An Agent Framework for CE Devices to Support Storage Virtualization on Device Ensembles

Woojoong Lee, Young-Ki Hong, Chanik Park
Department of Computer Science and Engineering
Pohang University of Science and Technology, Pohang, Korea
Email: {wjlee, bird0303, cipark}@postech.ac.kr

***Abstract*— The problem of accessing and managing digital contents scattered on personal portable devices such as smart phone, digital camera, MP3 player, or laptop in an integrated fashion is critical in ubiquitous computing. It is not trivially solvable because the underlying networks are dynamically configured and the data types supported by each device are constrained. Due to recent rapid growth in personal digital contents, it becomes a major issue.**

To address this challenge, we have proposed a self-managed distributed file services called PosCFS+ [1], [2], [3] in personal area networks (PAN). PosCFS+ automatically detects every devices available in PAN and integrates the storage space of all the devices into a virtual storage. In order to facilitate data access to the virtual storage, PosCFS+ creates virtual directories which enable a semantic file addressing from applications. However, in a real world consisting of multiple consumer electronics (CE) devices, it is not easy or even impossible to install software components like PosCFS+ in the CE devices because they are proprietary.

In this paper, we present the design and implementation of a CE device agent which enables the storage virtualization by PosCFS+ on CE device ensembles. Most CE devices are connected to a host computer via USB interface, so the agent will be executed on the host computer to export the physical storage space of a CE device to PosCFS+. Then, virtual storages and virtual directories are created by PosCFS+ according to predefined user profile. Finally, to solve the problem of limited data type and format supported in a CE device, the event notification framework proposed in [3] is used for automatic data transcoding.

I. Introduction

Nowadays, as portable devices are becoming more available and more diverse, people carry several portable devices such as smart phones, laptops, MP3 players and digital cameras at the same time. In this environment, accessing and managing personal digital contents scattered on each device is really terrible business to users. Every devices has own specialized network or restricted I/O interface. For instance, a cell phone has a USB-type interface for exporting its storage space as a USB mass storage device and a Bluetooth interface for headsets while a smart phone is equipped with WLAN interface.

The heterogeneity and ad-hoc characteristics of the network defined as *Devices ensemble* [4] makes people harder to manage their digital contents. Moreover, personal digital contents are rapidly increasing with the recent advance of flash memory and small form factor hard disk technology.

In our previous work regarding PosCFS+ [1], [2], [3], we addressed the main functionalities required for file systems in ubiquitous computing and presented a new smart file service which could be adapted to the requirements with storage virtualization. The file service provides per-user global namespace (*virtual directories*) for managing and accessing data stored on a virtual storage constructed from physical storage spaces detached to device ensembles. The file service was implemented by peer-to-peer manner and using the UPnP protocol [5] to automatically build up a virtual storage space over all personal devices in the network, and also by using WebDAV [6] for file I/O.

Storage virtualization in PosCFS was represented by two main interfaces. One was an WebDAV-based *storage* interface for I/O access and the other was the *virtual directory* interface for semantic file addressing.

The virtual directory trees were dynamically created by matching the file metadata maintained by the file service with keyword-base queries on each client-side.

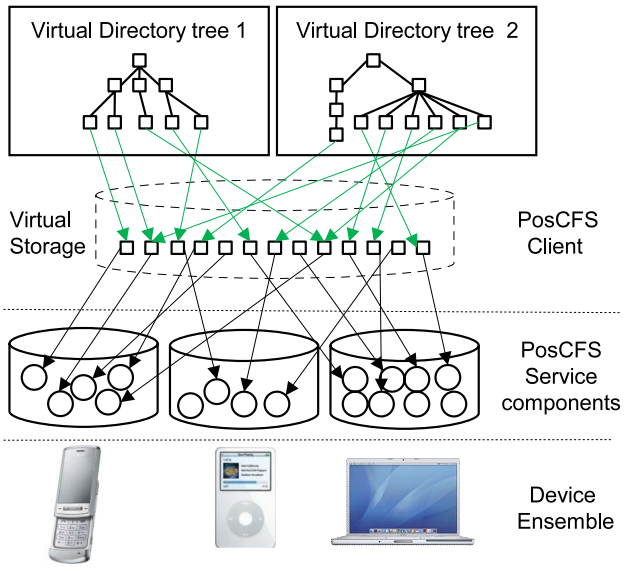

Fig. 1. A Conceptual View of Storage Virtualization on Device Ensembles: A virtual storage is constructed by peer-to-peer manner in personal area network and the virtual directory tree can be dynamically created on the client-side.

