This gives users more flexibility in deciding how to interact with applications by allowing them to choose among a number of input, output, and processing devices.

To address the challenges inherent in developing applications for ubiquitous computing environments, we have developed an application framework that assists in the creation of generic loosely coupled distributed component-based applications. The framework reuses the application partitioning concepts described by the traditional Model-View-Controller and introduces new functionality to export and manipulate the bindings of the application components. Exposing information about component bindings allows reasoning about the composition of the application components, and enables modification of this composition according to different properties, such as context and user preferences.

Our application framework defines three basic components that constitute the building blocks for all applications: the Model, Presentation (a generalization of the View), and Controller, where the Model implements the application logic, the Presentation exports the model's state, the Controller maps input events (e.g., input sensors and context changes) into messages sent to the model. To coordinate these components, the application framework introduces a fourth element, called the Coordinator, which is responsible for storing the application component bindings, as well as exporting mechanisms to access and alter these bindings, such as (dis)connecting a controller or presentation, and listing current presentations. The Model, Presentation, and Controller are strictly related to application functionality, while the Coordinator implements the meta-level functionality of the application. The application framework (MPCC) defines methods for performing common tasks, such as application mobility.

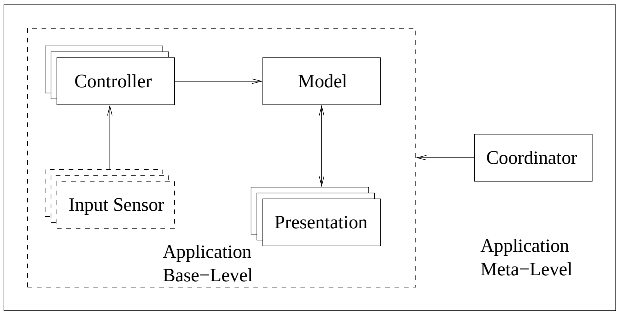


Fig. 1. The Model-Presentation-Controller-Coordinator application framework decouples application components and exposes the internal structure of the application

Applications are constructed independent of a particular active space by using generic application descriptions that can be customized for the resources available in a specific space. For example, a calendar application running in an active office may use a plasma display to present the appointments for the week, a handheld to display the appointments for the day, and may use a controller running in the desktop PC to enter data. However, the same calendar running in a vehicle may use the sound system to broadcast information about the next appointment, and use a controller based on speech recognition to query the calendar or to enter and delete data. The framework defines two types of application description files: the application generic description (AGD), and the application customized description (ACD). The AGD is independent of the space in which the application runs and contains information regarding the required components that compose the application, such as the name and type of components and the number of instances allowed. The AGD acts as a template from which concrete application configurations can be generated. The ACD is an application description that customizes an AGD to the resources of a specific active space. The ACD consists of information about what components to use, how many instances to create, and where to instantiate the components. The application framework offers tools that allow users to create and store ACDs. The resulting ACD is a script that coordinates the application instantiation process.

4 Application Data

Generated ACDs are typically associated with the context for which they are created. Therefore, we store application configurations in the context of the active space, particular to the owner of the configurations. We have developed a data management system that is context aware and plays the role of a file system in a traditional environment. It allows application data to be organized based on context to limit the scope of a user's full collection of data to what is important for the current task and provides dynamic data types to accommodate device characteristics and user preferences. The system uses context to alleviate many of the tasks that are traditionally performed manually or require additional programming effort. More specifically, context is used to 1) automatically make personal storage available to applications, conditioned on user presence, 2) organize data to simplify the location of data important for applications and users, and 3) retrieve data in a format appropriate to user preferences or device characteristics.

The file system is composed of distributed file servers and mount servers. Each file server manages data on the machine on which it is executing. An active space maintains a single mount server, which stores the current storage layout (i.e., namespace) of the space. The mount server contains mappings to data relevant to the active space, as well as the personal data of users. The user mappings are dynamic, changing as users move between spaces; when a user enters an active space, their personal data is dynamically added, making it locally available.

There are occasions when data is generated in a different context from which it will be used. In such cases, the context in which the data is important must be explicitly attached to the data to make it available in the specified context. Our system uses standard file system primitives (i.e., *rename*, *remove*, *copy*, *mkdir*) to attach and detach context to files and directories.

