A Context-Aware Data Management System for Ubiquitous Computing Applications *

Christopher K. Hess
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
ckhess@cs.uiuc.edu

Roy H. Campbell
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
roy@cs.uiuc.edu

ABSTRACT

One of the factors that differentiates ubiquitous computing from traditional distributed computing is context. Context, such as time, location, and situation, allows a system to adapt to the current surroundings in order to facilitate the use of the computational environment. In this paper, we present a file system for ubiquitous computing applications that is context-aware. Context is used to support the types of applications and devices that are found in ubiquitous computing spaces. Novel features of the system include how the view of data adapts to the activity being currently performed and how user data is imported into the local environment. Our system is evaluated as part of a ubiquitous computing infrastructure deployed in a seminar room to investigate issues of performance, scalability, and usability.

1. INTRODUCTION

The seminal research in ubiquitous computing at PARC involved custom built systems, from wireless networking and handheld devices to software that tied the different parts of the system together.[1] Only recently have advances in communications, device miniaturization, and sensor technology in commodity offerings brought the goal of making computation pervasive closer to reality. Many of these environments[2–4] consist of intelligent rooms or spaces, containing appliances (whiteboard, video projectors, etc), powerful stationary computers, and mobile wireless handheld devices, in which unconventional peripherals, such as lighting and temperature sensors, may be incorporated into applications

Relevant issues in ubiquitous computing, which include the mobility of users and applications, adapting to contextual information, and support for heterogeneous devices, require that distributed computing infrastructures be extended to accommodate this new operating environment. We term these environments and the software infrastructure to support them *active spaces*. Active spaces cover a range of environment types, including offices, workspaces, classrooms, and homes and are typically defined by some physical boundaries. Figure 1 shows an example of an active space that we are using to research ubiquitous computing environments. Active spaces introduce challenges in how applications are constructed, how data is managed, and how to support the types of applications found in such environments.

Figure 1. A view of our research space.

Applications are often not run as isolated entities, but as collections of software components that support an activity. An activity may involve one or more people and may last for a certain duration of time. For example, consider a seminar in which people gather in a room to discuss some papers; the activity may include scheduling the seminar for a certain time slot on a weekly basis in a particular room, in which attendance is automatically recorded via some authentication mechanism, the relevant papers are displayed on large wall displays controllable from handheld devices, a scratch pad is made available to take notes, and a ticker tape circulates around the room displays to present relevant information. Once the seminar is finished, the applications may be terminated and the room can be scheduled for the next activity, such as a meeting. Note that certain applications may be automatically launched with minimal or no ensuing human intervention, but they must be able to find relevant information. Users may wish to use some personal information during the activity, such as presenting a slide show, or a late participant may wish to view the activity on a handheld device. Therefore, data access challenges include how application data is managed when users are not directly involved in the process of launching and interacting with an application, how the personal data of physically present users is made available to the

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

local space, and how heterogeneous devices are supported.

Contextual information, such as location, time, or situation, can be used to drive the structure of the data organization, provide hints to locate correct application data, and trigger the availability of data based on the presence of people. In this paper, we present the design and implementation of a file system for ubiquitous computing environments that is context-aware. The system has knowledge of the environment in which it is running and supports the types of applications and devices that are found in ubiquitous computing spaces. Novel features of the system include how data is imported into a space, how the contents of directories change based on contextual information, and how the system adapts to device heterogeneity. Our file system is implemented as a collection of middleware services that run on top of existing native operating systems that provide low-level disk access. We evaluate our system from an application and user perspective within a seminar room that utilizes a ubiquitous computing platform we have developed.

The remainder of the paper continues as follows: in Section 2, we describe an overview of the capabilities of our file system and Section 3 describes the architecture, including the mount and file servers. The paper continues with a description of the current implementation in Section 4 and Section 5 describes an application we have developed for our prototype active space that leverages the file system features. An evaluation of our system is presented in Section 6. Related work is described in Section 7 and Section 8 ends the paper with some concluding remarks and future directions.