978-1-4244-2309-5/09/$25.00 ©2009 IEEE

In PosCFS+, we presented a *semantic file addressing* concept. All of the existing file systems have a namespace for file I/O, for instance, a directory structure represents file addresses on most systems. However, the structure is rigid and should be maintained by implicit assignment. For that reason, people tend to forgot where to store their files. This situation may become more serious on device ensemble or when they manages enormous amount of files. Especially, the explosion of personal data inspire the necessity of a new file addressing mechanism despite of emerging desktop search tools such as Google Desktop Search [7] and beagle[8].

People usually types in the search word in the search box with some keywords representing files that they want to search. This method is also applied to file systems and make them more comfortable for managing their files. For example, we can more easily access a file if we can use semantic metadata as representing file addressing like 'title=PosCFS & author=W.Lee & file-type=ppt' rather than the existing address scheme, '/home/wjlee/presentation-file/poscfs.ppt', because the former is more natural and user-friendly way than the latter when we try to accessing files. Especially, this scheme may be more useful if it can be apply to a distributed file system for integrating storage space on device ensembles.

However, in a real world consisting of multiple consumer electronics (CE) devices, it is not easy or even impossible to install software components like PosCFS+ in the CE devices because they are proprietary. Moreover, the data types supported by these CE devices are extremely limited comparing with laptops or desktop PCs. For instance, a Portable Network Graphics (PNG) format image may not be readable on a smart phone. Even if the phone supports the image format, the image resolution may lead to trouble.

In this paper, we present the design and implementation of a CE device agent for PosCFS+, which can be executed on general purpose computers, e.g. laptops, desktop PCs or PDAs with USB-host functionality, etc. Most CE devices can be attached to a host computer and be accessed as a removable USB mass storage device, so the agent exports the storage space of a CE device on behalf of the passive device. However, it is not simple to provide the virtual directory interface to the device as if the CE device is a normal PosCFS-enabled node because we cannot modify the default file browser or file system on the device. Furthermore, the operation of CE devices, such as taking a photo or playing a music file, may be disabled while it is connected to a host computer. Thus we use a prefetching technique with predefined user profile. Details of the technique will be described later. We also take advantage of the event notification framework proposed previously in [3] for automatic data transcoding in the prefetching process because most CE devices can only support limited data types as we mentioned before.

The remainder of this paper is organized as follows. Section II gives a brief outline of the PosCFS+ file service architecture. In section III, we present details of the CE device agent for PosCFS+. Experimental evaluations are given in Section IV. Comparison with other related works is described in Section V.

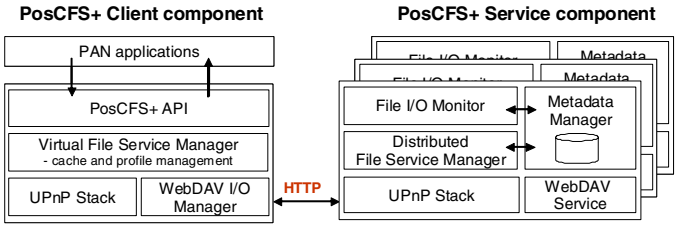

Fig. 2. PosCFS+ Architecture

Finally, Section VI presents our conclusion and future works.

II. Overall Architecture of PosCFS+

The PosCFS+ file service was implemented with two perfectly separate layers: the data access layer and the replication layer, and also provides Linux VFS support module based on FUSE [9]. For more details, please refer to [1], [2].

In this section, we present the overall architecture of the data access layer in PosCFS+ file service, which is shown in Figure 2. It is constructed using a peer-to-peer structure with UPnP and WebDAV protocols. The main role of PosCFS+ is to provide easy access to user data based on semantic metadata of files that are scattered in PAN. In our file service, it is supported by *storage virtualization*, which includes the concepts of *semantic file addressing* and the *virtual directory*. More details are discussed in the following subsections.

