MOBILE DEVICE MANAGEMENT WITH SYNCML

Alan Bok, Alan.Bok@motorola.com**,**
Sandeep Adwankar, Sandeep.Adwankar@motorola.com**,**
John Grosspietsch, John.Grosspietsch@motorola.com**,**
Venu Vasudevan, venuv@labs.mot.com**,**
Motorola Labs, 1301, East Algonquin Road, Schaumburg, IL 60196
Antonella Rinaldi, antonella.rinaldi@motorola.com
Motorola GSG Italy, Via Cardinal Massaia, 83 - 10147 Torino

ABSTRACT

Increasing complexity in wireless mobile devices is creating a need for remote management in a uniform and standard way. The SyncML specification is an open-industry, XML-based protocol, designed to facilitate synchronization of different devices and different networked data. SyncML based device management (DM) provides a unique way of managing millions of these devices by using the SyncML as a standard data exchange protocol among different entities across the network. In this paper, we outline the architecture of SyncML based device management and our work in designing and implementing both terminal side and server side SyncML DM functionalities. The paper discusses the end-to-end solution implemented by our current prototype to provide device management capabilities on GSM and GPRS based cell phones. The paper describes SyncML command implementation in native software to retrieve and configure phone specific data.

1. INTRODUCTION

Mobile devices like cell phones and PDA's are becoming more complex and providing more functionalities than those available in the past. With gradual integration of these devices in a single product, a cell phone is required to provide all features of a PDA and more. These changes are leading to a growing number of applications and user data residing on these multi-purpose cell phones. These devices have synchronization software allowing the user to manage and download new applications or synchronize PIM data with data stored on some different device

With rapid growth in the number and complexity of devices, the network operator is finding it extremely difficult to: reconfigure millions of devices for the latest firmware version, track the software inventory on each device and troubleshoot problems in these devices in an automated manner. The network operator needs the capability to manage mobile wireless devices Over-the-Air (OTA) remotely, reliably and securely.

What network operators primarily want from a Terminal Management System is to be allowed to query a mobile device for device parameter values in order to be helped in diagnostics the device itself, as well as to change values on the terminal to provide a requested service to the end customer. Thus following are two use cases of interest to network operator.

1.1. Terminal Tracking Use-Case

The terminal tracking use-case will give the Customer Support Center the ability to remotely query the Terminal for information such as Manufacture ID, Terminal Model ID, Software Revision ID, Serial Number and Current Web Session. During a Terminal Tracking session, the management operations are performed to locate and obtain the current status of the mobile devices.

1.2. Terminal Configuration Use-Cases

The terminal tracking use-case will give both the user and Customer Support Center the ability to remotely configure the terminal for information such as Current Clock, Alert Tone, Ringer Volume, Ring Tone, Web Session and downloading images. During a Terminal Configuration session, the management operations change the property/behavior of a mobile device.

1.3. SyncML Device Management

SyncML device management is a standard introduced to make use of widely adopted SyncML standard for data synchronization. SyncML data synchronization standard supports synchronization of networked data with any mobile device in a bearer independent way. Thus HTTP, WAP or SMS bearer can be used transparently by the SyncML protocol implementation.

SyncML Device Management consists of three main blocks:

1. *Protocol and Mechanism* – define the protocol used between a management server and a managed mobile device.

2. *Data Model* – defines the data made available for the remote manipulation.
3. *Policy* – decides who can manipulate or update a particular object on a device.

We developed a SyncML based device management system: a terminal side SyncML DM agent and a device management server is developed, both of which make use of the SyncML protocol. The rest of the paper is organized as follows. Section 2 introduces SyncML Representation protocol and Section 3 introduces SyncML Sync protocol. Section 4 describes our SyncML DM architecture. Section 5 describes some implemented terminal management operations. Section 6 describes terminal profile and Section 7 describes server profile. Section 8 shows performance results and Section 9 presents conclusions.

2. SYNCML DM REPRESENTATION PROTOCOL

The SyncML representation protocol [1] focuses on defining the data contents of the synchronization and methods for uniquely naming and identifying records. It also defines protocol commands and message containers.

2.1. Data Description Elements

Three element types classes are used to basic elements for data exchanged in the SyncML message. *Data* class specifies discrete SyncML data by accepting string as a parameter. *Meta* class specifies meta-information about the parent element type by encapsulating string type of Type and Format of data. *Item* class specifies a message for item data by using source and target URI information, Data and Meta classes.

2.2. SyncML DM Data Model

SyncML DM management objects are defined in [2]. A management object is an entity, which can be manipulated by management actions, carried over the SyncML DM protocol. Each management object has a type that determines what kind of management content can be set/read on that object. A management object might reflect a set of configuration parameters for a device: actions that might be taken against this object might include reading and setting parameter keys and values. A different management object might be the run-time environment for software applications on the device: actions that can be taken against this type of object might include installing, upgrading, uninstalling software elements.