2. OVERVIEW

Our context-aware file system (CFS) 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 locating data important for applications, and 3) retrieve data in a format based on device characteristics. The system creates a data namespace for each active space so that information may be programmatically accessed by applications. The system is a hybrid between a database and a file system; the database functionality offers the flexibility to search for relevant information and the file system functionality provides an interface that application developers are familiar with.

Our system allows users to move among active spaces and have their data available to local applications. Users can specify what portion of their personal data to make available and it is dynamically added when they are detected within a space.[†] Therefore, the namespace changes as users physically move in and out of a space.

A context-driven view of the namespace is available to limit the visibility of information to what is important for the current context (e.g., activity). The system allows files and directories[‡] to be tagged with metadata (as a list of type/value tuples[5]), which may be user or application-defined properties (i.e., type, subtype, category) or may be contextual information defined by the environment. Properties are used by applications to determine *what* information they are interested in; context is used to determine *when* that data is made available to the applications. Applications access data by specifying the properties they are interested in (e.g., a music jukebox is interested in music files) and the system uses the current context to determine what portion of the data is relevant. The metadata is converted into a directory representation so that applications can easily access data. For example, in our seminar scenario described above, a PDF viewer application can simply open a directory representing the current papers. The file system uses the current location, situation, and time information along with the fact that papers are requested to find the correct files for the application. The contents of this directory automatically change every week, as papers are added and old papers time out. However, from the application point of view, it simply opens the same directory every week and finds the relevant material. By separating the context knowledge from applications and placing it in the infrastructure, applications may run unchanged in different situations.

Heterogeneous devices and user preferences are supported through the use of *dynamic types*. Since some devices may be limited in how they may present data due to device characteristics (e.g., graphic context) or a user may wish to receive data in a certain format (e.g., audio), the system allows applications to open a data source as a desired type. The system attempts to convert the data to the requested format if the appropriate (set of) transformation(s) is available.

3. ARCHITECTURE

In this section, we describe the architectural design of our implementation and describe how context is integrated into the file system. The architecture is composed of mount servers that manage the layout of the namespace and file servers that provide access to distributed disks. The mount servers implement the database functionality and can be queried to find what files are associated to properties and contextual information. The file servers implement the file system functionality and provide an interface for accessing remote files.

[†]The presence of a user may be detected in several ways, including finger print matching, RF badges, cameras, electronic rings, etc.

[‡]For the remainder of the paper, *files* will refer to both files or directories.

3.1. Integrating Context

Our system uses the concept of the traditional mounting mechanism employed by many file systems to specify the placement of data in the directory hierarchy. Each space maintains a collection of data references that constitutes the *space file system*, which consists of space-specific (system) data and remotely-located personal (user) data. Users can carry personal *mobile mount points*, which are pointers to remote storage, that may be merged into the local space file system to make their data available, as shown in Fig. 2.[§] The personal storage of users is dynamically mounted under the directory */users* when they are detected within a space and allows access to all exported data. We call this view *file mode*, since the data is organized as a typical file hierarchy.

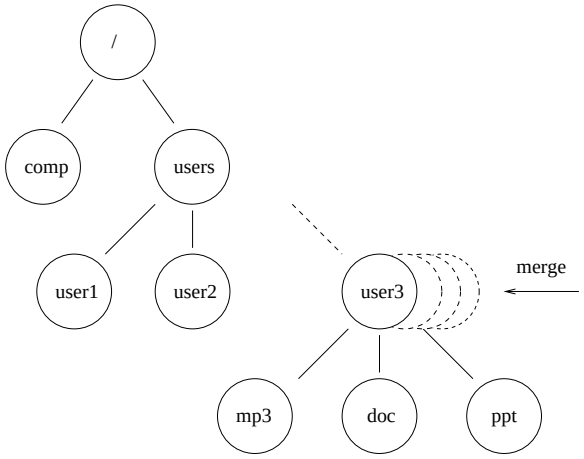

Figure 2. File mode allows users to add their personal data to the local directory hierarchy.

