The Role of Users and Devices in Ubiquitous Data Access *

Christopher K. Hess Roy H. Campbell M. Dennis Mickunas

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

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

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

Report No. UIUCDCS-R-2001-2226, UILU-ENG-2001-1733

May, 2001

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

The Role of Users and Devices in Ubiquitous Data Access

Abstract

Ubiquitous computing has been identified as one of the next major directions in computing. This paper identifies several requirements to support data access in ubiquitous computing environments and presents a design to meet the requirements. The impetus for the design lies in three areas; 1) content adaptation, 2) data grouping, and 3) location awareness. The system is composed of data access building blocks, which allows devices to customize the way in which they retrieve data. All data in the system is indexed and typed, which facilitates data grouping, automatic content conversions, random access, and higher level services, such as streaming. The personal storage of a user is implicitly linked to a user and can be automatically incorporated into an environment conditioned on their physical presence.

Contents

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, Abo99, 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. This coordination may be viewed as being analogous to the role of a traditional operating system. However, the heterogeneity, mobility, and sheer number of devices makes the system vastly more complex [RC00]. Applications may have the choice of a number of input devices, such as mouse, pen, or finger; output devices, such as monitor, PDA screen, wall-mounted display, or speakers. An infrastructure for such a space must be able to locate the most appropriate device, detect when new devices are spontaneously added to the system, and adapt content when data formats are not compatible with output devices. We term these environments *active spaces*; physical spaces are activated by the entities and context information contained within them and become available to applications. Such spaces require an infrastructure to coordinate entities, assist weak devices, and facilitate network communication.

A key requirement in such environments is remote access to data. Next generation applications will not be able to make assumptions about the devices that users possess in order to present data [BBG+00]. Therefore, the computing infrastructure must be able to alter data to best suit the needs of applications, whether it be by transforming content or organizing data in a form that is more easily accessible. These environments must also consider the mobility and current location of users. By using the location of a user, an environment can incorporate any personal (remote) storage or storage that the user may be carrying, thereby automatically customizing the storage layout of the local environment. This storage becomes available to devices that are resident in the environment, as long as the user is physically present. Therefore, the types of devices that users possess and the location of the user become important parameters in the design of ubiquitous data access systems.

The remainder of this paper discusses the rationale and design of a system to support data access in active spaces and continues as follows. Section 2 explains the problems encountered when accessing data in the active space environment and Section 3 briefly describes the active space infrastructure we are currently developing. Section 4 describes the design of our data access system and Section 5 continues with some further issues that are considered in the design of our system. Section 6 reports the current status of our implementation. Finally, Section 7 gives a survey of related work and Section 8 ends the paper with some concluding remarks.

2 Motivation

In the active space environment, applications are no longer tied to specific machines, but may be divided among different networked components. These applications are not restricted to a single machine and its attached peripherals (e.g., keyboard, mouse, and monitor); they may use whatever devices are locally available, such as the ambient lighting or resident video projectors. In addition, these applications are not restricted to being visually displayed on a single monitor; they may choose from a menu of output devices or may be divided among any of the devices that are available.

2.1 Adaptation

Applications may no longer make assumptions about the types and characteristics of output devices that a user may possess or prefer. If a user wishes to use a device that does not support the original data format that the application provides, or prefers to receive some data in a different format, the infrastructure must make an attempt to present the data in the desired format. For example, if a user wishes to view an on-going presentation on a small handheld, images of the slides could be sent to the roaming user, but in a format more appropriate for the device, such as a scaled down image to fit the small screen size. Moreover, content type transformations may be performed, such as converting text data to audio in order to read news headlines. 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 specific desired type and the system should be responsible for automatically adapting content to the desired format. Data sources therefore become *dynamically typed*; data maintains a native format, but may be changed to different formats, based on application needs.

2.2 Grouping

Small devices may possess limited resources such as bandwidth and processing power. In such situations, retrieving data sources in their entirety may place unnecessary burden on these devices. To support these devices, data should not be represented as bytes streams, but as typed objects, which can then be selectively retrieved. In addition to changing data format, the type of an object can be changed to alter the way in which data is accessed, thereby allowing applications to more finely describe exactly the portions of data they desire. This facilitates *data grouping* in a manner that is most appropriate for a particular device. For example, suppose an application is constructed to view a text document. If the document is large, a more appropriate grouping would be pages, rather that the original text characters. The application can then retrieve only the exact page which the user desires, alleviating the device from determining what constitutes a page, how to find the desired page, and formatting it. By imposing structure on the data source, an application may customize the way in which it retrieves the data, with support from the infrastructure.