The system constructs a virtual directory hierarchy based on what types of context are available and what context values are associated with particular files or directories. What data items are available depends on the current context of the active space, which may change over time as users move, situations change,

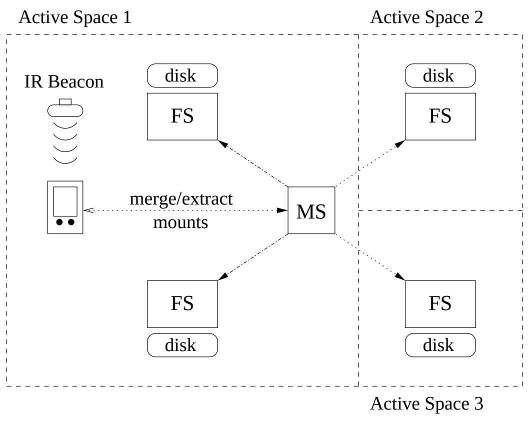


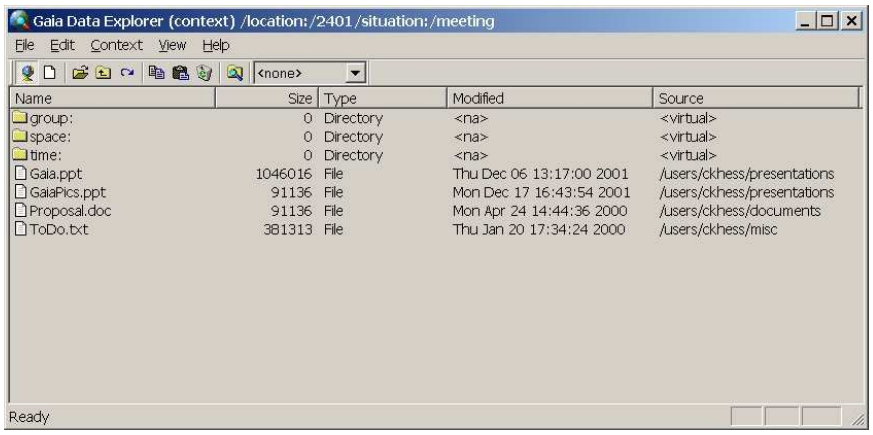
Fig. 2. The file system architecture consists of distributed file servers (FS) and mount servers (MS). A mobile handheld may be used to carry the mount points of a user, which can be merged into an active space to make remote personal storage available to the space

or new tasks are initiated. The layout of the directory hierarchy is implemented using the mounting mechanism, where mount points are owned by users and contain context indicators. Mount points may be automatically retrieved from a home server or can be carried with a user via a mobile handheld device and injected into the current environment to make personal storage available to applications and other users, as shown in Figure 2. In our current implementation, we employ the latter approach. A special directory called "current" aggregates data that is important to the current context. This supports the concept of the User Virtual Space by automatically configuring an active space to the needs of the user and the tasks being performed. Even though a user's data may be dispersed among several remote machines, that data is presented as a single source with only pertinent information visible.

We have implemented a shell and graphical directory browser, shown in Figure 3, that can be used to view the virtual directory structure, can be used to associate context with files and directories, and can launch applications.

5 Infrastructure

We have developed a middleware infrastructure, called *Gaia* [9], that exports and coordinates the resources contained in a physical space and provides a generic computational environment. *Gaia* converts physical spaces and the ubiquitous computing devices they contain into a programmable computing system. *Gaia* is similar to traditional operating systems by managing the tasks common to all applications built for physical spaces. Each space is self contained, but may interact with other spaces. *Gaia* provides core services, including events, entity


Gaia Data Explorer (context) /location:/2401/situation:/meeting
File Edit Context View Help
<none>
Ready

Name	Size	Type	Modified	Source
group:	0	Directory	<na>	<virtual>
space:	0	Directory	<na>	<virtual>
time:	0	Directory	<na>	<virtual>
Gaia.ppt	1046016	File	Thu Dec 06 13:17:00 2001	/users/ckhess/presentations
GaiaPics.ppt	91136	File	Mon Dec 17 16:43:54 2001	/users/ckhess/presentations
Proposal.doc	91136	File	Mon Apr 24 14:44:36 2000	/users/ckhess/documents
ToDo.txt	381313	File	Thu Jan 20 17:34:24 2000	/users/ckhess/misc

Fig. 3. Screenshot of the context file system browser. The virtual hierarchy is used to attach context to files and directories

presence (devices, users, and services), context notification, discovery, and naming. By specifying well-defined interfaces to services, applications may be built in a generic way that are able to run in arbitrary active spaces. The core services are started through a bootstrap protocol that starts the *Gaia* infrastructure.

