A Context File System for Ubiquitous Computing Environments *

Christopher K. Hess Roy H. Campbell

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

{*ckhess, roy*}*@cs.uiuc.edu*

http://choices.cs.uiuc.edu/gaia

Report No. UIUCDCS-R-2002-2285 UILU-ENG-2002-1729

July, 2002

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

A Context File System for Ubiquitous Computing Environments

Abstract

One of the distinguishing factors that differentiates ubiquitous computing from traditional distributed computing is context. Context 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 may be associated to files and directories and is used to limit the scope of available data to what is important for the current task, aggregate related material, and trigger data type conversions, therefore simplifying the tasks of application developers and users of the system. Our file system constructs a virtual directory hierarchy that is based on these associations and is implemented using an internal mounting mechanism, where mount points are owned by users and contain context tags. Mount points can be carried with a user via a mobile handheld device or retrieved automatically from a personal file server and injected into the current environment to make personal storage available to applications and other users.

Contents

1 Introduction **3**

2 CFS Overview **4**
2.1 Data Availability 5
2.2 Data Organization 6
2.3 Dynamic Types 8

3 Architecture **8**
3.1 Mount Server 8
3.2 Resolver 10
3.3 File Server 10

4 Virtual File Hierarchy **11**
4.1 Directory Listing 11
4.2 Directory Creation 11
4.3 Attaching Context 12

5 Implementation **13**

6 Related Work **14**

7 Conclusions **15**

8 Acknowledgments **16**

1 Introduction

Recent activity in ubiquitous computing research is attempting to merge the virtual and physical worlds by incorporating an array of software, hardware, and physical entities into next generation computing environments [Wei93, MIT, Hew, Mic]. These environments consist of intelligent rooms or spaces, containing appliances (whiteboard, video projectors, etc), powerful stationary computers, and mobile wireless handheld devices. The large collection of devices, resources, and peripherals must be coordinated and access to them must be made simple. Users should be able to easily interact with these devices and it should be easy for developers to construct applications utilizing any of the available resources. We term these environments *active spaces*. Active spaces cover a range of environment types, including offices, workspaces, classrooms, and homes, and have several defining characteristics, which include the following:

- *Mobile.* Users and their devices are mobile. Users may possess handheld devices, which can be used to interact with the resources and applications resident in the physical space and these devices can become integrated with the other resources of the space. Users can move between spaces and their environment (i.e., applications, state, data, etc.) can move with them.

- *Heterogeneous.* A wide variety of devices exist, which exhibit different resource requirements and form factors. Application can be mapped onto different spaces depending on the available resources.

- *Interactive.* Applications running in the space are interactive, may be controlled remotely, and may be notified of event changes that take place in the space or applications.

- *Networked.* The software components running in the physical space are connected through a wired network that supports multicast to discover services within the space. Mobile devices are able to interact with the services and applications in a space through wireless networking, such as IEEE 802.11 and infrared. Tasks are performed within a space, while the devices are present and connected.

- *Distributed.* Applications may utilize any distributed resource within a space and may be partitioned to increase the amount of information presented to the user. Portions of an application may also be moved or duplicated on different devices.

Active spaces (or simply *spaces*) are often designated for specific tasks; classes are held in lecture halls, shopping is done in stores, and work completed in offices, and therefore typically have a context associated with them. This context information can be used to determine which information is meaningful in a particular space. For example, a user may configure a presentation application based on personal preferences or resources available in a space, such as number and type of displays. Different configurations may be available and the user should be able to choose among them when launching an application. Furthermore, different users may have their own personal configurations, and the correct configurations should be displayed depending on who is launching the application.

Continually running or automatically launched 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. For example, a seminar application may automatically be started

every week at a certain time, triggered by a calendar or when the moderator arrives. Suppose the application displays the papers that are to be discussed that week. The application knows that it requires papers. However, those papers may be specific to the seminar, which is held at a certain time each week in a designated room. Therefore, the environmental context (i.e., seminar, time, etc.) can be used to display the correct material for the given task.

Users are highly mobile in active spaces and should not be burdened with manually transferring files or data, be it configurations, preferences, or application data from one environment to another. The environment should assist in making personal storage automatically available in the users' present location. Storage becomes implicitly linked to a user and can "follow" them around, becoming available whenever they enter a new space. Therefore, the physical location of the user triggers the automatic configuration of the user's environment.