Content adaptation and data grouping can be achieved most easily by typing and indexing data. Typing data allows the system to convert content formats and indexing allows data to be grouped in different ways. The task of these operations is delegated to infrastructure components, thereby allowing the end devices to remain simple [HLBF01]. Indexed data forms the foundation of *data access building blocks*, which allows applications to customize the way in which they access data.

2.3 Location

A main component that distinguishes active spaces from traditional distributed systems is the awareness of user location. Typical systems have no knowledge of user location or provisions for its use. However, active spaces can exploit user location to trigger events to customize a physical environment for a particular user or affect an application [SDA99, DAS99]. The supporting infrastructure must be aware of user location, incorporate a user (based on preferences), and incorporate any devices that they may be carrying. The presence of new devices must be added to a space so that the infrastructure can access and use them.

Users should not be burdened with manually transferring files or data. The infrastructure should assist in making personal storage available automatically. Users should simply define rules

stating which storage should be made available to remote locations. Storage becomes implicitly linked to a user and can "follow" them around, becoming available whenever they enter a new space, alleviating the need to manually transfer files. Therefore, the physical location of the user triggers the automatic configuration of the user's environment.

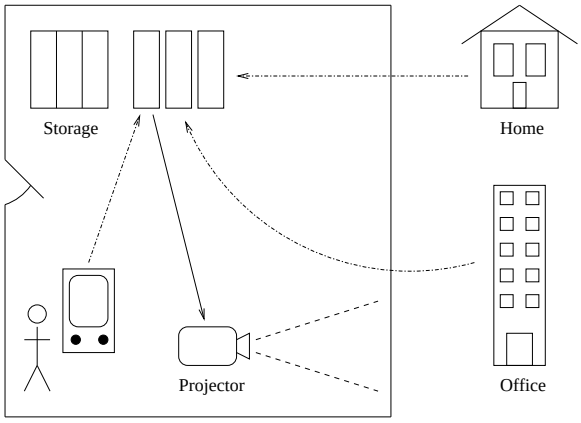

Figure 1: The presence of a new user in a space can trigger the addition of personal storage into the space. That storage then becomes available to local applications.

As illustrated in Figure 1, location information can be used to affect the available storage in an environment in two ways: 1) allow the remote storage of a user to be incorporated into an environment, and 2) allow mobile physically-local personal storage to be incorporated into a space. To illustrate the first case, consider a space that contains a projector and presentation software (i.e., a typical conference). When a person enters the space, they are discovered and the space is set up to incorporate their personal storage, which may be located remotely, at their home for instance. The presentation software can then browse the storage for the correct file which contains the presentation. However, this remote storage is only available when the person to whom it is associated is physically located in the same space as the projector and presentation software. Next consider the second case; suppose that a person has taken some pictures with a digital camera that they wish to print. Typically, this requires the user to transfer the image files to a PC and then initiate printing from there. However, if we consider the camera to be a storage device, the person could enter a space, triggering the new storage to become part of the space. Then the files could be printed directly from a printer, pulling the images from the camera, without manual transfer. The camera becomes a part of the space "file system". This has the added benefit of adding security by limiting access to data, since it can be accessed by the printer only when the camera and user are physically located in the same space. In both cases, location information has been exploited to simplify interaction among devices and storage.

To address the foregoing issues, this paper proposes a general data distribution service targeted at active spaces, that incorporates automatic content adaptation, customized data access, and location awareness. What follows is a brief description of our active space infrastructure currently under development, followed by the design of our data access service.

3 Gaia

Gaia is an active space infrastructure we are currently developing that exports and coordinates the resources contained in a physical space, thereby defining a generic computational environment [Gro]. *Gaia* converts physical spaces and the ubiquitous computing devices they contain into a programmable computing system. *Gaia* is analogous to traditional computing systems; just as a computer is viewed as one entity, composed of input/output devices, resources and peripherals, so is a physical space populated with many devices. An operating system for such a space must be able to coordinate the available resources. *Gaia* is similar to traditional operating systems by managing the tasks common to all applications built for physical spaces.

Gaia provides some core services, including events, entity presence (devices, users, and services), discovery, naming, location, and trading. Devices are able to detect when they have entered new spaces and can take advantage of the services available in the physical location. By specifying well-defined interfaces to devices and services, applications may be built in a generic way that are able to run in arbitrary spaces.

4 Architecture

The Data Object Service (*DOS*) is the data delivery mechanism for *Gaia*. *DOS* is a middleware data service that makes use of the native operating system to manage data on disk. However, the service offers more than simple access to file data. In general, data is no longer transported as streams of bytes (although this mode is supported), but as data objects.