A. Conceptual View of Virtual Storage Space in PosCFS and WebDAV-Protocol-Based File I/O

PosCFS+ nodes use UPnP [5] protocol to discover and control one another in a peer-to-peer manner. The WebDAV [6] protocol is used for file I/O in the system. This protocol is an extended version of HTTP, which defines some extended methods for supporting file I/O on a traditional network file system, such as file writing, directory and file property management and locking, as well as the basic methods defined as HTTP, GET, and POST, which are methods for file reading. By using these global standards, we have been able to implement a platform-independent and self-constructible file service.

In PosCFS+, there are two conceptual space for data accessing: the *storage view* and the *virtual directory view*.(See Figure 3) The *store* interface, a connection point to virtual space in PAN, is dynamically and automatically constructed and managed by the UPnP protocol. The storage interface in the store provides an abstraction of WebDAV-based storage.

The *virtual directory* is a basic unit of semantic file addressing in our system. The virtual directory is dynamically constructed by matching the file metadata maintained by the file service with some conditions, such as user query, profile, and context information. For the construction mechanism, please refer to our previous works. However, we simply describes the addressing method based on user profile in the next section.

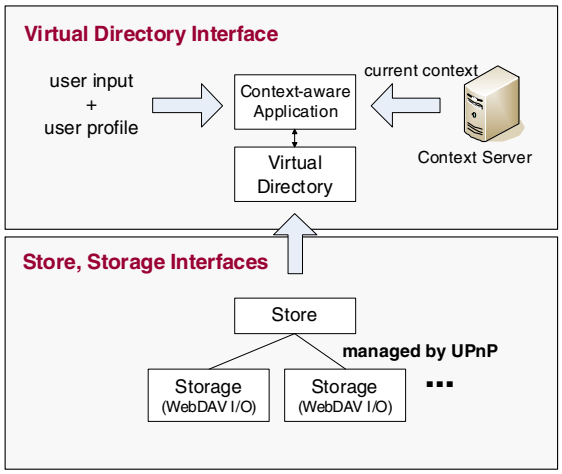

Fig. 3. Store, Storage and Virtual Directory interfaces

B. File addressing method based on user profile

In our file service, each client component has profile information, which consists of two parts. One is the *view preference* that defines *view-types* and *virtual directory construction rules*. As we mentioned before in section II-A, two types of view are provided, the *storage view* and the *virtual directory view*. The other part is a context DB configurations such as the *default DB location*, the *default DB* means a service component node that maintains user contexts, and information for access control.

Below, we briefly describe the virtual directory construction rules with some examples. The rules could be described with some pre-defined commands, such as 'DIR(NAME | NAMING-RULE) {CONDITIONS}' and 'SDIR(NAME | NAMING-RULE) {CONDITIONS}', and some specific keywords.

From the sequence of *DIR* and *SDIR* the categorization rules are included by some specific condition that correspond to the file metadata and context information maintained by the PosCFS service component. For instance, the case 'a)' in the example means, creating a virtual directory named 'docs' without sub-directories and with the condition, all the files in the directory should be document file class. Whereas the case 'b)' in the example means creating the virtual directory with sub-directories where each of them is named with distinctive author values.

The rules can be simply managed and converted into Structured Query Language in RDBMS with some other information such as the current context.

a) DIR(docs){file-class="document"};
b) DIR(docs){file-class="document"};SDIR(author=*){};
c) DIR(music){file-class="music"};SDIR(artist=*){};
d) DIR(wjlee){file-class="document" & author="wjlee"}
e) DIR(file-class=*){}; SDIR(location=*){};
f) DIR(current){ctx-name="project meeting"};
g) DIR(t-snapshot){}; SDIR(ctime=*){ctime $\leq$ 20070505 & ctime $\geq$ 20070501};

The virtual directory can be dynamically constructed by querying to the service side with simple-keyword based file queries generated from these rules, and the conditions and the naming rules are stored in the virtual directory interface.

III. CE DEVICE AGENT FOR POSCFS+

In this section, we introduce the design and implementation of the CE device agent for PosCFS+. The major functionalities of the agent framework is to provide a proxy service which makes closed-platform CE devices, such as digital cameras, MP3 players or smart phones, as if they are normal PosCFS enabled nodes.