Our context-driven view, called *context mode*, constructs a virtual directory hierarchy based on the metadata tags that represent application properties and contextual information that have been associated to files. Paths are composed of these type/value pairs in the form *[/<type:>/<value>]**, as shown in Fig. 3. Material with the same type/value pairs (i.e., activity relevance) are aggregated into the same directory, which may reside on different file stores owned by different users. It is important to note that although the context directory structure is viewed as a hierarchy, context directories impose no fixed ordering, resulting in a forest rather than a tree structure. The reasons for choosing this syntax are twofold: first, the syntax is simple and limited to the functionality we require; second, it allows our applications to use the same API for reading regular and context-aware files.

Files are tagged with specific metadata by copying them from file mode into a virtual directory representing some

[§] We have a second mechanism in which personal mount points can be fetched from a profi le server.

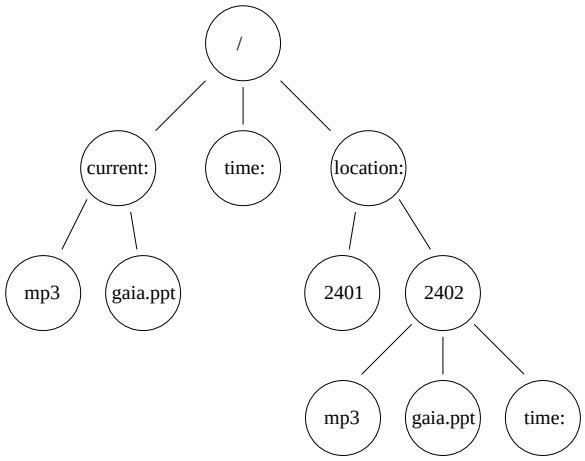

Figure 3. Context mode is a virtual directory hierarchy constructed from metadata attached to files.

properties and context (i.e., sequence of tuples) in context mode. Consider again our PDF viewer application. It is interested in files of type PDF, so we can tag *type == pdf* to files to target them at the viewer. To make these files available only in a certain context, we can add more tuples to specify the target context. Suppose we would like a file to be available during a seminar, we could then add the tag *situation == seminar*. Therefore, we would copy the file to the */type:/pdf/situation:/seminar* directory. We can add any number of other contexts, such as a time duration or location. By tagging metadata to files, we can bind the data to both applications and context.

When accessing data, applications explicitly specify the properties of the data they are interested and implicitly allow the system to determine the correct data for the current context. Appending the special keyword *current:* to a directory path instructs the system to show only the information that pertains to the current context by using the context of the environment (e.g., location, time, situation, weather) together with the application specified properties in the path to display the correct data. For example, our PDF viewer application can simply open the directory */type:/pdf/current:* to find the relevant papers. If any new data becomes available in the directory, applications are notified and may refresh their view.

3.2. Mount Server

Each space maintains a single mount server, which stores a table of mount points used to create the local namespace. Our mount points have been augmented to contain the metadata tags corresponding to properties and environmental context. The mount server is a database that may be searched for mount points that reference relevant information based on a

desired sequence of type/value tuples. Path resolution is performed by obtaining the mount(s) for a given tuple sequence (regular or context), extracting the file server(s) hosting the data, and retrieving the information. Mount points are described in XML format.

```
<CFS:Storage>
  <CFS:Owner>ckhess</CFS:Owner>
  <CFS:Mount>/office</CFS:Mount>
  <CFS:Host>srg181.cs.uiuc.edu</CFS:Host>
  <CFS:Path>C:\home\ckhess</CFS:Path>
</CFS:Storage>

<CFS:Storage>
  <CFS:Owner>ckhess</CFS:Owner>
  <CFS:Host>pc2401-2.cs.uiuc.edu</CFS:Host>
  <CFS:Path>C:\Temp\2381</CFS:Path>
  <CFS:Context>
    <CFS:Type>situation</CFS:Type>
    <CFS:Value>ubi-seminar</CFS:Value>
  </CFS:Context>
  <CFS:Context>
    <CFS:Type>location</CFS:Type>
    <CFS:Value>2401</CFS:Value>
  </CFS:Context>
</CFS:Storage>
```

Figure 4. Example of two mount entries. The first entry imports data into the local space. The second entry illustrates how properties and context can be added to files via metadata tags.