The main abstraction for data is the concept of a *container*. In the most basic form, containers are simply wrappers for native file data or directories. In general, however, containers may represent *any* collection of data, that may be generated on-the-fly, gathered from disparate sources, or common data shared among distributed applications. Containers are constructed as distributed objects and applications can communicate with them via the *Gaia* infrastructure. Containers act as proxies [Sha86] by splitting parts of application logic among nodes. An example is a container translating MPEG to bitmaps for streaming video to a Palm Pilot device. The application can simply retrieve bitmaps objects from an MPEG data source and direct them to a display device, unaware of the complexity of establishing proxies and translating between data formats [HRCM00].

4.1 Containers

Containers are the main abstraction for representing data, which parse data sources into indexed components. They provide methods for creation and deletion of the data objects they hold. Concrete containers are implemented using the *Gaia* component model. Each container is built as a dynamic link library (on Windows) or a shared object (on Solaris). The component model allows the service to load, create, and activate container components. Decoupling the containers from the service allows new container types to be added to the running system without interrupting current applications. There are several different container types that perform different roles in the system.

4.1.1 File Containers

File containers enable access to native operating system files and directories. File containers parse data of different file types into indexed data objects. A file container is associated with a data type and the logic to deal with that type of data is encapsulated within it. The container metaphor

is used to separate concerns; the internals of the container organize a specific type of data into a standard format that can be manipulated in a generic way by applications.

However, some files do not contain any well-defined structure. Such files may be represented as a stream of bytes, thereby supporting traditional file semantics. These semantics can also be used by applications that want to bypass the type system of *DOS*, be it for backward compatibility or due to the lack of an appropriate container type.

4.1.2 Psuedo Containers

Pseudo containers are a special type of file container and represent data that is generated on-the-fly. Typically, data is retrieved from one or more sources and is aggregated to be viewed as a single source. From the application perspective, the source is viewed as a file, although the file does not actually exist. The file name is simply a placeholder to represent the data. The extension of the file informs the system of which container to instantiate to create the source data. For example, a file representing news headlines might be called "internet.news". However, the ".news" extension tells the system that the source type is news. The system uses this information to instantiate the container representing data sources of type news, i.e., a news container. The news container is then passed the name of the "file", internet.news. The container then uses the file name (minus the .news extension) as the subject, which it then uses to connect to a web site to gather the requested data. The data is then presented to the application in a form that is easy to manipulate. Although the data is gathered as a response to the application request, from the application point of view, the source is simply a file. The benefit of treating data sources in this manner is that applications can view data in a consistent manner. The way in which pseudo containers deal with names is type-specific.

4.1.3 Converter Containers

Weak devices may not be able to render data in its original format and may require format conversions [Wir]. Conversion of content is performed via a *converter container*, which is used to transcode data to a new format. Converter containers may be created on demand or automatically, when it is determined that the original data format is inappropriate, to provide on-the-fly transcodings.

For example, if a converter exists that transforms Microsoft *Word* documents into ASCII text format, an application could perform searches on *Word* files. The search engine would simply open the file as type "text", which the system would transparently convert in order to present the file in the format that the engine expects.

Complex conversions may require the support of several converter containers by chaining converters together. Converters can be created on different hosts, such as the local machine, the machine maintaining the native data, or any other machine. The mechanism for constructing container chains is described in Section 4.4.2.

4.1.4 Grouping Containers

Grouping containers are a special type of converter container that do not transcode formats, but rather change the type of data by grouping the original data in a different way. Grouping data can assist applications by allowing them to retrieve only the data they desire. This can decrease latency by providing a mechanism for applications to finely control which data is accessed and enhances random access capabilities. If only a small section of a data source is desired, a device

should be able to retrieve only the desired part, rather that retrieving the entire source and then finding the part it is interested in.

Consider a small device that wishes to view particular pages of a long text file. A container may be used to break the document up into pages. The original document may have a data type of "text". However, the type may be changed to "page", from which the device can access only the selected pages. This is similar to the concept of directories, which can be scanned to select a particular file. In the case of grouping containers, the sub-components are generated on-the-fly.

4.2 Resolver

The *Resolver* consists of a component that 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 data servers listed in the prefix table.[1] Each remote data server manages the data content on their respective machines and is responsible for creating container objects representing data on that host.

4.3 Layout Manager

The *Layout Manager* stores the prefix tables that allows machines and devices to export all or a portion of their storage. This manager is implemented as a network service and may provide private local storage for a group or a physical space. Typically, there is a manager running in each space. The Resolver queries the Layout Manger for the storage mappings specific to a space. These queries can selectively retrieve mappings based on a user name or a specific tag identifier, explained below.