6 Presentation Manager

In this section, we present the Gaia Presentation Manager (GPM), an application based on the application framework that provides functionality for creating and presenting synchronized slide shows. These slide shows exploit multiple input and output devices contained in a ubiquitous computing environment, can present multiple (possibly different) slides simultaneously in multiple form factors, can attach and detach controllers, and can migrate slides views to different displays.

The functionality to edit synchronized slide shows allows users to define the number of steps for the presentation, select multiple slides from existing Power Point files, and specify synchronization rules. These synchronization rules define which slides to present in every step, and what display to use for every slide. The functionality to present the synchronized slide shows allows users to navigate through the different steps of the presentation. As an MPCC application, the GPM also exports functionality to move and duplicate controllers and presentations, adapt to different active spaces, and partition the application among multiple devices. Figure 4 shows the application running in our prototype active space. We have implemented and currently use the GPM to present slide shows in our experimental active space, which implements the full functionality described in this section.

Fig. 4. Example of an active meeting room in which the presentation manager is deployed

6.1 Application Design and Implementation

The GPM is composed of five components: Model, Presentation, Controller, Editor, and Coordinator, as shown in Figure 5. The GPM implements the first four components and reuses the default Coordinator provided by the application framework; the GPM does not require any special behavior from the Coordinator.

The *GPM model* uses an acyclic directed graph to model the synchronized slide show presentation. Nodes in the graph store information about the slides to present and the displays to use, while arcs define the transition order. The GPM model defines an abstraction called a *virtual display* that decouples the slide show views from specific environmental resources. While editing, users assign the slides to the virtual displays, which will be mapped to real displays contained in the ubiquitous computing environment. When the state of the model is changed by a controller, the model sends a notify event to all presentations. Presentations affected by the model's state change contact the model to retrieve the new changes and update their presented information. The GPM Model leverages the event functionality implemented by the *Gaia* middleware infrastructure to notify the GPM presentations.

The *GPM presentation* provides functionality to display the contents of a slide. When instantiating a GPM presentation, the application assigns it an id that corresponds to one of the virtual display ids used by the GPM model. During the presentation of a synchronized slide show, the GPM model sends update events including a virtual display id and information specific to the update. GPM presentations compare the received virtual display id with their assigned id to decide whether or not to they need to update their assigned slide.

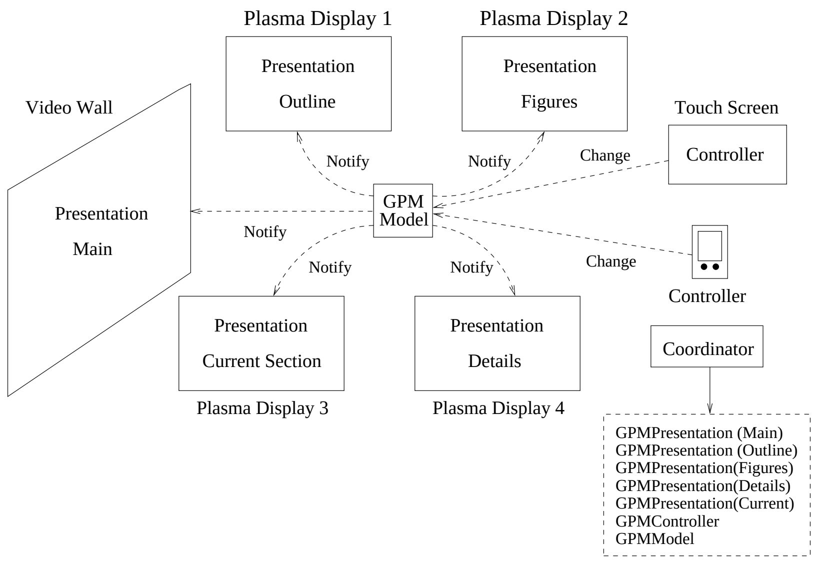


Fig. 5. Schematic of the GPM application architecture

The *GPM controller* is used during the presentation of the synchronized slide show and provides functionality to start and stop the presentation, and move to the next and previous step.

The *GPM editor* is a design-time tool used to edit the synchronized slide show presentation. It provides functionality to create and delete steps, create and delete virtual displays, select existing Power Point presentations, assign slides to virtual displays, and save the synchronized slide show presentation.