Figure 4 shows two mount descriptions; the first entry triggers the system to allocate a user directory and adds the mount under the namespace */users/ckhess/office*. The second specifies context-aware data that is available to a specific location and situation. The metadata tags are associated to a temporary directory on disk that contains links to the tagged files. Attaching metadata to a file involves generating a context-aware mount point from the tuples defined in the context directory path. We separate the metadata from the actual data so that the metadata can be easily searched, with only a minimal amount of information needed to be transported as users move between spaces. The underlying data is stored as native files, since most existing applications use files as a source of data. In our current implementation, activity context is set in the mount server via a calendar application, while other context values are fixed (e.g., location) or implicit (e.g., time). We are currently working on a general context service that will be responsible for gathering other contextual information from sensors and delivering this information to the mount server. In this case, the calendar will become an activity sensor.

The mount server responds to queries based on the tuples in the path and the environmental context and returns valid mount points that satisfy the query.[¶] The mount server first finds the mounts that contain metadata tags that exactly match the properties in the query tuples. If the keyword *current:* is included in the query, the current environmental context is then used to narrow the result to those mounts in which each tuple either exactly matches the mount tags *or* the mount does not contain a tag of the tuple type. The reasoning behind this is as follows: if the environmental context defines some value that a file has not been tagged with, that context is not considered an important attribute of the file and acts as a wildcard. Therefore, if the environmental context defines some attribute that is not meaningful to a piece of data, the mount point is returned as a result. For example, if the environmental context defines "situation" as an attribute, but an MP3 file is only tagged with a temporal property, the mount is valid, essentially making the MP3 available regardless of what the situation is. However, if the MP3 is tagged with a situation attribute, it will only be visible if the specified situation is active. This restricts the environmental context from hiding data for which the context is irrelevant, since when data is tagged with metadata properties, the full collection of environmental properties that are present in a space may not be known.

[¶] A separate component resolves the returned mounts into directory entries.

3.3. File Server

File servers manage data on the machine on which they are executing and use the native operating system to access disk. Dynamic types are realized through the use of modules, which contain the logic to handle the particular structure of a type to facilitate access to the underlying file format or to convert between formats. The details of the implementation are described elsewhere.[6] We have modules to represent various image formats, presentations, directories, news items, etc. The system is able to convert the type of the stored data to the desired type requested by an application by finding the correct module or modules to handle the conversion (if available) based on the input and output types of the available modules. Some modules support specific attributes that may be used to affect the data type in some way, such as changing the dimensions of a retrieved image object.

4. IMPLEMENTATION

We have implemented our system on Windows 2000, with the mount and file servers developed as application-layer servers written in C++. Communication among mount and file servers is achieved through CORBA method invocations. File servers access local data through the native file system and export portions of the storage to the space file system. Data is accessed by applications through an object-oriented interface designed to simplify retrieval of different data types.[7] The file system is a part of *Gaia*,[8] a ubiquitous computing middleware infrastructure we have developed that turns physical spaces and the resources they contain into a single programmable system. *Gaia* is similar to traditional

operating systems by creating an abstraction layer that applications are written to so that they may be portable across different spaces. The abstraction layer is implemented as a set of core services, including execution nodes, events, entity presence (devices, users, and services), context notification, location, discovery, trading, 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 by mapping them onto the resources available within a space.[9] The core kernel services are started through a bootstrap protocol that initializes the *Gaia* infrastructure. We have deployed *Gaia* in several rooms throughout the our computer science department; our main demonstration room contains four large wall-mounted plasma displays, 5.1 audio system, 15 Pentium-4 PCs, video wall, IR beacons, finger print detectors, wireless and wired Ethernet network, and X-10 devices.