The Layout Manager contains two types of storage mappings; static and dynamic. The static mappings refer to storage that is always available to a space. This may consist of local storage or shared remote storage. The dynamic portion can change as users move between spaces. When a user enters a new space and is detected, the user's personal space is dynamically added, making it locally available. Each user can specify a description of which storage they wish to export to other spaces.[2] By convention, this storage gets added to the */users/<username>* namespace. The system will automatically create the *<username>* directory under the */users* node. In this way, an application can always be sure that the storage of a user is located in a consistent place in the namespace. When the user leaves a space, the user's directory is automatically deleted from the space, which can restrict access unless the user is physically present, thereby enhancing security.

A further possibility is to export the storage on a mobile device (e.g., PDA or camera) as it enters a space. Consider a room that contains a projector and presentation software. The mobile device of the user may contain the presentation file. When the user enters the room, the device contacts the Layout Manager and informs it of which part of its storage it wishes to export and the room then adds this storage to its namespace. The user may then navigate with the presentation software, which resides in the room, to the directory containing the presentation of the user, residing on the mobile. This scenario requires a storage server to be running on the PDA. Such devices run a minimal server, while the remaining components (e.g., containers to manipulate content) run in the space infrastructure.

[1] Path prefixes, or "mount points", are translated to object references.

[2] The layout descriptions are stored at the user proxy, which holds personal preferences, which are automatically retrieved when a user enters a space.

The Layout Manager removes the need for users to manually transfer files that they will need when they move between spaces; the space automatically detects the existence of a new storage device and incorporates it. Hence, the namespace (i.e., what storage the room is aware of) can change dynamically as new users and devices enter and leave physical spaces. This allows a space to be customized by making a users personal storage available in any space that they are physically located. This can ease storage management and enhance security. This mechanism allows users constant access to their data, while remaining in control of it, i.e., the data can reside on their personal computer and is not required to be maintained by a central authority.

The Layout Manager is initialized with an XML configuration file. This file contains entries which specify which machines export a part of their storage, how it gets mapped to the users namespace, and to whom the description belongs. The description also contains a tag that can be used by applications as an identifier to a specific type of mapping entry. For example, the tag could specify what type of files are available in a certain directory.

Figure 2 shows two example layout descriptions. The first description is a static mapping that specifies that machine *srg181.cs.uiuc.edu* is exporting its *C:\Temp* directory and it will get mapped to */tmp*.[3] The second description contains two dynamic mappings that will get incorporated into a space when the user "ckhess" enters. The system will create a */users/ckhess/home* directory that will get mapped to the *C:\MyFiles\Export* directory on the host machine *machine.home.com* and a second entry that will map */users/ckhess/mp3* to *C:\MyMusic*. By using this convention, a user can always be sure that their personal storage is located in a consistent place regardless of the space they are currently in.

4.4 Container Manager

Access to each data source is initiated via a *Container Manager*. The Container Manager acts as a factory for container creation and is the main entry point to gaining access to container object references. Once a manager has successfully created an association between a container and a native file, pseudo, converter, or grouping container, a reference to the container is returned.

Container Managers also assist in data content adaptation/conversion, as described above, by finding the appropriate converter(s) and returning a new interface. Conversion is performed automatically by the manager when a request to open a container type does not match the underlying data source type.

Complex processing operations is done on a *just-in-time* basis, in contrast to *just-in-case* [PW97]. This lazy processing approach is taken to reduce the potential exponential growth of required processing, increasing latency only on first access or when data content changes.

4.4.1 Container Descriptions

In order for containers to be linked together to provide the proper conversion chain, a description of the containers must be available. Containers are described using XML. Each description specifies the name of the container component (i.e., the name of the library that must be loaded that contains the component), type of the container (file or converter), input data object type, output data object type, and an optional file type (expressed as a file extension) that the container is associated with. When a Container Manager first starts up (or when a new container type is added to the system), it reads the XML descriptions and creates a graph based on the input/output types. This graph

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

```
<DOS:Layout xmlns:DOS="http://choices.cs.uiuc.edu/gaia/DOS">

  <DOS:Storage>
    <DOS:User>system</DOS:User>
    <DOS:Tag>null</DOS:Tag>
    <DOS:Mount>/tmp</DOS:Mount>
    <DOS:Host>srg181.cs.uiuc.edu</DOS:Host>
    <DOS:Path>C:\Temp</DOS:Path>
  </DOS:Storage>

</DOS:Layout>



<DOS:Layout xmlns:DOS="http://choices.cs.uiuc.edu/gaia/DOS">

  <DOS:Storage>
    <DOS:User>ckhess</DOS:User>
    <DOS:Tag>null</DOS:Tag>
    <DOS:Mount>/home</DOS:Mount>
    <DOS:Host>machine.home.com</DOS:Host>
    <DOS:Path>C:\MyFiles\Export</DOS:Path>
  </DOS:Storage>

  <DOS:Storage>
    <DOS:User>ckhess</DOS:User>
    <DOS:Tag>mp3</DOS:Tag>
    <DOS:Mount>/mp3</DOS:Mount>
    <DOS:Host>machine.home.com</DOS:Host>
    <DOS:Path>C:\MyMusic</DOS:Path>
  </DOS:Storage>

</DOS:Layout>
```

Figure 2: Example of two storage descriptions, one for the system space (static) and one for a user (dynamic).

is used to determine which containers need to be instantiated and in what order to perform a particular conversion.