Active space environments can be populated by a wide variety of computing devices. Applications may no longer make assumptions about the types and characteristics of devices that a user may possess or that are resident in a particular space. If a user wishes to use a device that does not support the original data format or prefers to receive some data in a different format, the infrastructure must make an attempt to present the data in the desired format. Applications should not be bothered with the complexities of such conversions; they should gain access to data in a particular format by simply opening the data source as the desired type and the system should be responsible for automatically adapting content to the desired format.

To address the foregoing issues, this paper presents a context-aware file system (CFS) targeted at ubiquitous computing environments. CFS uses context to facilitate data access for mobile users, to aggregate related data, and to drive dynamic data types to support heterogeneous devices and user preferences. Novel features of the system are the implicit connection between users and data, the virtual directory structure constructed using context information, how that information is associated with files and directories, and how data is imported into a space. The remainder of this paper discusses the design and implementation of our file system to support data access in active spaces and continues as follows. Section 2 describes an overview of the capabilities of our file system and Section 3 describing the architecture of the system, including the mount server, resolver, and file server. The paper continues with Section 4, which describes the virtual directory hierarchy and how file system primitives are used to manipulate the hierarchy and Section 5 describes the current implementation. Related work is described in Section 6 and Section 7 ends the paper with some concluding remarks and future directions.

2 CFS Overview

A distinguishing feature of ubiquitous computing is the use of context to affect applications. Context can be used to automate tasks or adapt an application to the current surroundings [SDA99, DAS99]. 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 users, and 3) retrieve data in a format based on the context of user preferences or device characteristics.

CFS categorizes context into *external context* and *internal context*. We define external context as any information that is gathered from the surroundings, outside the scope of the current device or application, which the system uses to organize data so that material important to the current task is aggregated in well-known locations, thereby allowing relevant files and directories to be

easily discovered by applications and other users. We define internal context as any information that is determined from the current device or application, for example, device characteristics (i.e., graphic context) or user preferences such as data format. This form of context is used to change the type of a data source so that it is compatible with application needs.

2.1 Data Availability

Location information is used by CFS to transparently incorporate a user's personal data into the environment encompassing the current space in which they are located. The file 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 that constitutes the *space file system*, which consists of space-specific (system) data and remotely-located personal (user) data. Users maintain personal *mobile* mounts that may be merged into the space file system to make their data available within the space and act as pointers to remote storage, as shown in Fig. 1.

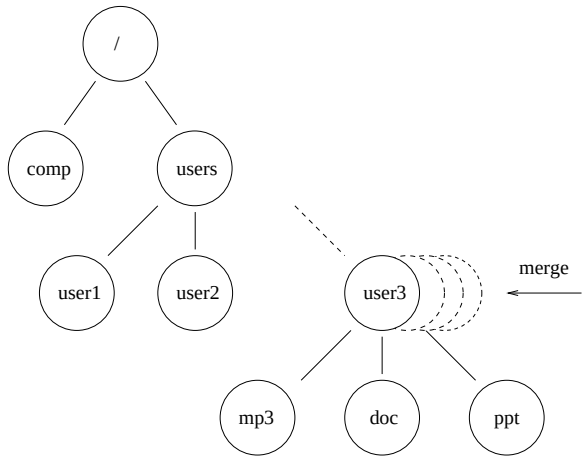

Figure 1: The mount points of mobile users may be dynamically added to the space file system to make data available to applications running in the space. The system adds a temporary user directory under which personal mount points are added, which reference remote file servers.

The personal storage of users is dynamically mounted under the directory */users* when they are detected within a space. [1] Since many users may be present in a space, each user is allocated a temporary directory using their unique user name. [2] Personal mount points may be carried with a user via a mobile handheld device or automatically retrieved from a home server and merged into the current environment to make personal storage available to applications and other users. Our current implementation employs the latter approach. This allows users to move between spaces and be able to find their data in a consistent location within the directory hierarchy of the space. Therefore, the space file system namespace changes as users physically move in and out of the space.

[1] The presence of a user may be detected in several ways, including RF badges, cameras, electronic rings, etc.
[2] Unique system-wide user names are mapped to system-dependent local names.