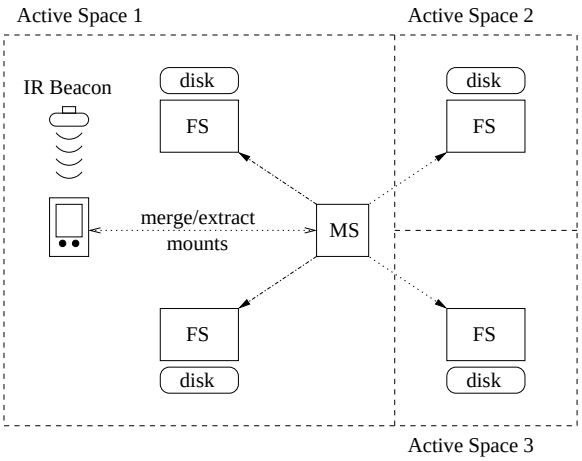

Figure 5. The architecture of the system consists of the mount server (MS) and file server (FS). The mount server defines the layout of storage for a space via mount entries, which contain pointers to storage on remote file servers. A mobile handheld can be used to carry personal mount points, which may be merged into the local namespace.

We have implemented several methods by which a user may merge their personal data into the local space. One method allows users to carry their own personal mount points with them via a handheld (see Fig. 5). We have developed an application for PocketPC devices that is used as the conduit for transporting mount points. When a user enters a space, the device obtains a handle to the space via IR beacon. This handle is the entry point to all services running in the space and is used for further communication with the infrastructure via the 802.11 wireless network. The application includes graphical controls to merge mount points residing on the handheld into the space file system and to extract mount points from the space and store them on the handheld. An alternative methods does not require users to carry any handheld device with them. Users are authenticated in the system via finger print detector through our authentication service (AS). The mount server receives an event when a new user has entered the local space and the user mount points are retrieved from a profile server (PS). Both the AS and PS are domain service that support multiple spaces.

5. APPLICATION

We have implemented a suite of applications to support our weekly seminars. The suite is launched from a calendar and allows us to select and view relevant papers on our large wall-mounted displays, control the synchronized views from any device including handhelds, authenticate users via biometric finger print detectors, announce important information via a ticker tape that sequentially travels across all displays, and take attendance of participants. The file system is used throughout the *Gaia* environment, but due to space limitations, we describe in greater detail one application, the attendance recorder, which illustrates how context is used to find relevant application data.

When the attendance recorder is automatically launched, it opens the file */current:/attendance.att*.[‖] Specifying *current:* instructs the system to find the attendance file that pertains to the current seminar, i.e., based on the seminar location, name, and time. For example, there are several seminars that are held during the week; the context is used to retrieve the correct data file. Note that the application remains unchanged each time it is executed, but the correct *attendance.att* file will be opened because the contents of the */current:* directory changes based on the current activity; the application is relieved from locating files. Each time a user is authenticated within the system via the finger print detector, an event is pushed into the system. The attendance recorder listens for these events and logs the new user to the file so that we can determine the number of times that people have missed the seminar at the end of the semester.

6. EVALUATION

Ubiquitous computing systems are difficult to evaluate since they often stress new functionality and ease of use over pure performance. In our evaluation, we will show the usability of our system for both people and applications during everyday normal use. We do not show the times to transfer files, which is largely dictated by the overhead of CORBA, but we have found the speed to be excellent for our applications.

In this section, we evaluate the response time of performing mount server queries, creating and destroying context directories, and copying and removing files to/from context di-

[‖]Note that we do not have to specify the kind of data with a property since there is only one *attendance.att* file active at any one time.

rectories. We have implemented a shell program and graphical browser, that allows us to manipulate the file system hierarchy. While the graphical browser is used on a daily basis to attach context and launch applications since it only requires us to push buttons on our large wall-mounted touch panels, the command line shell more accurately measures the performance of the file system components since the graphical overhead is eliminated. We have not optimized the code to a great extent and our current implementation runs in debug mode. When using the graphical browser, response times are dominated by the MFC Windows GUI libraries (˜40 ms/operation). Therefore, response times for operations initiated by human users is comparable to those experienced with typical desktop graphical directory navigation tools.