As we mentioned before, the agent framework can be executed on general purpose computers, e.g. laptops, desktop PCs or PDAs with USB-host functionality, etc. Most CE devices can be attached to a host computer and be accessed as a removable USB mass storage device, so the agent exports the storage space of a CE device on behalf of the passive device.

However, it is not simple to provide the virtual directory interface to the device because we cannot modify the default file browser or file system on the device. Furthermore, the operation of CE devices, such as taking a photo or playing a music file, may be disabled while it is connected to a host computer. Thus we use a prefetching technique with predefined user profile.

The issues related to the automatic data transcoding are also important because the data types supported by these CE devices are extremely limited comparing with laptops or desktop PCs. For instance, a Portable Network Graphics (PNG) format image may not be readable on a smart phone. Even if the phone supports the image format, the image resolution may lead to trouble. Thus we take advantage of the event notification framework proposed previously in [3]. The details of this feature will be mentioned later, with introducing the data prefetching technique.

A. Design and Implementation of the CE device agent framework

The overall architecture of the agent is shown in Figure 4. The agent framework utilize the HAL[10] and the D-Bus[11] architecture on Linux for detecting CE device connections and disconnection. When it detected a CE device connection, the default *callouts* which are programs that run on device add/remove, are invoked by the HAL daemon and the event can be sent to the CE device manager in the agent framework by the D-Bus protocol. Then the device manager wakes up the *PosCFS+ Service Manager* that inspects the mounted storage space of the CE device whether a special information called *PosCFS service profile* exists or not. The *service profile* will be described in the next section. If the file is exist, the *PosCFS+ Service Manager* checks the profile and initializes a PosCFS+ service component for the device. Otherwise, the manager creates a new service profile from the service profile template and user inputs.

The roles of *PosCFS+ Client module* and *Third-party data converters* will be described in section III-C.

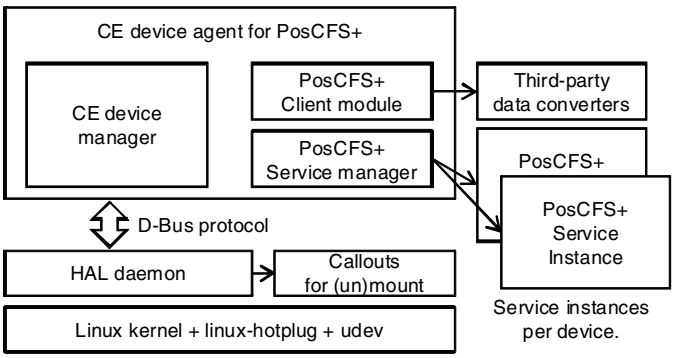

Fig. 4. The CE Device Agent Framework

B. User and Service Profile of PosCFS+

The user profile is used for constructing virtual directory tree as we mentioned in section II-B. Since CE devices cannot manage their virtual directory tree with the profile, the agent service handles it on behalf of the devices, i.e. it provides PosCFS client functionalities for the device. For more details, please refer to III-C.

The service profiles are mandatory information for the PosCFS+ service component, which are composed of two parts. One is the basic service profile that is defined and advertised by the SSDP/UPnP and the other is the extended service profile which is exported to other devices by an UPnP action defined in PosCFS+.

1) Basic Service Profile all the properties of the profile are defined in UPnP specification; *Device Type*, *Friendly Name*, *Manufacturer URL*, *Model Description*, *Model Name*, *Model Number*, *Model URL*, *Serial Number*, *Universal Device Name(UDN)*, *Universal Product Code(UPC)*

2) Extended Service Profile
- *Storage Size* : The storage size of the device. This field can be used to regulate prefetching operation.
- *Repository Path* : represents an export directory.
- *Supported data types* : mime type list delimited by ';'
- *Optional fields for data types* : optional fields, which can be used for describing details of the supported data types. e.g. resolution:320x240, etc.

C. Data Prefetching and Automatic Data Transcoding in PosCFS+

Most CE devices are closed platform, so we cannot modify or alter the default file browser application or file system on the devices. Furthermore, the operation of CE devices may be disabled while it is connected to a host computer.