A description specifies which container is responsible for a particular data type and the logic to handle that type is encapsulated within the container implementation. Therefore, descriptions specify *what* container should handle a data type, and containers specify *how* that data is handled.

4.4.2 Container Chains

Using the container descriptions described above, containers can be chained together to provide multiple levels of content conversion. The system maintains an in-memory data structure of the current container descriptions to determine if a sequence of containers/converters may be linked together to provide a desired output type for a given data source. The data structure generated is shown in Figure 3. The structure consists of an interface map and a graph of data types, where edges between nodes identify the name of a library that performs the transformation. All nodes on the extreme right side of the graph are file extensions. All other nodes represent data types.

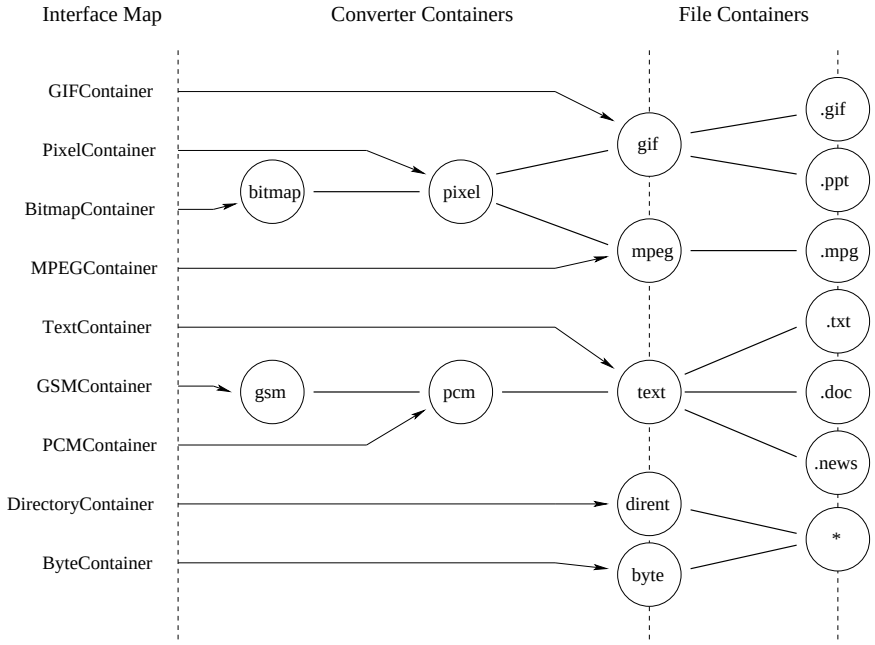

Figure 3: The system constructs a map/graph data structure that is used to determine how to convert content to a desired type.

When a request arrives to open a data source using a particular interface, first the desired output type is determined from interface map. Next, the graph is traversed to determine if there is a path to the input type or source.[4] If a path is found, the list of containers/converters necessary to perform the transformation is traversed, instantiating the components and linking them together. A handle to the container implementing the desired interface is then returned to the caller.

[4]The *createContainer* method is used to open a source and convert it to a desired type, while the *adaptInterface* method is used to convert a previously opened source to another type.

5 Further Considerations

The previous sections described the general architecture. The following sections describe some further issues that have been considered in our design to support additional features.

5.1 Attributes

Each container represents a particular type of data, which may have particular attributes associated with it. Some attributes are fixed, such as the frame rate of a video sequence. However, others can be set to affect the way in which data is manipulated. For example, a text container may have an attribute which specifies the language the text is in, or an image may have an attribute that allows the size of the image to be adjusted.

Attributes are specified as <*name, value*> tuples and are propagated along a container chain with requests for data. Any container in the chain that recognizes the attribute may use it to affect how it processes a request for data. If an attribute is not meaningful to a particular container, it is silently ignored and passed on to the next container in the chain.

One can view attributes as analogous to method calls, since they can change the state of a container. By adding new converters, the list of attributes that apply to the final data type can grow dynamically, in essence increasing the number of interface methods an object supports. Allowing attributes to act as method calls simplifies client development, since an application only requires a single handle to the chain and can affect the processing of any link in the chain by sending attribute tuples to it.