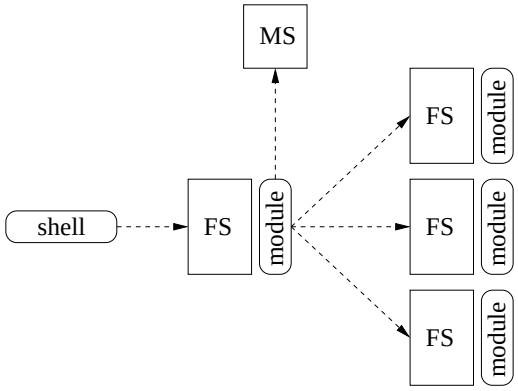

Figure 6. The performance numbers were obtained from data from our active space that we use on a regular basis.

Figure 6 shows our experimental setup. All machines in the experiment are 1.5 GHz Pentium-4 PCs with 256 MB RAM running Windows 2000, service pack 2. All nodes are connected through a 1 Gbps Ethernet switch. Our active space that we use daily is configured in this way and therefore reflects actual response times experienced in normal operation. The shell communicates with a distributed file server, which loads a module that represents our context directories and is responsible for querying the mount server, retrieving mounts, fetching directory entry objects from the remote file servers, and returning them to the shell. All experiments were run 1000 times and results were averaged to obtain the time for a single operation.

The first experiment involved simply measuring the time to perform queries at the mount server in order to investigate scaling issues. We increased the number of mount points that were available in the mount server from 10 to 50. We performed queries with a single tuple (i.e., */location:/2401*), with all mount points containing a single metadata tag so that we could obtain matches. We varied the number of mount points that matched each query from 1 to 4. The results are shown in Figure 7. We expected times to increase linearly with the number of mount points and matches, which the experiments corroborated. Given that the number of mount points that each person adds to the system is limited, the scale of the system is loosely proportional to the number of people that fit in a room. During typical use, scale has not been an issue for the types of activities that we are supporting and the number of people involved in those activities.

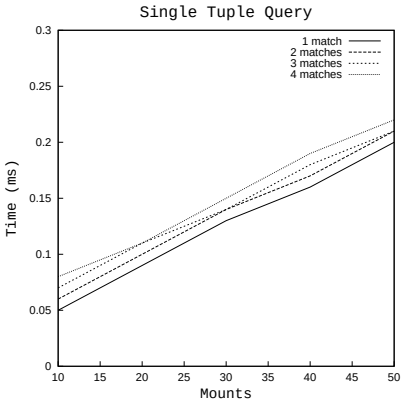

Figure 7. Response times increase linearly with the number of mounts.

In the next experiment, we determined the time for various context mode operations in order to measure the end-to-end response time of the overall system. We used the mount points available in our active space that have accumulated over time, which number approximately 50. The first of these experiments listed the contents of the root directory, which contained 8 entries. In this case, no data file servers are contacted other than the initial module (the center file server in Fig. 6), since the virtual directory is constructed entirely from the contents of metadata tags in the mount server. Each listing took 0.711 ms. Next, we listed two directories that contained files, the first which required that a single file server be contacted to retrieve 4 items and the second which required contact with two file servers to retrieve 4 items from each. Results for each listing was 1.720 ms and 2.825 ms, respectively. Finally, we tested the time it takes to create/remove context directories and copy/remove context files. Creation and destruction took 8.948 ms and 36.669 ms, respectively. The main reason directory destruction is more time consuming is that the system makes sure that the directory is empty before it removes it, therefore, adding an extra directory access. The time for copying and removing a file to a context directory was 11.303 ms and 9.021 ms, respectively. Overall, the system is responsive for human users and applications are able to locate and transfer data at an acceptable rate for those that we have developed. When using our graphical browser, context mode directory navigation feels natural with no noticeable delays.

In using our system, we experienced a situation where

performance can be significantly degraded. If a user enters a space and a file server hosting their personal data cannot be reached, the system hung until the server could be reached. However, sometimes we had outdated mount points or a particular laptop was not attached to the network. In order to alleviate this problem, we decreased the timeout when a file server could not be reached down to 2 seconds. However, one can imagine that different types of applications will require a different response to inaccessible servers. For example, an MP3 player may skip any music files that it cannot retrieve and continue. However, other applications, such as the attendance application, may want to wait until the file server is reachable again. These observations will dictate design changes to the system and are areas of future research.

