OTA MOBILE DEVICE SOFTWARE MANAGEMENT

Sudharshana M, sudharshana@motorola.com, Parixit Aghera, parixit@motorola.com,
Suresh Chintada, Suresh.Chintada@motorola.com,
Motorola India Electronics Ltd., 33/A, Ulsoor Road, Bangalore, India – 560 042
Alan Bok, Alan.Bok@motorola.com, John Grosspietsch, John.Grosspietsch@motorola.com,
Motorola Labs, 1301, East Algonquin Road, Schaumburg, IL 60196

ABSTRACT

There is a tremendous proliferation of Mobile devices, fueled by intense competition among various players to provide consumers with better and better ways to communicate and as the technology evolved from Analog, to Digital and now 3G. The software content on these mobile devices continues to increase as the underlying processor and DSP technology matures. The increased software content on these devices comes fraught with the risk of containing buggy code leading to disruption of service to the consumer. Also as increased competition puts pressure on the service providers to differentiate their offerings, the number and variety of features that get added to these devices also is increasing steeply. Both these situations require managing the software content on these mobile devices in an innovative way if one wants to contain costs and sustain revenues.

In this paper, we propose a new architecture for over the air management of software on a mobile device. A software architecture supporting software patches, including secure downloading of software from a data network and robust installation of the same on a mobile device based on Sun J2ME platform and SyncML framework is discussed in this paper. Using this architecture, a network operator can notify a mobile device user about the software upgrade and send the upgrade to the mobile device over the air. The paper uses our current prototype implementation and discusses the implementation of the proposed architecture. The current prototype implementation demonstrates remote management of DSP software on mobile phones in GSM or GPRS networks using an efficient installation algorithm with error recovery mechanism. It uses digital signature for checking authenticity and integrity of the downloaded DSP software patch.

1. INTRODUCTION

Research on software-defined radio (SDR) has become the latest buzzword with in the Mobile device manufacturers and operators community. Software defined radio technical implementation issues vary according to the form factor and the applications. Recent developments in SDR technology have been discussed in [1]. A number of technologies have allowed the development of commercially viable platforms providing an efficient bridge to the third generation wireless systems as discussed in [2] and [3]. Though the core issue lies in hardware support for SDR phones [4], an important issue lies in customizing the software for different needs of the network and the consumers. Some issues pertaining to maintaining the protocol software stack software have been addressed in [5]. In this paper we identify SDR is not restricted to just maintaining network related software, but also maintaining mobile device specific software and attempt to show this device software could be managed by using over the air mechanisms.

Over the air Mobile Device Management is receiving enormous amount of interest in the mobile device community. This is due to the fact that, number of mobile device users has been exploding and the size and complexity of software content on each of these devices is also increasing, leading to difficulty in managing all these devices. A number of issues related to managing the software on these devices using over the air service provisioning, re-programming, quality of service and other similar problems are being addressed and discussed extensively. Mobile device software management includes operations such as application provisioning, application management and software upgrades. Software upgrades could be used for adding a new functionality, enhancing an existing feature, or bug fix to the existing phone software. As of today mobile device software management is done offline at a customer care center or at the factory. The disadvantage with this approach is that user has to personally visit the customer care center and surrender the mobile device for maintenance. This results in unavailability of the mobile device to the user for that particular period as well as increased costs of maintenance to the mobile device manufacturer as well as to the operator. As the mobile device user population increases, this task of offline-management becomes tedious and expensive.

In this paper we introduce the problem of device management and cover related issues in Section 2. A detailed discussion on our solution and use cases is discussed in Sections 3 and 4. Finally, in Section 5 we conclude by summarizing the advantages of our solution and future work.

2. ISSUES IN OTA DEVICE SOFTWARE UPGRADES

Figure 1 is a simplified diagram showing elements involved in OTA (Over The Air) software management and describing the problem of software management.