Operations on a certain management object require predefined type of value to be sent and, the time the object being queried, value of that type being returned. For example, a certain management object can have a simple text type so that just simple text values can be set. At the same time a different management object might store a more complex type like the WAP Provisioning document type. Requiring that value set in that object comes with the WAP Provisioning document MIME type. Other more complex type for the management object might be the WAP setting type or installed software type.

Our Device Management implementation supports different types, starting from simplest one, like text plain, going towards more complex types like WAP setting, image, and software package.

2.3. SyncML Commands Category

There are two types of SyncML commands: Request and Response defined in [2]. Sequence is a super command that specifies the SyncML command to order the processing of a set of SyncML commands (Figure 1). There are two response commands: Status and Results. Status command specifies the request status code for a corresponding SyncML command. Result command specifies the SyncML command that is used to return the results of some request commands.

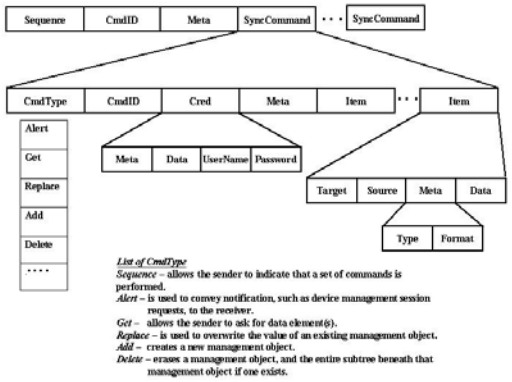

Figure 1 SyncML Request Commands Format

2.4. SyncML DM Management Tree

SyncML DM management objects, in accord to SyncML Device Management Tree and Description standard [7], have to be organized in a management tree. The management tree structures all management objects supported by a device in a hierarchy tree, addressing all objects with a unique URI. The URI is built traversing the management tree, where any single node has been assigned by a unique name.

3. SYNCML MANAGEMENT PROTOCOL

SyncML Device Management Protocol [5] allows management commands to be executed on SyncML DM management objects, where two phases (setup and management phases) are defined. Actions that can be taken against these objects might include reading and setting parameter keys and values.

3.1. SyncML Security

SyncML Protocol proposes following two security mechanisms. The SyncML Cred element includes the credentials information of SyncML message. The mechanism type and format must be specified in the Cred element.

3.1.1 Basic Authentication

SyncML basic authentication uses the Base64 Content-Transfer-Encoding [8] is designed to represent arbitrary sequences of octets in a form that need not be humanly readable. The encoding and decoding algorithms are simple, but the encoded data are consistently only about 33 percent larger than the unencoded data. In the SyncML Package 1, Sync Header contains Cred element. The type of credential is "syncml:authbasic" which is basic authentication and format is "b64" which is Base64 encoding. The encoded data is specified in Data element.

3.1.2 MD5 Digest Access Authentication

MD5 [9] authentication is another security mechanism supported in SyncML. This algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given pre-specified target message digest. The type in the Cred element in SyncML package must indicate "syncml:auth-md5" to identify MD5 authentication.

3.2. Capability Exchange

The mobile device must send device capabilities to the server during initialization. These will enable server to identify and locate a device in its own server database or to create a new entry for such a device in a database. In the Package 1, mobile device sends following device capabilities

- Manufacturer Name ***(Motorola)***
- Model Name ***(TP280)***
- Device Serial Number ***(IMEI:0000000000000)***
- Language Setting ***(en-US)***
- Device Management Software Version ***(R1.0.2)***

4. SYNCML DM SOFTWARE ARCHITECTURE

We modeled the terminal Management software system as a symmetric 3-tiered architecture as shown in Figure 2.

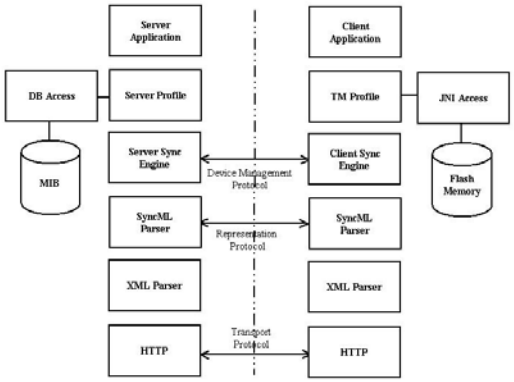

Figure 2 SyncML DM Software Architecture Diagram

We implemented the overall system for a cell phone use-case using Java technology: a new Servlet running on a Linux based Tomcat Web-server has been implemented on a server side while a MIDP 1.0 complaint MIDlet has been provided on a T280i Motorola phone. On both side three main software entities have been designed and implemented.

A *User Interface* has been provided to an end user (client application/server application), which allows him to interact with the system during a SyncML DM session and provide feedback about the result of executed management operations. A *Terminal Management Logic* engine has been implemented, which reacts to user/server requests enable the management of the terminal. This engine makes use of a *Namespace* mapping media that allows identifying the management object being managed in the server with internal data store objects in the client. Namespace defines the name and value of the management objects using the management tree that organizes all available objects in the device, where all management objects are addressed with unique URIs.