7. RELATED WORK

Early work in integrating context with file access was investigated as part of the ParcTab project.[10] 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. The Stick-e document framework[11] describes information in SGML format that includes data and context information. The framework includes several modules that allow Stick-e documents to be created, managed, triggered, and shown. When a specified context matches an available document, a trigger makes the data available. This framework has some similarities to our system, but is generally a push-based system. We are also interested in studying how applications could remain unchanged and have the data adapt to the activity. Context-aware retrieval (CAR)[12] has been studied with regards to presenting users with important information based on their current context. A proactive context-aware cache is proposed in which the user's (future) activity can help to warm a store of anticipated useful documents. The context information can be gained from the users location and context history. The Personal Server[13] is a small data server that can be carried around with a user and offers no graphical display, but rather uses larger displays that are available within the local space. This was one of our early scenarios for use of our system as well. Our mobility support relies on users carrying data references on handheld devices that are merged with the infrastructure. However, users can also integrate the data on their laptop into the local environment by running a file server and exporting data.

Presto[14] developed a document management system that uses property tags to organize data so that different users may have personalized views of the data hierarchy. Any number of properties can be added to data files. They developed a graphical desktop where files could be grouped together and properties could be dropped on the desktop to display the items that exactly matched the active properties. Our system has some similarities to this system in that our contexts act like their properties. However, their properties act as filters, where each added property narrows the list of data matching the property list. We differ from this work in several respects. First, we use implicit environmental context to display relevant material. Second, our queries are different from filtering. In our system, some environmental contexts may not be relevant to a given application, and we therefore ignore such contexts. Some of our queries would fail to match items in the Presto system. Third, our system is targeted at organizing data for applications in addition to users. Finally, we incorporate the mobility of users, allowing them to merge their data into a new space.

The syntax for our virtual file hierarchy is similar to the Semantic File System.[15] Semantic indexes data sources when files and directories are created and updated and the path components are used to provide associative data access. While our work is basically different from the Semantic File System, their research developed fundamental concepts of virtual file systems. Gopal *et.al.* have extended the ideas of the Semantic File System to include semantic directories to groups of related material.[16] Their system is able to accommodate multiple mounts with similar semantic meaning to aggregate files and directories. The Prospero File System[17] constructs a virtual namespace that is used to group logically related material together in different ways to facilitate data organization. A main difference between our system and the above described file systems is that our system explicitly integrates information from the environment to affect how and when data is made visible. A further difference is that our virtual mode directories are not hierarchical, i.e., they may be traversed in any order, whereas the others organize data in an ordered fashion. Since contextual information is typically not ordered, such systems would not suite the type of environment we are targeting. In addition, we have introduced mobile data references and dynamic data types, both considerations that we believe must be addressed in a file system for ubiquitous computing environments.

Several systems have investigated providing data adaptation to different device types.[18,19] These services are similar to our dynamic data types by providing impedance matching of data types to convert data formats. Our system hides the details in the file system abstraction to simplify programmability.

8. CONCLUSIONS

Ubiquitous computing is poised to revolutionize computer systems design and use. New challenges arise beyond what is required in typical distributed systems, such as how to integrate context, account for mobility, and accommodate heterogeneous devices. Applications are no longer tied to a single machine, but may be mapped to a physical space based

on resource availability or the role that the space is playing. Applications running in a physical space may be affected by context, such as the location, what is happening in the space, or who is present. Our file system is used to address these issues by limiting the scope of information to what is meaningful for the current context, such as application configurations or application data, making personal information available conditioned on user presence, and adapting content for resource availability through dynamic data types.

Mount points are mobile and can be injected into an environment to make personal data locally available. We have augmented the mount point concept to include metadata tags that are used to create a virtual directory hierarchy, which allows data to be bound to a specific application and context. Through this mechanism, the system determines what data is relevant for an activity, thereby allowing applications to run unchanged in different contexts. Our experiences with the system have shown it to perform well in our seminar application suite. Further use of the system will drive future design decisions in terms of functionality, fault tolerance, and ease of use.