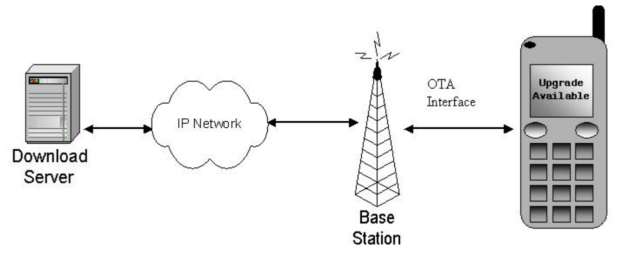

Figure 1: Elements involved in OTA mobile software upgrade

Following are the issues that any OTA device software management architecture must address:

2.1. Interoperability

An operator's network provides services to MEs (Mobile Equipment) from multiple vendors and a vendor's ME accesses services from different operator's network. So interoperability between an operator network and the device manufacturer is the basic requirement for device software upgrade.

2.2. Variations in ME architectures

Different ME vendors manufacture MEs with various software/hardware architectures. Hence the process of storing and installing the software upgrade or a software patch is specific to ME platform. These variations in the ME hardware and software architectures need to be understood for captured in the architecture for a solution generic enough to suit different devices.

2.3. Extensibility

Extensibility implies, that it should be easy to add software upgrade support in the system for a new ME or for a different hardware platform software of an existing ME.

3. PROPOSED ARCHITECTURE

In this section we propose a software architecture that addresses the issues discussed in Section 2.

3.1. Architecture

OTA Software Upgrade can be divided into 3 major functionalities.

Notification and Download Protocol for patch: This functionality is proposed to be implemented using common industry standards to make software upgrade solution interoperable. We have used SyncML [6] for this functionality. SyncML is a new industry initiative to develop and promote a single, common data synchronization/ device management protocol that can be used industry-wide. To maximize the reuse, we have implemented SyncML specifications in J2SE [8] at server side and J2ME [9] at client side. More specifically we have used MIDP (Mobile Information Device Profile) [10] at client side, since MIDP is most widely available implementation for J2ME based devices.

Storage and Installation of patch on ME: Though storage and installation of a patch is platform dependant functionality, its necessary to have a defined interface for this functionality. Implementation of this functionality can vary but the interface should remain the same. We have used J2ME at client side for defining Java APIs for this functionality.

Generation and Storage of patch on download server: A generic structure for the patch is not defined as the generation of the patch is ME specific. Patch and its license are generated by the ME manufacturer. These are in turn distributed to operators for hosting on a software management server.

Architecture proposed in **Figure 2** as a solution for OTA Mobile Device Software Management adheres to SyncML Device Management specification. Server side components are collectively known as Management Server and Client components are collectively known as Management. Both client and server side components fall into 3 categories.

- User Interface (client / server application)
- Device Management Logic (SyncML Profile)
- Data Repository Management (TM [Terminal Management] Profile, Patch Profile, Server Profile, Flash Memory/Management Information Base)

3.2. Server Application

Server Application provides web-based Graphical User interface (GUI) to operator for setting a management operation for a specific device. For example operator can select particular software version of an application to be distributed to particular type of devices. Server application initiates the software upgrade or another management operation by sending SMS notification message to terminal. In J2SE environment a server application is Java servlet but it can be implemented in other HTTP server program environment like CGI, ASP etc…