5.2 Push

Many sources of data are generated on-the-fly at an *a priori* unknown rate or should be received at a particular rate (i.e., 3 objects/sec). In such cases, it is more appropriate to push the data out to the client, rather than have the client explicitly request each object of data. Since data object containers encapsulate the knowledge of a particular data type, they can be queried for generic data objects, and pushed out to interested clients. This encapsulation allows the component that is responsible for pushing the data (i.e., the streaming component) to be fully generic, in that it does not need to know the peculiarities of the particular object type, but must only deal with generic data objects. The streaming component only needs to know the rate at which objects should be pushed and who to send them to. Separating concerns in this way allows the streaming component to be used with a variety of data types. For example, video/audio objects or data objects from an electronic whiteboard can be streamed to clients in exactly the same manner. There is no need to build a new streaming component whenever a new data type is introduced into the system; they can be immediately incorporated into the system. The streaming component includes methods to change the rate of the stream, to start and stop the stream, and to specify the data source to push.

5.3 End-to-End Issues

Any distributed system is subject to failure and should be resilient to any transient faults that may occur. The mechanism that the service employs to achieve fault-tolerance is the requirement that intermediate components remain stateless and state is only maintained on the endpoints [SRC84]. To achieve this goal, the current state of a data container is described as attributes, explained in Section 5.1. Any attributes that can affect the operation of a container (e.g., retrieving objects with a particular dimension if the container supports resizing) are sent along with each request.

Since the number of attributes associated with each data type is small, there is minimal overhead in sending the extra information in each request.

Data objects are typically transfered using a reliable protocol. When a component in the system fails, the underlying communication infrastructure can attempt to re-establish the connection, without intervention from the client application. Once the connection has been re-established, operation can continue as before. If a chain of containers is present for the particular data access, this operation may be recursively performed until the complete chain is rebuilt.

Providing end-to-end semantics also facilitates sharing of data containers. Since containers do not maintain per-client state, there is nothing in the container that is specific to the needs of a particular client. The entire requirements for accessing a data object is self-contained in each request. For example, if two devices are accessing a presentation, but one desires the dimensions of the slides to be different, the dimension attribute is added to the request and will be processed at the container for that request only.

6 Status

A prototype of much of the described system has been implemented, including the container chain graph constructor, container manager, layout manager, and resolver. A variety of container types have been implemented to determine the issues involved in constructing different container types that are able to be handled in a uniform way. The mechanisms to support location have also been incorporated. Currently, a user is able to specify descriptions of personal storage that are automatically mapped to a space, triggered by their physical presence.

Continuing work includes completion of our generic data push component and the implementation of a storage server for small devices that can be integrated into our system. We have experimented with running several types of servers on handheld computers to validate the feasibility of our approach.

7 Related Work

Our work resembles the functionality of a file system in traditional operating systems. Several existing systems have treated data as groups of data, rather than contiguous bytes of unstructured storage. The semantic file system [GJSJ91] indexes data when files and directories are created and updated to facilitate associative data access. They allow extraction of data using *transducers*. 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.

The *Choices* file system [Mad92] defines a framework for building different file system types. Data on secondary storage is represented as containers and is parsed and indexed depending on file type. In addition, container contents can be viewed in different ways. However, the system is not distributed and does not perform conversions.

The Symphony multimedia file system supports the retrieval of multiple data types [SGRV98]. The system is designed in two layers, consisting of a type independent layer and a type specific layer. The type independent layer deals with low-level block management. The type dependent layer is composed of type-specific modules that utilize a meta data manager to index data stores. Data can be accessed based on how modules define data boundaries within a file. Video and text modules were developed. In addition, the system supports both client pull and server push

models of data delivery. The use of type information for meta data generation is similar to our system. However, they concentrated on the low-level design, such as disk scheduling and block placement, whereas our system concentrates on content conversion and considers user location as a fundamental design parameter.

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 cooresponds 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. The system monitors current resource availability so that client applications can request changes in data fidelity. The speed at which applications can accommodate resource availability is defined as *agility*. Odyssey focuses on mobile applications, in which network connectivity can vary. While their system is similar to ours in that it encapsulates data type-specific logic into separate components, it does not consider data grouping or user location. Their system focuses on the varying degrees of network connectivity for mobile users and the speed at which applications can adapt to changing conditions.

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.

Work on digital libraries has identified some of the same issues that this work is attempting to address. Digital libraries contain vast amounts of data, often in different formats. The Digital Library Production Service at the University of Michigan has developed a system that can dynamically provide users with HTML content that is derived from various formats [PW97]. They identify the need for on-the-fly content conversion as essential to managing the vast amount of data. On-demand conversion ensures that data is only touched when needed, without burdening the system with maintaining data in all formats.