2.2 Data Organization

While mobile mounts enable all of a user's data to be available to a space when they are present, an alternate view of available data that is context-driven is used to organize data by limiting the visibility of data to what is important for the current context, which may include user preferences, application configurations, and application data. Since each user may place their own data in a different location in their own private hierarchy, the task of finding data of another user can become difficult for automated processes, during a group collaborative task, or when a user must decide from a choice of application configurations. CFS uses context to organize data so that related material are co-located by constructing a virtual directory hierarchy, where irrelevant information is pruned from view. Paths are composed of context types and context values (a concrete value for a given type) of the form */<type1:>/<value1>/<type2:>/<value2>...* 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 files and directories. 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. Context paths do not follow a strict hierarchy and can be traversed in any order, but are convenient for locating information. The system allows context to be attached (detached) to (from) files and directories [3] by generating context-aware mount points, where mount points are owned by users and contain context tags. Once a context is associated to a file, the data is visible in the directory representing the context, as shown in Fig. 2. We chose to represent the standard file hierarchy (which we refer to as *file mode*) and the virtual hierarchy (*context mode*) as distinct hierarchies for cleaner separation, although we could have combined them with a single root. [4]

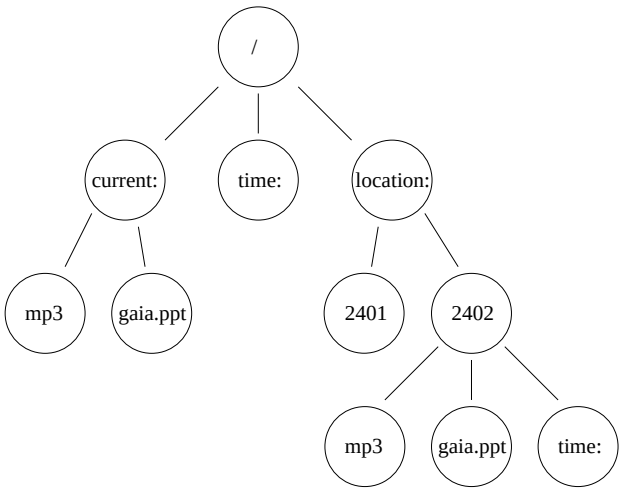

Figure 2: An abridged view of the context mode virtual directory hierarchy. The virtual (context) file hierarchy aggregates files and directories with the same context associated to them. The */current:* directory contains all files for the current context only. Note that the *time:* context directory appears twice in the figure, illustrating that no fixed hierarchy is imposed on context.

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

[4] The system can infer which mode is in used from the structure of the path names. Context mode directories append a colon after context types.

Context types are user or application defined. Some examples of useful contexts are:

- *Location* - represents the location of the current space, such as a specific room number.
- *Situation* - refers to an activity that is taking place within a space, for example a meeting or lecture.
- *Space* - represents the type of space, e.g., office or store.
- *Time* - can specify a valid duration for data.
- *Weather* - the current weather, which can be used for tele-presence applications.

The virtual file system hierarchy is based on what contexts have been attached to files. Appending the special keyword *current:* to a path specifies that the directory should contain all files that pertain to the current context. When this is done, CFS uses the current context properties of the environment (e.g., location, time, situation, weather) together with user specified properties in the path to display the correct application data. For example, returning the seminar application described earlier, the application may require all papers that are to be discussed during a seminar. The application simply opens the directory for the current papers, e.g., */type:/papers/current:*. The file system will use 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 may 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. This is also convenient because all the papers can collected in the same real directory, so that the papers of previous weeks can be found.

Recall that the data may be located in the personal repositories of individual users. Even though the data of a single user or group of users may be dispersed among several remote machines, that data is aggregated and presented as a single source with only pertinent information visible. Name clashes are handled by indexing different files with the same name.

The virtual directory hierarchy forms a simple query language that can be used to determine what types of context are attached to files. The path of the directory is used to identify the context of interest. For example, to determine which files have the context of *location == 2401 && situation == meeting* attached to them, one may enter the */location:/2401/situation:/meeting* directory. This directory contains all files with the specified context attached to them (which may reside on multiple machines for different users), plus directories for the remaining contexts that are not included within the path name (e.g., group, time, and space). While the above example illustrates that queries naturally support *AND* boolean operations, *OR* queries are supported by attaching different contexts to the same file.

The queries that are performed are not simply a combination of the current context and the application requested material; the space may define a context that is irrelevant to the current task. For example, the context "the weather is sunny" may be meaningless to the seminar application, but may make sense for a tele-presence application. The system resolves this issue by ignoring any context that is valid in the environment, but that is not explicitly associated with the data.