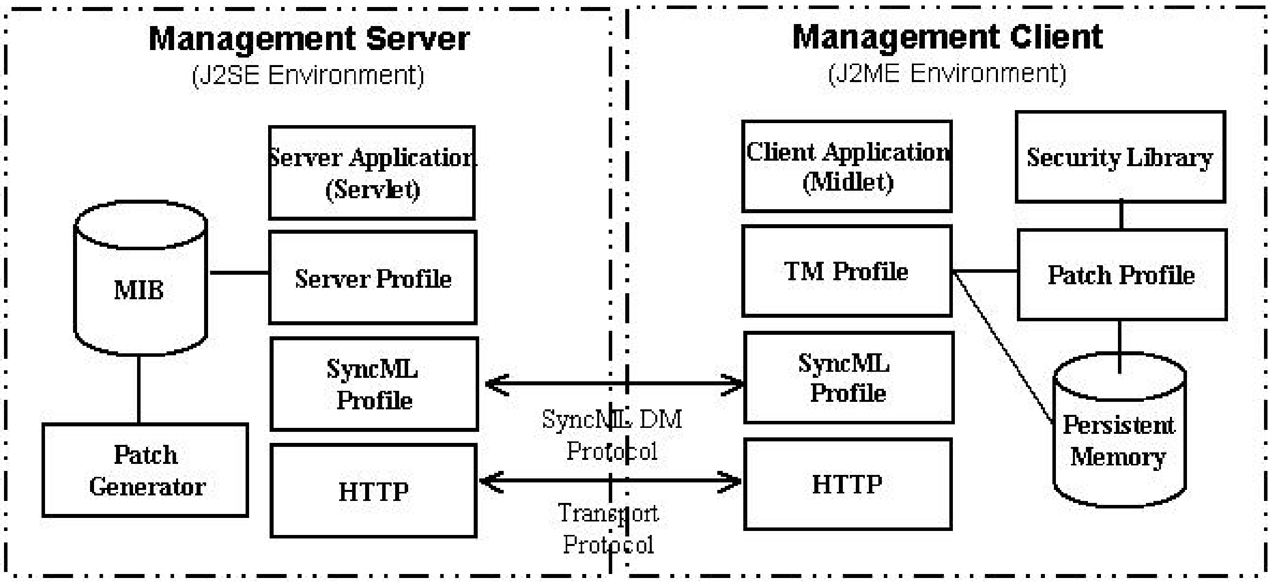

Figure 2: OTA Mobile Device Software Management Architecture

3.3. Server Profile

Server profile provides access to appropriate data stored in a MIB during a SyncML management session with terminal.

3.4. SyncML Profile

SyncML Profile implements SyncML Management Protocol, version 0.9 as specified in [11]. SyncML profile implementation of client and server talks according to SyncML DM (Device Management) protocol. This component generates SyncML messages to be sent to client and parses the received SyncML messages.

3.5. Client Application

Client Application is a MIDlet application which will present User Interface to mobile device user and accept user input whenever it is required. Client application is required to initialize SyncML profile for exchanging SyncML packages with Management Server. It also initializes TM profile for executing the device management command specified by the Management Server.

3.6. HTTP

HTTP transport protocol is used as transport protocol in proposed solution. But it is possible to use a different transport protocol for completing the communication between client and server.

3.7. TM Profile

TM profile executes the command specified by the management server. Some of the command execution may require invoking native application or methods. This is accomplished by using JNI (Java Native Interface). It uses Patch Profile Services for software patch storage and software patch installation services.

3.8. Patch Profile

The Patch Profile provides following services required for a software upgrade.

Version and Resource Information: Patch Profile maintains version information of all upgradeable software applications on the ME. It also keeps track of available resources for download and installation of a new patch.

This version and resource information is sent to server during SyncML message exchange so that Server will be able to decide whether ME requires upgrade for an application.

Secure Patch Storage: Patch Profile stores downloaded software patches in the flash memory of ME. Patch Profile manages storage of multiple patches. It verifies the authenticity and integrity of the patch by using patch license downloaded along with patch. If confidentiality of patch is important for an ME manufacturer, patch can be encrypted and then the encryption information can be provided in license.

Fail-safe Installation: Patch Profile ensures that patch installation proceeds automatically and is managed in a fail-safe manner that is supposed to prevent partial or corrupt installation from crashing the system or otherwise rendering it inoperable. Since software upgrade could be for system critical component any corruption during software upgrade may render the device unusable. Once installation is started it ensures that installation is completed.

Patch Profile is designed in generic way so that it can be extended easily for different software patch types.

3.9. MIB

The Management Server maintains MIB or Management Information Base. MIB contains information for each terminal that Management Server manages. It also stores different software patches that ME needs to download.

3.10. Flash Memory

Flash memory is re-programmable persistent memory in a ME. All the ME configuration parameters and upgradeable application software are stored in this memory. TM profile and Patch Profile re-programs the flash memory to set new configuration parameters and upgrade the device software.