In order to provide the virtual directory interface to a CE device, the CE device agent has to directly manage the storage spaces mounted on the host computer. It builds up a virtual directory tree by creating or deleting real directories on the mounted space. The rules for virtual directories can be generated from the *user profile* previously defined by a user. The *user profile* is stored to the CE device's storage space like the service profile. All the data which have to be in the virtual directories are prefetched into the CE device storage in order to make the embedded file browser in the CE device access the data after detaching the CE device.

Some examples of the user profile for the CE device are listed below.

- DIR(Music){mime-types="audio/mpeg"};
- DIR(Music){mime-types="audio/mpeg"};SDIR(Ella) {artist="Ella Fitzerald")};

This is a very useful functionality of PosCFS+. For instance, let us imagine a situation where an user wants to copy some music files of an artist from a home server to his MP3 player connected by his laptop. First of all, the user must mount each storage of the server and the MP3 player, and he have to find the files on the server. And finally, the user can copy them to the player. However, in PosCFS+, the series of process described above can be performed very simply and easily. The user only need to plug the player into an available USB port on the laptop and then to insert some conditions for creating a virtual directory tree.

The automatic data transcoding is one of the most important features because the data types supported by these CE devices are extremely limited comparing with laptops or desktop PCs. For instance, a Portable Network Graphics (PNG) format image may not be readable on a smart phone. Even if the phone supports the image format, the image resolution may lead to trouble. However, there are diverse data format, so the file system or service cannot handle all of them. Furthermore, some formats are only encoded by a special program.

Thus we take advantage of the event notification framework proposed previously in [3]. The event notification framework was designed and implemented for managing the virtual directory namespaces, so the all the file system events, especially file metadata changes, could be delivered to client-side.

Some third-party data converting applications such as *imagemagick*, *ogg2mp3* or *mp32ogg* can be used for data type converting process and invoked by the agent service using the framework. The agent service usually monitors the virtual storage space. When a file is prefetched into the CE device, the service detects the events and checks the information included in the *service profile* such as *Supported data types* and *Optional fields for data types*. If the file type is not described in the *Supported data types*, then the agent converts the data format to the desired data format using the third-party applications.

IV. Experimental Evaluation

In order to evaluate our proposed framework, we conduct several experiments especially on file I/O performance. For the purpose, the prototype of PosCFS+ has been developed in Linux 2.6.24. The testing platforms we have used include a Sony VAIO S55LP laptop with Pentium M, 1GB RAM, 802.11g, a Fujitsu P1510 mini-tablet with Pentium M, 512MB RAM, 802.11g and a Nikon D-80 digital camera.

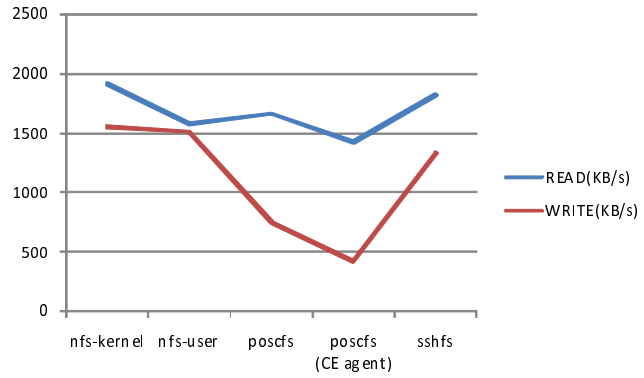

Fig. 5. File I/O Performance Comparisons with other network file systems, performed by IOzone Benchmark

The average initialization time of the proxy service invoked by the CE device agent was about a second, and the file I/O performance of the proxy service for CE devices was shown in Figure 5. The read performance of PosCFS+ was slightly better than that of the NFS user-server, and that of the proxy service was slightly lower than that of NFS user-server. However, the write performance of them were quite poor than that of other file systems because the service is currently implemented to support only synchronous I/Os. It must be improved significantly.

V. Related works

There are various distributed file system implementations for supporting device ensemble or for integrating storage space in PAN [12], [13], [14]. However, these systems cannot support the real CE devices because the devices are usually closed-platform.