Table 1 presents the AGD defined for the GPM. The model, for example, is implemented by a component named CORBA/GPMModel, requires the name of a previously created synchronized slide show as an input parameter, has a cardinality of one (a GPM application has exactly one model), and requires an ExecutionNode device (device with functionality to execute components) running on Windows 2000.

6.2 Using the Application

This section describes the different mechanisms involved in using the GPM application in a real environment: a meeting room equipped with a variety of resources including four 61" plasma displays, 6 touch screens, two Sound Web audio systems, badge detectors, and IR beacons. Mobile devices communicate with the applications and services in the active space via 802.11 wireless. We describe the usage of this application for a group presentation in the active meeting room, consisting of the following actions: creating an ACD for the active meeting room, entering the meeting room, registering a handheld device,

Table 1. Examples of a presentation manager AGD (left) and ACD (right)

```
Model {
  ClassName    CORBA/GPMModel
  Params       -f <fileName>
  Cardinality  1 1
  Requirements device=ExecutionNode
               and OS=Windows2000
}
Presentation {
  ClassName    CORBA/PPTPresentation
  Params      -i<VirtualDisplayID>
  Cardinality  1 *
  Requirements device=Display
               and OS=Windows2000
}
Controller {
  ClassName    Exec/VCRController
  Cardinality  1 *
  Requirements device=Touchscreen
               and OS=Windows2000
}
Coordinator {
  ClassName    CORBA/Coordinator
  Cardinality  1 1
  Requirements device=ExecutionNode
               and OS=Windows2000
}
```

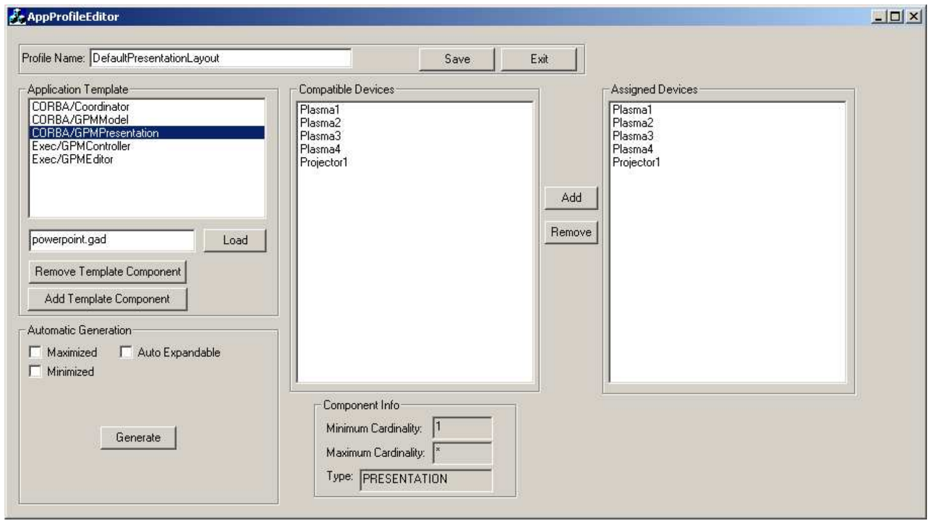
```
Application = {
  Model = { {
  ClassName ="CORBA/GPMModel",
    Hosts = {{ "amr1.as.edu" },}
  } },
  Presentation = {
    ClassName ="CORBA/GPMPresentation",
    Hosts = { { "plasma1.as.edu","-i Outline" },
              { "plasma2.as.edu","-i Current" },
              { "projector.as.edu","-i Main" },
              { "plasma3.as.edu","-i Figures" },
              { "plasma4.as.edu","-i Details" }
            }
    } },
  Controller = { {
    Classname ="Exec/GPMNavigationController",
    Hosts = {{ "touchscreen1.as.edu"},}
  } },
  Coordinator = { {
    ClassName ="CORBA/Coordinator",
    Hosts = {{ "amr2.as.edu","" },}
  } },
}
```

mounting personal data storage, starting the application, and interacting with the application.

Creating an ACD for the Active Meeting Room. The application framework provides a tool to generate ACDs from an AGD based on the resources available in an active space, as shown in Figure 6. This tool parses the GPM AGD and presents a list of devices contained in the active meeting room compatible with the requirements specified by each application component. When the specialization is finished, the specialization tool generates an ACD customized for the active meeting room where the presentation will be held.