3.11. Application Patch Generator

Patch and its corresponding license for software of an ME are released by the application Patch Generator component. Both patch and license are stored in MIB along with the their description, software version, ME model ID for which the patch is, resource requirement for installation of the patch on ME. Application Patch Generator assigns a unique ID to each new patch generated.

3.12. Security Library

Before installing a patch, it is important to verify the authenticity and integrity of the patch data. Patch Profile uses security library available on the platform for this purpose. Patch Profiles provides a Java wrapper around this security library. Patch Profile verifies the integrity and authenticity of downloaded patch by passing patch and patch license to the security library. Support for confidentiality in Security Library is optional to ME manufacturer.

3.13. Patch Download Process

Patch Download Process complies with SyncML Device Management protocol as described in [12]. There are 2 phases as in a management session between Management Server and Management Client.

Setup Phase: Setup phase requires exchange of package0 to patckage2 between Management Server and Management Client. Pakcage0 is required only in server initiated management session. Client initiated management session starts with package1.

Management Phase: Management phase consists of a number of protocol iteration. Protocol iteration means a package from client to server and a package from server to client.

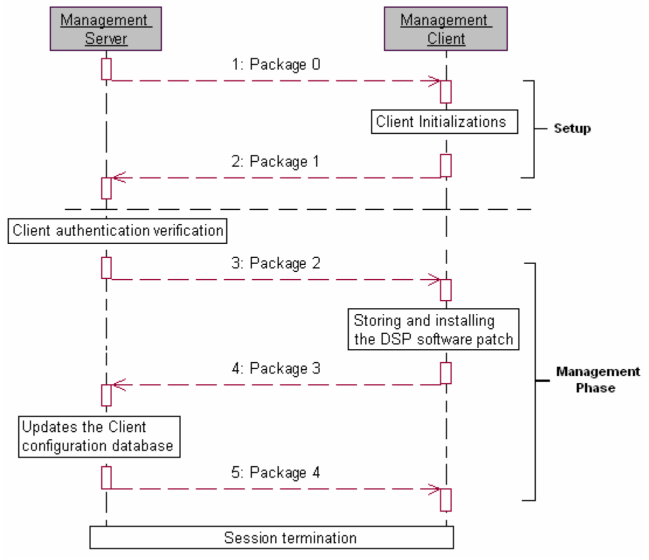

Figure 3: SyncML package exchange for DSP software upgrade.

Figure 3 shows the example SyncML package exchange between Management Server and Management Client for DSP software upgrade.

Package 0: "Package 0" is optional. Whenever a new DSP patch is available for a terminal, Management Server invokes on the terminal by sending a SMS auto-launch message via SMS-C. SMS auto-launch is parsed by SMS engine, which in turn launches the Client Application.

Package 1: "Package 1" contains client authentication credentials and device information including DSP software version.

Package 2: Terminal Management Server sends following information in a SyncML document.

- Patch Information (Description of the patch)
- Patch
- Patch License

Package 3: Management Client stores the DSP Patch and performs DSP patch installation DSP patch installation. Once the installation is complete it sends back the results back to Management server for operation in package3.

Package 4: Signals client to terminate the management session.

Large sized patch can span across multiple SyncML packages. SyncML Sync Protocol version 1.1 specifies how to handle large objects.

4. PROTOTYPE IMPLEMENTATION DETAILS

Prototype implementation of the proposed architecture was done on Motorola's 280i phone with J2ME platform. Prototype currently upgrades DSP software of the phone. Typically DSP software in 280i is responsible for all signal processing, vocoding and other multimedia related computationally intensive tasks.

4.1. Patch Profile Design

Patch Profile is designed while keeping following goals in consideration.