REFERENCES

1. M. Weiser, "The Computer for the 21st Century," *Scientific American* **265**(9), pp. 66–75, 1991.
2. R. Brooks, "The Intelligent Room Project," in *Proceedings of the Second International Cognitive Technology Conference (CT'97)*, pp. 271–278, (Aizu, Japan), August 1997.
3. T. Kindberg and J. Barton, "A Web-Based Nomadic Computing System," *Computer Networks* **35**(4), pp. 443–456, 2001.
4. B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. A. Shafer, "EasyLiving: Technologies for Intelligent Environments," in *Handheld and Ubiquitous Computing (HUC)*, pp. 12–29, 2000.
5. J. Pascoe, "Adding Generic Contextual Capabilities to Wearable Computers," in *International Symposium on Wearable Computers (ISWC)*, pp. 92–99, October 1998.
6. C. K. Hess and R. H. Campbell, "The Role of Users and Devices in Ubiquitous Data Access," Tech. Rep. UIUCDCS-R-2001-2226 UILU-ENG-2001-1733, University of Illinois at Urbana-Champaign, July 2001.
7. C. K. Hess, F. Ballesteros, R. H. Campbell, and M. D. Mickunas, "An Adaptable Data Object Service Framework for Pervasive Computing Environments," in *6th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'01)*, pp. 31–45, (San Antonio, TX), January 29-February 2 2001.
8. M. Romàn, C. K. Hess, R. Cerqueira, A. Ranganat, R. H. Campbell, and K. Nahrstedt, "Gaia: A Middleware Infrastructure for Active Spaces," *IEEE Pervasive Computing* **1**, pp. 74–83, October-December 2002.
9. C. K. Hess, M. Román, and R. H. Campbell, "Building Applications for Ubiquitous Computing Environments," in *First International Conference on Pervasive Computing (Pervasive 2002)*, pp. 16–29, Springer-Verlag, (Zurich, Switzerland), August 26-28 2002.
10. B. N. Schilit, N. Adams, and R. Want, "Context-Aware Computing Applications," in *IEEE Workshop on Mobile Computing Systems and Applications*, pp. 85–90, (Santa Cruz, CA), 1994.
11. P. J. Brown, "The Stick-e Document: A Framework for Creating Context-Aware Appliations," in *Electronic Publishing (EP'96)*, pp. 259–272, (Palo Alto, CA), January 1996.
12. G. J. F. Jones and P. J. Brown, "Context-Aware Retrieval for Pervasive Computing Environments," in *First International Conference on Pervasive Computing (Pervasive 2002)*, pp. 47–56, Springer-Verlag, (Zurich, Switzerland), August 26-28 2002. (short paper).
13. R. Want, T. Pering, G. Danneels, M. Kumar, M. Sundar, and J. Light, "The Personal Server: Changing the Way We Think about Ubiquitous Computing," in *4th International Conference on Ubiquitous Computing (UbiComp 2002)*, (Goteborg, Sweden), September 29-October 1 2002.
14. P. Dourish, W. K. Edwards, A. LaMarca, and M. Salisbury, "Presto: An Experimantal Architecture for Fluid Interactive Document Spaces," *ACM Transactions on Computer-Human Interaction* **6**(2), pp. 133–161, 1999.
15. D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O. Jr., "Semantic File Systems," in *Proceedings of the 13th Symposium on Operating System Principles*, pp. 16–25, 1991.
16. B. Gopal and U. Manber, "Integrating Content-based Access Mechanisms with Hierarchical File Systems," in *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, pp. 265–278, (New Orleans, LA), February 1999.
17. B. C. Neuman, "The Prospero File System: A Global File System Based on the Virtual System Model," *Computing Systems* **5**(4), pp. 407–432, 1992.
18. A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir, "Adapting to Network and Client Variation via On-Demand Dynamic Distillation," in *ASPLOS-VII*, (Boston, MA), October 1996.
19. B. Raman, R. H. Katz, and A. D. Joseph, "Personal Mobility in the ICEBERG Integrated Communications Network," Tech. Rep. UCB/CSD-99-1048, UCB EECS, May 1999.