Other work in digital libraries has recommended that next generation file systems incorporate higher-level data operations [BC98]. They encourage the design of systems that federate dispersed data stores into single systems. In addition, the need for typed file systems is identified, in which untyped byte streams are replaced with data objects that are meaningful for applications. Our system incorporates many of these recommendations. Specifically, we treat data as typed objects and federate storage servers to be viewed in a single namespace, through our storage server mapping mechanism.

Several projects have investigated the problem of information access and sharing in heterogeneous environments. IBM's TSpaces enhances the concept of a Tuplespace by adding consideration for heterogeneity of devices, scalability, and persistence [WMLF98]. TSpaces allow distributed applications to share information in a decoupled manner and allows a high degree of interoperability, via tuples. Their implementation includes support for access control, event notification, and efficient retrieval of information. In addition, new operators may be dynamically added to the server, which may be used immediately. The TSpaces project resembles a database system and facilitates network communiction, where our system is more focused on the access of existing data sources and adaptation of content. However, we could create a container type that was specifically tailored for tuples, which could be used as a shared data among applications. The Interactive Workspaces

project at Stanford uses the TSpace for communication and to distribute events to decouple components in a shared workspace environment [FJHW00]. Similar to *Gaia*, they identify the need for an "operating system" for physical spaces. Their infrastructure handles device coordination and hides device and hardware heterogeneity.

The Infospheres project at Caltech is constructing an infrastructure for organizing task forces [Cha96]. Their goal is to build a system that allows highly dynamic groups to be rapidly assembled and share information. Other concerns are how to scale to billions of objects, restricting access to objects to authorized personnel, dealing with message delays over networks that may scale globally, and managing resources by "freezing" and "thawing" objects when needed. This research targets the organization of dynamic groups.

The Information Bus is a mechanism for distributing data in large systems [OPSS93]. The system defines communication protocols and self-describing objects that encapsulate operations of specific types of data. The design distinguishes between two types of objects; *service objects* and *data objects*. Service objects represent a data type and contain the functionality to gain access to the data. Data objects represent the data and may constitute data from some source. In addition, the bus defines two methods of operation. Data can be retrieved either using a request/reply or a publish/subscribe model. New service objects can be dynamically added to the system at run-time to support new application needs, without interrupting the system. The system exhibits some of the same concepts as our system, such as wrapping data sources as objects to adapt different sources to present a more generic interface to applications.

The work most similar to ours is that of the Ninja project from UC Berkeley [GWBC99, GWvB+00]. The Ninja architecture defines four main components: bases, units, active proxies, and paths. Bases are manifested as a cluster of workstations that provide scalability, fault tolerance, and concurrency. Units comprise the myriad of devices that may be connected to the infrastructure. The active proxies provide adaptation of content (similar to our containers), and are the result of previous research in data distillation using the TACC [Fox] model to perform on-the-fly data transformations [FGBA96]. Transcoding data formats was found to greatly increase the performance of certain applications [FGG+98]. The last component, paths, constructs flows of data that may be transformed while passing through different components, using their active proxies [CMI]. These are similar to our container chains. Our methodology is slightly different in that we index data to enable random access and data grouping, and we allow client applications to pull specific data objects and consider the location of the user.

The Iceberg project uses the Ninja framework to provide an infrastructure for integrated network communications [RKJ99]. The infrastructure supports call establishment and signaling. 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]. Iceberg is an infrastructure for messaging, in contrast to *Gaia*, which is developed as a general computational environment.

8 Conclusions

Ubiquitous computing is poised to revolutionize computer systems design and use. The disparity in device capabilities offers challenges in integrating these heterogeneous devices into such environments. Services must be available to each device, but it may be necessary to modify certain services if the connected device does not have the desired resources. Access to data is a key ingredient that must be considered, regardless of whether the data is locally resident, remote, or exists on a small

mobile handheld device.

This paper identifies some of the main requirements for accessing data in ubiquitous computing environments and proposes the design of a system to meet the requirements. The system incorporates data content adaptation, customized data access, and location awareness. The contribution is in modeling data in a way that enables the integration of various devices into such environments and considering the location of the user to trigger the mapping of personal storage into the local environment.

Next generation applications will not be able to make assumptions about the devices that users possess in order to present data. Therefore, the computing infrastructure must be able to alter data to best suit the needs of applications, whether it be by transforming content or organizing data in a form that is more easily accessible. The proposed system arranges data in such a way as to enable the formation of data building blocks, which allows customized access and assists in the development of higher level functionality.

A further consideration is the physical presence of the user. Storage becomes implicitly linked to the user and their location triggers the reconfiguration of storage in the local environment. In essence, personal storage can "follow" a user and becomes incorporated into a space when they are physically detected by the system. In addition, small mobile devices can be used as storage servers. This can be used to enhance security or to simplify the way in which data is accessed from special-purpose devices.