- Fail-Safe Installation
- Extensibility
- Reusability

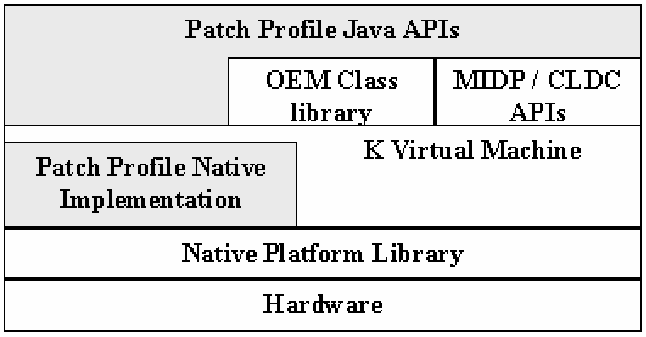

Figure 4: Patch Profile as stack

Figure 4 shows the patch profile as stack in J2ME environment. Following is the description of each component in the stack.

Installation of native software such as DSP software is a very platform dependant task. To achieve above design goals and to separate out platform dependant part of the patch profile from platform independent part Patch Profile implementation is divided into following two parts.

- Patch Profile Java APIs
- Patch Profile Native implementation.

Patch Profile Java API: This is platform independent part of patch profile, which exposes set of APIs to TM Profile and other applications. This part performs platform independent task such as storage management of patch and its license. For platform specific task, which requires native method execution Patch Profile Native implementation is called via Java Native Interface. Patch Profile Java APIs are designed in a generic way so that they handle different types of patches without any changes in the exposed Patch Profile APIs. This enables easy extensibility for different kind of software patch. Patch Profile Java APIs are LCC (Licensee Closed Class). For example, current patch profile supports only DSP patches to be installed in the phone. To add installation support for native applications there will be addition of installation functionality for that new patch type in the patch profile and there will be very few changes in the other components of the system. This part of patch profile implementation can be reused without any changes for other platforms also.

Patch Profile Native Implementation: Patch Profile Native Implementation includes the native code required to install a particular type of patch and gather version and resource information for a particular type of patch installation. Current Native Implementation includes native method implementation for getting current DSP software version, available space for installation process and installation of the DSP patch. Installation process of DSP patch uses patent pending mechanism for upgrading the DSP patch. This installation process is fails-safe and allows recovery of installation in error like power failure or system crash. Implementation of this part of patch profile changes depends on the platform architecture.

MIDP/CLDC APIs: These are standard APIs available for MIDlet development.

OEM Class Library (LCC)
Given the broad diversity of MEs, it is not possible to fully address all OEM requirements in by MIDP/CLDC libraries. These classes may be provided by an OEM to access certain functionality specific to a given ME.

K-Virtual Machine: KVM is a compact, portable Java virtual machine intended for small, resource-constrained devices such as cellular phones, pagers, personal organizers, mobile Internet devices, and so forth. All native functions accessed by Java code are part of KVM implementation.

Native Platform Library: Native platform library includes APIs provided by OS system calls and native libraries. Patch Profile native implementation uses this native libraries and OS system calls performing installation specific task. For example, DSP installation uses flash programming routines for re-programming the upgraded DSP software.

Hardware: Platform is Motorola's 280i phone hardware.

4.2. Patch

Figure 5 shows the patch packet format used in our implementation. Patch header gives the "type" information of the patch used by patch profile to call corresponding storing and installing functions.

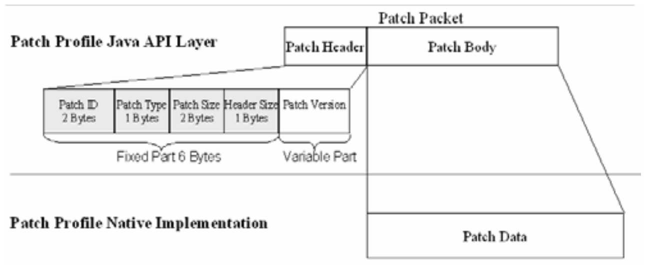

Figure 5: Generic Patch Format

Figure 6 shows demonstration sequence for our prototype implementation in which 280is vocoding software is being upgraded.