The file system allows a specific context to be attached to data by simply copying it to a context directory. Presenting context through the file system interface allows standard file system primitives (i.e., *rename*, *remove*, *copy*, *mkdir*) to be used to attach and detach context to files. The details of how the virtual file hierarchy is manipulated is explained in Section 4.

2.3 Dynamic Types

CFS supports device heterogeneity through dynamic data types. Since different devices have different characteristics and users may have preferred ways to represent data (e.g., text v. audio), the system allows applications to specify the desired format of the data when it is first accessed. Access to data is handled through modules, which contain the logic to handle the particular structure of the data type to facilitate access to the data or to convert between formats [HCM01]. For example, 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). For example, a presentation file may be opened as GIF images, pixmaps, or bitmaps. However, some files do not contain any well-defined structure. Such files may be represented as a stream of bytes (by opening the source as bytes), thereby supporting traditional file system semantics. 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.

3 Architecture

In this section, we describe the architectural design of the system. The main components of the file system are the mount server, resolver, and file server, as shown in Fig. 3.

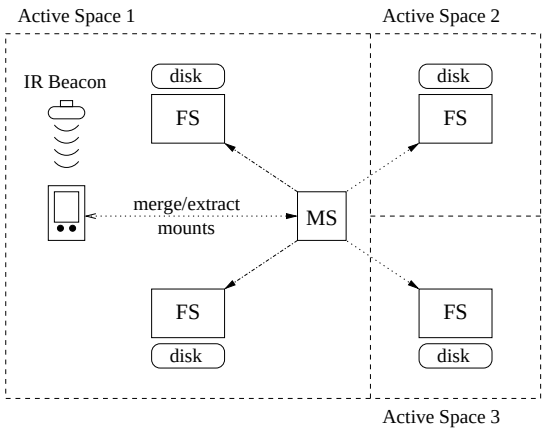

Figure 3: The architecture of CFS consists of the mount server (MS), resolver, and file server (FS). The mount server defines the layout of storage for a space via mount entries, which contain pointers to storage on remote file servers. A mobile handheld is used to carry a user's personal mount points. They may be merged into a space to make remote personal storage available to applications in the space.

3.1 Mount Server

Each space maintains a single mount server, which stores the current storage namespace layout of the space file system and is essentially a database for searching for relevant material. The mount server contains both system and user storage mappings as described in Section 2.1. These mappings

acts as meta-data for files on disk. We split the meta-data from the actual data so that the meta-data can be easily searched, but only a minimal amount of information needs to be transported as users move among spaces. The underlying data is stored as files, since most existing applications use files to store their data. The personal storage description of a user may contain a number of mount points to organize their directory hierarchy. By convention, this storage gets added to the */users/<user>* namespace, which the system automatically creates for the user. When the user leaves a space, the user's directory mappings are automatically deleted from the space file system, which restricts access unless the user is physically present. The mount server removes the need for users to manually transfer files that they will need when they move between spaces.

The mount server is initialized with an XML configuration file, which contains the space-specific system mounts. This file contains entries that specify which machines export a part of their storage, [5] how that storage gets mapped into the space file system namespace, to whom the descriptions belong, and (optionally) what context is associated to the data.

```
<CFS:Storage>
  <CFS:Owner>system</CFS:Owner>
  <CFS:Mount>/scratch</CFS:Mount>
  <CFS:Host>pc2401-1.cs.uiuc.edu</CFS:Host>
  <CFS:Path>C:\Temp</CFS:Path>
</CFS:Storage>

<CFS:Storage>
  <CFS:Owner>system</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>meeting</CFS:Value>
  </CFS:Context>
  <CFS:Context>
    <CFS:Type>location</CFS:Type>
    <CFS:Value>2401</CFS:Value>
  </CFS:Context>
</CFS:Storage>
```

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

<CFS:Storage>
  <CFS:Owner>ckhess</CFS:Owner>
  <CFS:Host>srg181.cs.uiuc.edu</CFS:Host>
  <CFS:Path>C:\Temp\15687</CFS:Path>
  <CFS:Context>
    <CFS:Type>situation</CFS:Type>
    <CFS:Value>meeting</CFS:Value>
  </CFS:Context>
  <CFS:Context>
    <CFS:Type>location</CFS:Type>
    <CFS:Value>2401</CFS:Value>
  </CFS:Context>
</CFS:Storage>