ACDs are space dependent and user specific. Different users may have different configuration preferences for the same application and the same active space. The specialization mechanism stores the resulting ACD in the users' personal storage and attaches information specific to the context for which the AGD was customized. The customized application description presented in Table 1 is the result of customizing the AGD to the active meeting room.

Entering the Active Meeting Room. The active meeting room is equipped with active badge detectors, iButtons, and infrared beacons to detect people entering the room. The speaker walks in the room with a handheld device (equipped



Profile Name: DefaultPresentationLayout

Application Template:
- CORBA/Coordinator
- CORBA/GPMModel
- CORBA/GPMPresentation
- Exec/GPMController
- Exec/GPMEditor

powerpoint.gad

Automatic Generation: [ ] Maximized [ ] Auto Expandable [ ] Minimized

Compatible Devices:
- Plasma1
- Plasma2
- Plasma3
- Plasma4
- Projector1

Assigned Devices:
- Plasma1
- Plasma2
- Plasma3
- Plasma4
- Projector1

Component Info:
Minimum Cardinality: 1
Maximum Cardinality: *
Type: PRESENTATION

Fig. 6. The specialization tool is used to customize an AGD to the resources available in an active space

with an infrared port) and points it at the infrared beacon. The handheld device receives a reference to the active meeting room that allows the speaker to access available services via the handheld's wireless interface. The handheld uses this reference to register itself with the the active space, thereby becoming a resource of the active meeting room. The speaker then uses the handheld to merge the personal data mount points into the mount server of the room. Based on the current context, the synchronized presentation and the ACD files are visible in the "current" directory.

Starting the Application. Using the graphical directory browser, the speaker finds the file for the presentation in the "current" context directory. The context file system browser allows mapping file extensions to applications. Although this is a common practice in almost every traditional file browser (e.g., Windows Explorer, and KDE File Manager), ubiquitous computing environments require additional functionality for the mapping mechanism. Specifying a single default application for every file extension is not sufficient because an active space allows multiple layouts for the same application. Furthermore, different users may have different preferred configurations. The file browser solves this problem by selecting the ACDs in the speaker's "current" directory and allowing the speaker to choose one of these ACDs. This mechanism presents the speaker with ACDs associated to the file extension and relevant to the current context. Once the application is started, the four plasma displays and the projector have an instance of a GPM presentation and the GPM controller is displayed on the touch screen.

The GPM controller is a GUI with four push buttons and is used to start, stop, and navigate through the slides.

Interacting with the Application. The speaker starts the presentation using the controller, which triggers the four plasma displays and the projector to display the first slide of the presentation, according to the synchronization information stored in the model. The flexibility of the application framework allows the speaker to move or duplicate both controllers and presentations to any display in the room, including handhelds owned by attendees. For example, if the speaker wishes to start a video that requires an occupied display, the slide rendered on the display can be moved to an auxiliary display. Since both the video and presentation applications have the same control interface, the controller may be reused by simply attaching it to the video application.

7 Related Work

Our work is related to work involving ubiquitous computing, application construction, and data management. The Easy Living project at Microsoft [6] focuses on home and work environments and includes an infrastructure that allows user interfaces to move, according to user location. We differ in that we change the way in which applications are built by partitioning them across devices. Our work is similar to the Interactive Workspaces [4] from Stanford in that we believe there is a need for a supporting infrastructure or operating system in workspaces. However, we consider the concept of the virtual space in which applications and data are associated to the user and can move with them. The Aura architecture [12] allows tasks to be bound to users, similar to our virtual space concept. We are also considering application partitioning and dynamic composition. The Virtual Home Environment (VHE) model [3] proposes an architecture where mobile users may access their environment (e.g., services) from different locations and devices. The model considers device and network heterogeneity with the goal of presenting a consistent look and feel to services. Our model is more related to an operating system, by treating a space as a programmable entity to assist in the development of interactive ubiquitous applications.

PIMA [1] and I-Crafter [8] propose models for building platform independent applications. Developers define an abstract application that is automatically customized at run-time to particular devices. PIMA and I-Crafter generate applications for a single device, while we consider applications partitioned across devices. The Pebbles [7] project is investigating partitioning user interfaces among a collection of devices. While Pebbles is mostly concerned with issues related to GUIs, our application model focuses on the application structure (logic, control, presentation, and meta-level management), application life-cycle, and application adaptability and configurability.

