FEW Phone File System [⋆]

João Soares and Nuno Preguiça

CITI / DI, FCT, Universidade Nova de Lisboa

Abstract. Mobile phones have evolved from simple communication devices to powerful computing devices with large storage capacity and support for Bluetooth communications. In this paper we present a system that allows users to transport files in their mobile phones. Our system allows users to access these files in any nearby computer with Bluetooth support, as any other local file. For providing acceptable performance, our system automatically replicates files as necessary.

1 Introduction

In recent years, mobile phones have evolved from simple communication devices to powerful computing devices with built-in digital cameras, large storage capacity and multiple wireless communication capabilities (including not only GSM/3G but also Bluetooth and WiFi). These devices are bundled with complex software, such as games and multimedia players. Additionally, there is a widespread support for running Java applications.

In the context of the Files EveryWhere (FEW) project, we are building a distributed file system that allows users to explore the multiple available storage devices to safely store and share their data. In this paper, we present a simplified version of our system that allows a single user to store files in a mobile phone and access these files from any nearby computer with a Bluetooth networking card.

1.1 Related work

Either integrated in the operating system or relying on mobile-phone provided software, there are tools that allows users in a computer to browse and copy file from/to the mobile phone storage. Unlike these tools, our system allows a user to access files stored in the mobile phone as normal files, without the need for explicitly copying them from/to the mobile phone.

In recent years, several storage systems have been developed for mobile computing environments (e.g. [4, 5, 2, 3]). From these systems, Footlose [2] is the one that is most closely related with ours. However, unlike Footloose, our system allows mobile phones to store normal files that can be used by legacy applications in the same way as they use files stored in the computer's file system.

[⋆] This work was partially supported by FCT/MCES – POSC/FEDER #59064/2004.

2 System model

The FEW PhoneFS system allows to mount on the file system of the users' computer, containers (sets of files) [1] stored on the users' mobile phone file system. To this end, the software component (client) on the computer intercepts file system calls and communicates with the server on the mobile phone using Bluetooth communications. While the file system is mounted, the mobile phone is able to make and receive phone calls or execute any other operation the user wants to do.

2.1 Server

The server component of our system runs on the mobile phone and it creates a new service that can be accessed over Bluetooth. In this way it is possible for nearby devices to discover and connect to this service, creating a new communication channel between both devices.

The server accepts incoming connections from other devices, allowing clients to execute operations on the server. The available operation are the common file system operations, including directory listing, read and write operations on individual files.

2.2 Client

The client component of FEW PhoneFS runs on the user's computer and intersects and handles the file system calls executed in a specific directory. The client's main task is to communicate with the server to receive data contents (on read operations) or to send data (on write operations).

On startup or on user's request, the client will search for nearby devices and allow the user to select the device with which it will establish a new communication channel. After this, the client uses the established channel to provide the user access to the files stored in the mobile phone.

3 Implementation

3.1 Software requirements

The system is based on the client-server paradigm. The server, which runs on the mobile phone, uses the Java platform for mobile devices (J2ME); the client runs on a PC and it is implemented in Java and uses the FUSE system for file system calls interception. The mobile phone's J2ME platform must include the JSR-75 API. This is an optional API that gives access to the mobile phone's file system including removable storage media.

For using Bluetooth, both platforms must include the JSR-82 Bluetooth API (a.k.a. JABWT). The functionality provided by this API includes three main types of services:

1. **Discovery** - Providing device and service discovery and service registration.
2. **Communication** - Providing reliable connections between devices.
3. **Device Management** - Allowing to manage and control connections, local and remote devices state and properties.

Bluetooth-enabled application can be divided into two groups: servers, which create and publish new services; and clients that consume remote services. Clients can discover mobile phone that provide a given service by relying on the JSR-82 discovery service. The protocol used for communication channels is the RF-COMM protocol, which emulates an RS-232 serial port enabling the creation of reliable stream oriented communication channels.

For our prototype, in the PC we are using a JSR-82 implementation by Avetana. Regarding the mobile phones (Nokia E-70), they already included the JSR-82 and JSR-75 API.

3.2 Server

The server, being executed on the mobile phone, starts by creating a new service based on the RFCOMM protocol over Bluetooth (btspp) and publishing it. Thus, this service can be discovered by nearby devices. When a remote device attempts to establish a connection, the server opens a new bidirectional communication channel with that device and starts waiting for requests.

After this point, the protocol for service request is very close to the interaction established with an web server. The requests from the client will include an operation code followed by the URL denoting the file in which the operation is to be made. The server executes the requested operation and returns a response code to the client, denoting the success of the operation (or not), followed by its result. For example, for a read operation on a file, the result will include the file contents.

3.3 Client

The client consists of a user-space application that receives file system calls intercepted by the FUSE system and handles it appropriately by communicating with the server (mobile phone) to execute the requested operation. To this end, it uses the JSR-82 API to discover devices that implement our service and to establish communications with the server.

4 Cache management

The biggest drawback of Bluetooth communication is its relatively low transfer rates. To minimize the impact of this characteristic, the client component includes a cache system that tries to minimize data transfers between the mobile phone and the PC. The cache will eventually store, on disk, all files accessed by the user on the PC side. This way, when a user accesses a file that is in the cache, minimal communication with the mobile device is required (as explained later). When a user reads a file that is not present in the cache, the client caches the file by requesting the file contents from the server (phone). When a user writes a file, initially only the cached version is modified. Only when the file is closed (released), the updates are propagated to the server.

To guarantee that the user does not access an old version of a cached file, the client requests the last modified date of a file to the server before deciding to execute the operation on the cached replica. If the date returned by the mobile phone is older or equal to the date of the local replica, the cached replica if valid and it can be used for executing operations. Otherwise, it must be invalidated and a new replica needs to be fetched.

4.1 Implementation of the cache

Besides the cache for file contents, that works as described before, we implement a cache for the file system structure. This is important because for each directory or file that is accessed by the user, the file system requests the attributes of all individual directories in the given path. Thus, to reduce the required communications with the mobile phone, we start by creating a cache of the attributes for all files and directories in a container. This cache is partially invalidated when files are modified (or deleted).

References

1. M. Bento and N. Preguiça. Reconciliation for mobile computing environments with portable storage devices. In *Actas da Conferência sobre Sistemas Móveis e Ubíquos (CSMU-2006)*, 2006.
2. J. M. Paluska, D. Saff, T. Yeh, and K. Chen. Footloose: A case for physical eventual consistency and selective conflict resolution. In *Proceedings of the 5th IEEE Workshop on Mobile Computing Systems and Applications*. IEEE Computer Society, Oct. 2003.
3. D. Peek and J. Flinn. Ensemblue: Integrating distributed storage and consumer electronics. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, November 2006.
4. S. Sobti, N. Garg, F. Zheng, J. Lai, Y. Shao, C. Zhang, E. Ziskind, A. Krishnamurthy, and R. Wang. Segank: A distributed mobile storage system. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST '04)*. USENIX Association, Mar. 2004.
5. N. Tolia, J. Harkes, M. Kozuch, and M. Satyanarayanan. Integrating portable and distributed storage. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST '04)*. USENIX Association, Mar. 2004.