<CFS:Storage>
  <CFS:Owner>mroman</CFS:Owner>
  <CFS:Host>barna.cs.uiuc.edu</CFS:Host>
  <CFS:Path>C:\Temp\34981</CFS:Path>
  <CFS:Context>
    <CFS:Type>situation</CFS:Type>
    <CFS:Value>meeting</CFS:Value>
  </CFS:Context>
  <CFS:Context>
    <CFS:Type>location</CFS:Type>
    <CFS:Value>2401</CFS:Value>
  </CFS:Context>
</CFS:Staorage>
```

Table 1: Example of several system (left) and user (right) mount descriptions. Entries which do not contain *Mount* tags are used in the construction of the virtual directory hierarchy.

[5] Mount and file servers are implemented at the application-level.

Table 1 shows five example mount descriptions. The first description is a system mapping that specifies that machine *pc2401-1.cs.uiuc.edu* is exporting its *C:\Temp* directory and it will get mapped to */scratch*. [6] The second mount is a temporary directory that represents the context *situation == meeting && location == 2401* and has been generated for the local space (system). The remaining three mounts (right) have been dynamically added by users. The third entry specifies a mount that gets add to the namespace as */users/ckhess/office* from a remote machine. The last two mount points are references to context data for two different users. Note that there are no *Mount* tags, signifying that these generate virtual directories.

The mount server maintains the current context of the space in which it is running. In our current implementation, the context is set manually; future versions may be able to detect the context automatically through environmental sensing. However, since the context of a space does not change rapidly and some context values are fixed (e.g., location) or implicit (e.g., time), manual configuration is not a burden. The mount server exports a query interface and acts as a database, which can be used to search for specific mount points, based on the XML description tags, and is used to find mount points during the construction of the virtual directory structure. For example, to determine which files are important to the current task, the mount server is queried for all mount points that match the current context of the space.

3.2 Resolver

The resolver is part of a client-side library that is linked in with applications. [7] The library stores no per application state; state is stored in a small library header file, which is compiled into each application. The resolver maintains a cache of references to machines exporting storage and provides name resolution facilities. The resolver includes a prefix table mechanism [WO86] and, when needed, attempts to make connections to available remote file servers listed in the prefix table. Each remote file server manages the data content on their respective machines and is responsible for allowing access to that data. The resolver maps UNIX-style directory formats to the native machine format. For example, the directory */scratch/junk* would get mapped to *C:\Temp\junk* on host *pc2401-1.cs.uiuc.edu* using the first storage description shown in Table 1.

3.3 File Server

The file server manages data on the machine on which it is executing. Access to each data source is initiated via the file server, which locates the appropriate module, loads the module into memory if not already resident, and returns a handle to the module to the client library.

The ability to dynamically change the type of a data source is achieved through a description graph constructed from an XML module description file. Each module defines an input and output type (stored in the XML file). A module in the system may receive input data from a native file or from another module, that is a (file) module may simple wrap a native file type to facilitate accessing its structure, or a (converter) module may accept the output of another module to convert the data to a new type. When a data source is opened as a specific type, the description graph is consulted to determine whether a module or chain of modules exists to convert the original data type into the desired type. If such a sequence exists in the graph, the file server loads the modules into memory and connects them together. New modules may be added to the system by updating

[6] The user name *system* is reserved for the infrastructure.

[7] We reuse existing applications by wrapping them with our application framework [HRC02].

the XML description file. Data attributes are maintained in the client-side library (transparent to applications) and sent along with remote procedure calls.

4 Virtual File Hierarchy

The virtual file system uses information in the path to perform queries on the mount server to retrieve all mounts that contain the given context(s). Paths are handled differently depending on whether the last component of the path is a context type or context value.

4.1 Directory Listing

When the last part of a path is a context type (e.g. location), the system queries the mount server for all mounts that contain tags for the given context. The directory entries returned by the module are those contexts that appear in the context type tag (only a single entry for redundant entries are returned by the mount server, i.e., when different users have files with the same context attached to them). For example, when an application opens the the */location:* directory, the mount server returns all mounts that include a location tag and the directory entries are the result of the values associated to the tag; the contents of the directory includes all location values that are attached to files. The following listing signifies that there are files in the system that have one (or more) of the four different room context locations attached to them:

```
> cd /location:
> ls
2401         2402         1310
1320
>
```

Listing a context value directory (e.g., 2401 in the above example) shows what files actually have the given context attached to them, as specified by the path. The module queries the mount server for all mounts that match the given context and then contacts all file servers specified in the mounts to retrieve the names of all files and aggregates them to make it appear as if they are located together. In addition, the remaining context names that are not in the current path are also returned, so that they may be navigated further. For example:

```
> cd /location:/2401
> ls
situation:      time:          group:
space:          cfs.ppt        demo.mpg
>
```

From this example, *cfs.ppt* and *demo.mpg* have the context of *location == 2401* attached to them, which may be located on different file servers. Since there is no fixed order for navigating the context hierarchy, opening */space:/office/group:/srg* is equivalent to */group:/srg/space:/office*.

4.2 Directory Creation

Next we address how context directories are created. When an application creates a virtual directory, the system creates a mount point that contains context type/values tags as specified in the path. This mount acts as a place holder until the context is eventually attached to files (described in the following section), so that the system can display the context directory if no files are yet added. For example, creating a directory */location:/2401/situation:/meeting* creates the following mount:

```
<CFS:Storage>
  <CFS:Owner>ckhess</CFS:Owner>
  <CFS:Context>
    <CFS:Type>situation</CFS:Type>
    <CFS:Value>meeting</CFS:Value>
  </CFS:Context>
  <CFS:Context>
    <CFS:Type>location</CFS:Type>
    <CFS:Value>2401</CFS:Value>
  </CFS:Context>
</CFS:Storage>
```

This mount then gets added to the mount server, which is specific to the user that created it and becomes a part of their set of personal mount points. Deleting the directory removes the mount from the mount server.

4.3 Attaching Context

The operation of explicitly attaching context to files is handled by the copy operation, which is a primitive available in the CFS interface. Copying a file to a context directory attaches the context associated with the path to the file by creating a directory on disk for that context and creating a link to the real file in the generated directory. A mount is then generated with the context from the destination directory path, information about the host machine containing the newly created directory, and the path to the directory. If a mount is already available for the given context, the system will simply reuse the associated directory and add the link to it. In the following example, the generated directory *C:\Temp\15687* is used for all files (and links to files) that have the context of meeting and 2401 attached to them (on the given host):