5. CONCLUSIONS

Software download is a necessary mechanism needed to support re-configurable feature of any SDR device. In this paper, we discussed the core problems and issues related to OTA device software management that future SDR mobile radios must address. We proposed a possible architecture solution for managing the mobile radio software over the air for deployment of software modules (upgrades of DSP software) below the application level in mobile terminals (280i).

A generic architecture for the proposed solution was discussed extensively by introducing the patch profile that separates out platform dependent part from platform independent part. We also made use of widely adopted SyncML standard for data synchronization in order to make software upgrade solution interoperable.

6. REFERENCES

[1] D. Efstathiou, J. Fridman, and Z. Zvonar, "Recent Developments in Enabling Technologies for Software Defined Radio", *IEEE Communications Magazine*, pp. 112-117, August 1999.
[2] "Globalization of Software Defined Radio", *IEEE Communication Magazine*, February 1999.
[3] "Special Issue on Software Radio", *IEEE ISAC*, April 1999.
[4] H. Tsurumi and Y. Suzuki, "Broadband RF Stage Architecture for Software-Defined Radio in Handheld Terminal Application", *IEEE Communications Magazine*, pp. 90-95, February 1999.
[5] R. Shepherd, "Engineering the Embedded Software Radio", *IEEE Communications Magazine*, pp. 70-74, November 1999.
[6] SyncML Specifications, http://www.syncml.org
[7] Sun Java J2EE Standard Specifications, http://java.sun.com/j2ee
[8] Sun Java J2SE Standard Specifications, http://java.sun.com/j2se
[9] Sun Java J2ME Standard Specifications, http://java.sun.com/j2me
[10] Mobile Information Device Profile, http://java.sun.com/products/midp/
[11] SyncML Representation Protocol, Device Managemet Usage, Version 0.8, 2002-02-13.
[12] SyncML Management Protocol, Version 0.9

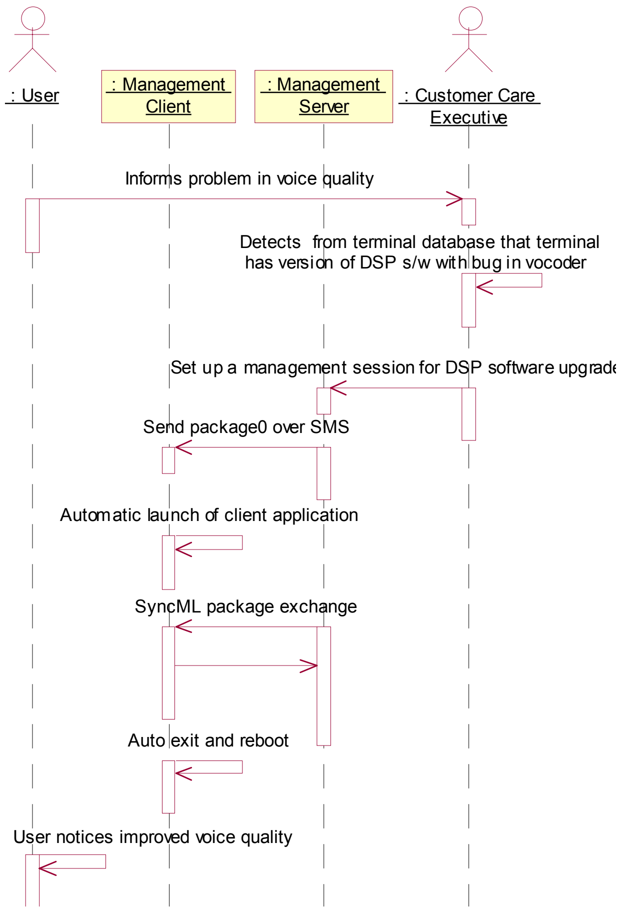
: User
: Management Client
: Management Server
: Customer Care Executive
Informs problem in voice quality
Detects from terminal database that terminal has version of DSP s/w with bug in vocoder
Set up a management session for DSP software upgrade
Send package0 over SMS
Automatic launch of client application
SyncML package exchange
Auto exit and reboot
User notices improved voice quality

Figure 6: Demonstration Flow for DSP software upgrade