9 Acknowledgments

The authors would like to thank Francisco Ballesteros for many helpful discussions during the early stages of our design. We would also like to thank Manuel Román for numerous discussions regarding the design and application of our system within the *Gaia* environment.

References

[Abo99] G. D. Abowd. Classroom 2000: An experiment with the instrumentation of a living educational environment. *IBM Systems Journal*, 38(4), 1999.

[BBG+00] Guruduth Banavar, James Beck, Eugene Gluzberg, Jonathan Munson, Jeremy Sussman, and Deborra Zukowski. Challenges: An Application Model for Pervasive Computing. In *Proceedings of the Sixth ACM/IEEE Int. Conf. on Mobile Networking and Computing*, pages pp. 266–274, 2000.

[BC98] Mic Bowman and Bill Camargo. Digital Libraries: the Next Generation of File System Technology. *D-Lib Magazine*, February 1998.

[Cha96] K. Mani Chandy. Caltech Infosperes Project Overview: Information Infrastructures for Task Forces. `http://www.infospheres.caltech.edu`, November 1996.

[CMI] Sirish Chandrasekaran, Samuel Madden, and Mihut Ionescu. Ninja Paths: An Architecture for Composing Services Over Wide Area Networks. `http://ninja.cs.berkeley.edu/dist/papers/paths.ps.gz`.

[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.

[FGBA96] Armando Fox, Steven D. Gribble, Eric A. Brewer, and Elan Amir. Adapting to Network and Client Variation via On-Demand Dynamic Distillation. In *ASPLOS-VII*, Boston, MA, October 1996.

[FGG+98] Armando Fox, Ian Goldberg, Steven D. Gribble, David C. Lee, Anthony Polito, and Eric A. Brewer. Experience With Top Gun Wingman: A Proxy-Based Graphical Web Browser for the 3Com PalmPilot. In *IFIP International Conference on Distributed Systems Platformas and Open Distributed Processing*, Lake District, UK, September 1998.

[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.

[FJHW00] 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.

[Fox] Armando Fox. The Case for TACC: Scalable Servers for Transformation, Aggregation, Caching, and Customization. Qualifying Exam Proposal.

[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.

[Gro] Software Research Group. Gaia: Enabling Active Spaces. `http://choices.cs.uiuc.edu/gaia`.

[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.

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

[HLBF01] Andrew C. Huang, Benjamin C. Ling, John Barton, and Armando Fox. Applicance Data Services: Making Steps Towards an Appliance Computing World. In *CHI 2001 Workshop: Building the Ubiquitous Computing User Experience*, Seatle, WA, April 2001.

[HRCM00] Christopher K. Hess, David Raila, Roy H. Campbell, and Dennis Mickunas. Design and Performance of MPEG Streaming to Palmtop Computers. In *Multimedia Computing and Networking 2000 (MMCN00)*, San Jose, CA, January 25-27 2000. ACM.

[IU97] Hiroshi Ishii and Brygg Ullmer. Tangilble 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.

[Mad92] Peter William Madany. *An Object-Oriented Framework for File Systems*. PhD thesis, University of Illinois at Urbana-Champaign, June 1992.

[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`.

[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.

[OPSS93] Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen. The Information Bus: An Architecture for Extensible Distributed Systems. In *Proceedings of the 14th ACM Symposium on Operating System Principles (SOSP'93)*, pages pp. 58–68, Asheville, December 5-8 1993.

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

[PW97] John Price-Wilkin. Just-in-time Conversion, Just-in-case Collections: Effectively leveraging rich document formats for the WWW. *D-Lib Magazine*, May 1997.

[RC00] Manuel Roman and Roy H. Campbell. GAIA: Enabling Active Spaces. In *9th ACM SIGOPS European Workshop*, Kolding, Denmark, September 17-20 2000.

[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.

[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.

[SGRV98] Prashant J. Shenoy, Pawan Goyal, Sriram S. Rao, and Harrick M. Vin. Symphony: An Integrated Multimedia File System. In *ACM/SPIE Multimedia Computing and Networking 1998 (MMCN98)*, pages 124–138, January 1998.

[Sha86] Marc Shapiro. Structure and Encapsulation in Distributed Systems: the Proxy Principle. In *International Conference on Distributed Computing Systems (ICDCS'86)*, Cambridge, MA, May 19-23 1986.

[SRC84] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4), pp. 277-288 1984.

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

[Wir] Wireless Application Protocol Forum, Ltd. Wireless Application Protocol Architecture Specification. `http://www.wapforum.org`.

[WMLF98] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford. T Spaces. *IBM Systems Journal*, August 1998.

[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.