```
<CFS:Storage>
   <CFS:Owner>ckhess</CFS:Owner>
   <CFS:Host>srg181</CFS:Host>
   <CFS:Path>C:\Temp\15687</CFS:Path>
  <CFS:Context>
    <CFS:Type>situation</CFS:Type>
    <CFS:Value>meeting</CFS:Value>
  </CFS:Context>
  <CFS:Context>
    <CFS:Type>location</CFS:Type>
    <CFS:Value>2401</CFS:Value>
  </CFS:Context>
</CFS:Storage>
```

We use mounts to store context information rather than directories on disk because context directories are not hierarchical and having the information in the mount points makes finding and aggregating files with a particular context easier and more efficient.

Implicit attachment of context is handled in a slightly different manner. In this case, when a file is created in one of the current context directories, the current context is used to generate the mount context tags. However, there is no existing reference file to link to; rather, the system creates the file directly in the generated directory representing the current space context. Removing the file deletes the data from the system. In contrast, removing a file to which context was explicitly attached only removes the link, and not the real data.

5 Implementation

We have implemented our system on Windows 2000 and Solaris UNIX platforms, with the mount and file servers developed as application-layer servers written in C++. File servers access local data through the native file system and export portions of the storage to the space file system. CFS is a part of *Gaia* [RHC+02], 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 managing the tasks common to all applications and provides core services, including events, entity presence (devices, users, and services), context notification, 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. The core kernel services are started through a bootstrap protocol that starts the *Gaia* infrastructure. We have deployed *Gaia* in a prototype room containing large four wall-mounted plasma displays, 5.1 audio system, 15 Pentium-4 PCs, video wall, IR beacons, badge detectors, wireless and wired Ethernet network, and X-10 devices.

We have implemented a shell program to perform command line operations, as well as a graphical interface to navigate the file system hierarchy and launch applications. Figure 4 shows a screen shot of our graphical file browser. The browser is shown in the context directory */location:/2401/situation:/meeting*. The file system has aggregated all files that are associated to the same context and displays them together. New context directories may be created by creating a new folder, which internally calls the *mkdir* operation. The two top left buttons toggle the views of the file system (i.e., file mode and context mode), allowing users to easily copy a file to a context directory.

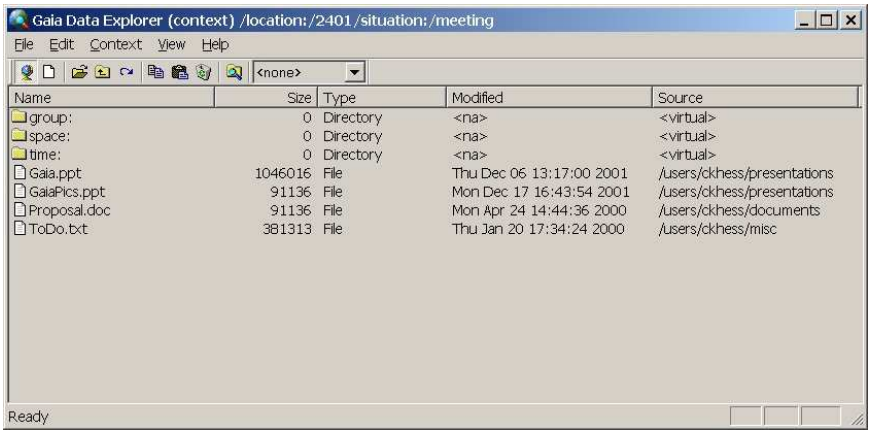

Figure 4: The graphical browser allows users to navigate and manipulate the virtual file hierarchy. Context can be associated to a file by simply copying it to a context directory.

We allow users to carry their own personal mounts with them via a handheld (see Fig. 3). We have developed an application for WindowsCE devices that is used as the conduit for transporting mounts. 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 mounts residing on the handheld into the space file system and to extract mounts from the space and store them on the handheld.

6 Related Work

Early work in integrating context with file access was investigated as part of the ParcTab project [SAW94]. 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 syntax for our virtual file hierarchy is similar to the Semantic File System [GJSJ91]. The system indexes data sources when files and directories are created and updated and the path components are used to provide associative data access. Data extraction is performed using *transducers*, where different transducers can be applied to provide the user with alternate views of data. The system also employs a query mechanism for finding information using *attributes*. The system can be extended through the use of user-defined transducers to customize data retrieval. While our work is basically different from the semantic file system, their research developed fundamental concepts of virtual file systems.

Presto [DELS99] 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. Thirdly, our system is targeted at organizing data for applications in addition to users. Lastly, we incorporate the mobility of users, allowing them to merge their data into a new space.

Gopal *et.al.* have extended the ideas of the semantic file system to include semantic directories to groups of related material [GM99]. Their system is able to accommodate multiple mounts with similar semantic meaning to aggregate files and directories. While their aggregation of material is similar to ours, we trigger grouping on context, allow file system primitives to be used to attach context to files in a natural way, and allow files and directories be created directly using the current context, which eliminates user intervention in determining data grouping semantics. In addition, we have introduced mobile mount points and dynamic data types, both considerations that we believe must be addressed in a file system for ubiquitous computing environments.

The Prospero File System [Neu92] constructs a virtual namespace based on *views* that are generated by different *filters*. Filters are attached to directory links and are applied to the link to allow the view of data to be altered. The system is able to group logically related material together in different ways to facilitate data organization. Data can reside on different servers and has been used to organize information on Internet sites. Their use of the virtual hierarchy to group data is similar to our implementation, but our approach considers other aspects that are related to ubiquitous computing environments.

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 [FIB95, IU97]. Attaching digital information to tangible objects can ease the way in which information is made available to a new environment. We expand on this idea by treating the *user* as the physical object that triggers the addition of information into a space, implicitly linking storage to a user. In

addition, physical objects, such as handheld devices, may actually contain the information, rather that just being linked to it.