Early work in integrating context with file access was investigated as part of the ParcTab project [11]. The tab allowed access to files that were meaningful to a particular location. As users moved between office spaces, the file browser would change to display relevant data. While they only considered location in their file system, this seminal work was important in establishing the relevance of context in data access and application adaptation. Research in tangible interfaces has proposed tying digital information to physical objects, which can trigger some action (e.g., file transfer) when they are discovered by a new environment [5]. We expand on this idea by treating the *user* as the physical object that triggers the addition of information into a space. The FUSE research group has developed applications for teamwork support based on active documents [13]. Our work differs in that we allow arbitrary context to be attached to data and we consider user configurations and preferences as part of the user's context data.

8 Conclusions and Future Work

This paper has presented our framework for building applications in future computing environments in which a user's environment is implicitly linked to them and is available as they move between physical spaces. Such an environment diverges from the traditional desktop environment due to factors such as context, mobility, and device heterogeneity. Our framework deals with both the structure of the application and the management of the data, which includes application data, configurations, and user preferences. Application logic is separated from presentation and control and the framework introduces a meta-level component that is used to expose the internal structure of the application so that it may be manipulated. Our framework is built on top of a ubiquitous computing middleware infrastructure we have developed that has allowed us to experiment with several applications. Applications built using the framework are described in generic terms and may be customized for the resources that are available in a particular space. As a user moves between spaces, the application can (un)bind new hardware resources and interfaces when they become (un)available. We believe that future applications will have increased flexibility in terms of choice of devices, partitioning, and migration to accommodate the requirements of User Virtual Spaces. This degree of suppleness is amenable to applications being constructed with a component-based approach, where portions of the application may be swapped, removed, or added.

Our future work will involve increasing the ease with which our applications may be used and configured. We have built several applications that have fit well into the framework. However, we will be constructing more applications to validate our approach to building a wide range of ubiquitous applications. Gaining more experience with building and instantiating ubiquitous applications will help us to refine the framework and the management of application data. Our current implementation does not include access control or authentication. We are currently developing a security architecture to control access to devices, applications, and data.

9 Acknowledgments

We would like to thank Herbert Ho for his work on the implementation of the GPM. We would also like to the thank the anonymous reviewers for providing valuable comments.

References

1. Guruduth Banavar, James Beck, Eugene Gluzberg, Jonathan Munson, Jeremy B. Sussman, and Deborra Zukowski. Challenges: an application model for pervasive computing. In *Mobile Computing and Networking*, pages 266–274, 2000.
2. 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.
3. EURESCOM. Realizing the Virtual Home Environment (VHE) concept in ALL-IP UMTS networks. `http://www.eurescom.de`.
4. 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.
5. Hiroshi Ishii and Brygg Ullmer. Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'97)*, pages 234–241, Atlanta, GA, March 22-27 1997.
6. Microsoft Corp. Easyliving. `http://www.research.microsoft.com/easyliving`.
7. B. A. Myers. Using Hand-Held Devices and PCs Together. In *Communications of the ACM*, volume 44, pages 34–41, 2001.
8. S. R. Ponekanti, B. Lee, A. Fox, P. Hanrahan, , and T. Winograd. ICrafter: A Service Framework for Ubiquitous Computing Environments. In *Ubiquitous Computing, Third International Conference (Ubicomp 2001)*, Atlanta, GA, 2001. Springer.
9. Manuel Roman, Christopher K. Hess, Renato Cerqueira, Klara Narhstedt, and Roy H. Campbell. Gaia: A Middleware Infrastructure to Enable Active Spaces. Technical Report UIUCDCS-R-2002-2265 UILU-ENG-2002-1709, University of Illinois at Urbana-Champaign, February 2002.
10. 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.
11. Bill N. Schilit, Norman Adams, and Roy Want. Context-Aware Computing Applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, 1994.
12. Joao Pedro Sousa and David Garlan. Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments. In *Working IEEE/IFIP Conference on Software Architecture*, Montreal, August 25-31 2002.
13. Patrik Werle, Fredrik Kilander, Martin Jonsson, Perter Lonnqvist, and Carl Gustaf Jansson. A Ubiquitous Service Environment with Active Documents for Teamwork Support. In *Ubiquitous Computing, Third International Conference (Ubicomp 2001)*, pages 139–155, Atlanta, GA, September 30-October 2 2001. Springer.