EnsemBlue [15], proposed by the University of Michigan, is one of them that are specially designed for support closed-platform CEDs with a centralized file server with data synchronization by an event notification framework called *persistent query*. It also utilizes energy efficiency and file I/O performance. These features, inherited from BlueFS [16], were also developed by the same authors. However, it is not suitable for fully distributed environment due to its centralized feature, and it only provides a static global shared space, a global file tree, among devices which belong to users of the same group, such as a family or an organization.

VI. Conclusion

In PosCFS+, all the storages in PAN could be integrated automatically into a virtual storage by the file service and the digital contents in the storage could be accessed by the virtual directory interfaces. However, in a real world consisting of multiple consumer electronics (CE) devices, it is not easy or even impossible to install software components like PosCFS+ in the CE devices because they are proprietary. Moreover, the data types supported by CE devices are extremely limited comparing with laptops or desktop PCs.

In this paper, we present the design and implementation of a CE device agent for PosCFS+, which can be executed on general purpose computers, e.g. laptops, desktop PCs or PDAs with USB-host functionality, etc. Most CE devices can be attached to a host computer and be accessed as a removable USB mass storage device, so the agent exports the storage space of a CE device on behalf of the device. However, it is not simple to provide the virtual directory interface to the device as if the CE device is a normal PosCFS enabled node because we cannot modify the default file browser or file system on the device. Furthermore, the operation of CE devices, such as taking a photo or playing a music file, may be disabled while it is connected to a host computer. Thus we use a prefetching technique with predefined user profile. We also take advantage of the event notification framework proposed previously in [3] for automatic data transcoding in the prefetching process because most CE devices can only support limited data types. In order to show the applicability of the file system, some user scenarios were presented.

Acknowledgment

This work was supported by the IT R&D program of MKE/IITA [2008-S034-01, Development of Collaborative Virtual Machine Technology for SoD (System on-Demand) Service]

References

[1] Woojoong Lee, Shine Kim, Jonghwa Shin, and Chanik Park, “PosCFS: An Advanced File Management Technique for the Wearable Computing Environment”, LNCS 4096 - Proc. EUC’06, IFIP, 2006, pp. 965-975.
[2] Woojoong Lee, Shine Kim, and Chanik Park, “PosCFS+: A Self-Managed File Service in Personal Area Network”, ETRI Journal, vol.29, no.3, 2007, pp. 281-291
[3] Woojoong Lee, Chanik Park, ”An Event Notification Framework for PosCFS+ Distributed File Service in Personal Area Network”, 2nd Int’l Conf. on Next-Generation Computing, 2007.
[4] B. N. Schilit, and U. Sengupta, “Device ensembles”, Computer 37, 12, pp 56-64, 2004
[5] UPnP Forum, “UPnP: Universal Plug-and-Play”, `http://www.upnp.org`
[6] IETF, “WebDAV: Web-Based Distributed Authoring and Versioning,” RFC 2518.
[7] Google Desktop Search, `http://desktop.google.com`
[8] Beagle, `http://beagle-project.org`
[9] Filesystem in Userspace, `http://fuse.sourceforge.net`
[10] HAL 0.5.10 Specification, `http://people.freedesktop.org/~david/hal-spec/hal-spec.html`
[11] D-Bus Specification, `http://dbus.freedesktop.org/doc/dbus-specification.html`
[12] C.K. Hess and R.H. Campbell, “A Context-Aware Data Management System for Ubiquitous Computing Applications”, Proc. Int’l Conf. Distributed Computing Systems, 2003.
[13] A. Karypidis and S. Lalis, “OmniStore : A System for Ubiquitous Personal Storage Management”, Proc. Fourth Annual IEEE Int’l Conf. Pervasive Computing and Communications (PERCOM’06), 2006.
[14] Brandon Salmon, Steven W. Schlosser, Lily B. Mummert, Gregory R. Ganger, “Putting Home Storage Management into Perspective”, Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-06-110, September 2006.
[15] D. Peek and J. Flinn, “EnsemBlue: Integrating Distributed Storage and Consumer Electronics,” 7th Symp. Operating Systems Design and Implementation (OSDI), 2006.
[16] E.B. Nightingale and J. Flinn, “Energy-Efficiency and Storage Flexibility in the Blue File System”, 6th Symp. Operating Systems Design and Implementation (OSDI), 2004.