Odyssey is a file system that has been developed to support mobile applications [NFS00, Nob00]. Odyssey defines a metric called *fidelity* that measures how closely the data a client application receives matches the reference data. Data may have varying degrees of fidelity, which corresponds to different levels of data degradation. The system is composed of *wardens* that encapsulate type-specific data knowledge and can also be queried to retrieve data attributes. This concept is similar to our dynamic data type attributes, in which a data source maintains a canonical format, but may be adapted to the needs of an application. Their system focuses on the varying degrees of network connectivity for mobile users and the speed at which applications can adapt to changing conditions. Our system assumes connected service and is more heavily focused on context and how to make information retrieval more simple for users.

The Iceberg project [RKJ99] uses the Ninja framework [GWBC99, GWvB+00] to provide an infrastructure for integrated network communications. One of the main focuses is support for user device modality. Users can switch between devices and the system can adapt to the properties of the device. The Automatic Path Creation Service provides the mechanism to transform content on-the-fly [Pro]. This services is similar to our mechanism that provides dynamic data types by providing impedance matching of data types to build paths to convert data formats.

7 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 integrate 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. Information becomes associated with the current task and the context may define the personal preferences or applications configurations. 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.

We have extended the concept of mounts points to include context tags to control the way in which data is organized in the directory hierarchy. Mounts are dynamic and can be injected into an environment to make personal data available. The mount mechanism uses the context tags to organize data so that data important to the current task is easy to locate, for a single user, group of users, or automated processes. The file system creates a virtual hierarchy where context is represented as directories and file system primitives are used to manipulate the virtual hierarchy to attach context to particular files and directories.

We have begun to experiment with the system in our *Gaia* infrastructure and have used it in several applications we have constructed. An important focus of future work will be to experiment further with the system and investigate what types of operations are most useful and how the system may be modified to meet user demands. We have developed an application framework that allows users to customize distributed applications based on the resources of a space. These application configurations are specific to a particular space and should only be visible in the context they are designated for. We will be exploring the best way in which to store these configurations

using context, thereby limiting the number of configurations visible to a user. We will also be investigating the way in which different applications can leverage the functionality of CFS. For example, the song-list for an audio application may change automatically as the context changes by simply opening the current context directory and reading the visible audio files. The functionality of importing the correct files is handled by the file system with no required changes to the application. Our system currently uses a fixed set of context types. While we believe this list to be useful and covers many of the requirements, we would like to make context types user defined. We will be reporting performance numbers in the future, but have found that the system works well for our prototype applications.

8 Acknowledgments

The authors would like to thank Manuel Román for numerous discussions regarding the design and application of our system within the *Gaia* environment, in addition to the implementation of the WindowsCE handheld application.

References

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

[DELS99] Paul Dourish, W. Keith Edwards, Anthony LaMarca, and Michael Salisbury. Presto: An Experimantal Architecture for Fluid Interactive Document Spaces. *ACM Transactions on Computer-Human Interaction*, 6(2), 1999.

[FIB95] George W. Fitzmaurice, Hiroshi Ishii, and William Buxton. Bricks: Laying the Foundations for Graspable User Interfaces. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'95)*, pages pp. 442–449, New York, 1995.

[GJSJ91] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole Jr. Semantic File Systems. In *Proceedings of the 13th Symposium on Operating System Principles*, pages 16–25, 1991.

[GM99] Burra Gopal and Udi Manber. Integrating Content-based Access Mechanisms with Hierarchical File Systems. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, New Orleans, LA, February 1999.

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

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

[HCM01] Christopher K. Hess, Roy H. Campbell, and M. Dennis Mickunas. The Role of Users and Devices in Ubiquitous Data Access. Technical Report UIUCDCS-R-2001-2226 UILU-ENG-2001-1733, University of Illinois at Urbana-Champaign, May 2001.

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

[HRC02] Christopher K. Hess, Manuel Romàn, and Roy H. Campbell. Building Applications for Ubiquitous Computing Environments. In *International Conference on Pervasive Computing*, Zurich, Switzerland, August 26-28 2002. (submitted).

[IU97] 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.

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

[MIT] MIT Media Lab. Smart Rooms. `http://ali.www.media.mit.edu/vismod/demos/smartroom`.

[Neu92] B. Clifford Neuman. The Prospero File System: A Global File System Based on the Virtual System Model. *Computing Systems*, 5(4):407–432, 1992.

[NFS00] Dushyanth Narayanan, Jason Flinn, and Mahadev Satyanarayanan. Using History to Improve Mobile Application Adaptation. In *3rd IEEE Workshop on Mobile Computing Systems and Applications*, San Diego, CA, December 2000.

[Nob00] Brian Noble. System Support for Mobile, Adaptive Applications. *IEEE Personal Communications*, 7(1):pp. 44–49, February 2000.

[Pro] Iceberg Project. Automatic Path Creation Service - APC. `http://iceberg.cs.berkeley.edu/release/APC.html`.

[RHC+02] 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.

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

[SAW94] 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.

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

[Wei93] Mark Weiser. Some Computer Science Issues in Ubiquitous Computing. *CACM*, 36(7):74–84, 1993.

[WO86] Brent B. Welsh and John K. Ousterhout. Prefix Tables: A Simple Mechanisms for Locating Files in a Distributed System. In *Proceedings of the Sixth International Conference on Distributed Computing Systems*, Cambridge, MA, 1986. IEEE Computer Society Press.