The *Terminal Management Logic* engine implementation in the system also provides the client/server application with the SyncML Parser and XML Parser features, as well as the implementation of the SyncML bearer on a HTTP transport layer protocol. At the time we implemented our prototype, one of the input we used in selecting the protocol on which implement the data synchronization protocol bearer, was the opportunity to exploit the HTTP protocol implementation required on any MIDP1.0 complaint device. In this way our prototype could be run on different devices, not just on mobile terminal,

which supports the HTTP protocol implementation. Minor customizations are required to manage these different kinds of devices and will be outline in next sections. A *Data Repository* database has been implemented on both sides that provides a common interface to get/set the objects defined in the Namespace and thus to perform the synchronization on them.

4.1. Namespace Based Data Synchronization

The management objects name in Namespace is used to identify the management object being managed across a network in a horizontal way. At the same time, Namespace also plays in a role of containing the global management objects name used both in internal SyncML software and Client Application running on a MIDP device in a vertical way. When we combine HTTP/XML Parser/SyncML Parser/Sync Engine into a Sync Agent from Figure 2, Namespace shows as a bridge between the SyncAgent and the Application as shown in **Figure 3**.

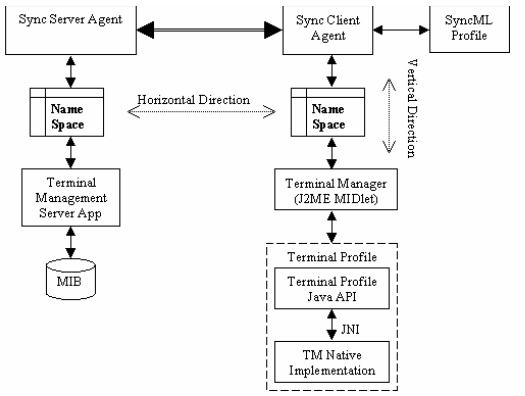

Figure 3 Namespace Based Data Synchronization System

4.2. Radio Parameter Management Objects Types

SyncML DM Protocol allows SyncML commands to be executed on networked management objects. Depending on the types of these objects, actions taken against them might include retrieving and setting parameter keys and values.
We classified the set of radio parameters supported by our prototype, on T280i terminal, in System, Network, Image, Memory and Audio parameters. The System radio parameters define terminal characteristics such as manufacturer identity, device model and identity, software and hardware version, CLDC and MIDP versions and the clock.

Most of the System parameters are supported as READ ONLY parameters because the terminal management operations query these values to the terminal. Jointly with the Memory parameters, the System parameters are called Capability Definition parameters. The Memory parameters provide in term of capability definition of the terminal, information about the maximum and available memory on the terminal and the memory schema supported by this. Differently all Network, Image Data and Audio parameters are READ/WRITE parameters. They allow us to track and configure WAP /Web setting values as well as different format of terminal supported images and ring settings.

Carrying out operations such as adding, deleting or replacing all these parameter types require native implementation in C on the terminal. This native implementation has been wrapped by a new Terminal Management Java profile residing besides the MIDP 1.0 profile: this new profile allows performing Over The Air Terminal Management operations.

5. CLIENT AND SERVER TERMINAL MANAGEMENT APPLICATIONS

The *Client* application in Figure 2 implements the Terminal Management MIDlet to allow a sync server to retrieve terminal information from the T280i phone and remotely configure it by changing data. The *Server* application, on the other side, manages a resource on behalf of multiple clients by storing the Networked Data resources in a MIB (Management Information Base) and providing a uniform access interface regardless of the actual implementation of the resources it manages.

The Client and Server applications jointly, perform the management operations on the T280i terminal according to SyncML DM protocol specifications [5]. The two applications communicate through the SyncML DM protocol using the services provided by the Client/Server Sync Engine and Terminal/Server Profile in order to perform a Sync Terminal Management session. A typical session is defined by the two phases of SyncML DM protocol shown in Figure 4, where the Management phase can be repeated as many times as the server wishes.

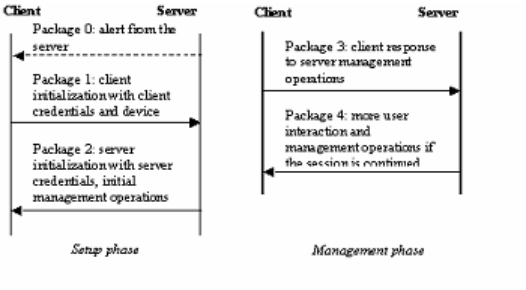

Figure 4 SyncML Protocol Phases

During the Set up phase, the Server might send a Package 0 to the Client in order to perform a Server initiate synchronization session. Because many devices cannot continuously listen for connection from a management server, most of them can receive unsolicited messages, so called, *notification messages* [6]. Our SyncML Server application, when a need for server initiated sync sessions is recognized, sends a SMS message to the terminal as a notification to cause the phone to initiate a connection back to the management server.

The SMS, for prototype purpose, has been coded in 7-bit default GSM format and comes along with the Alert Command (Alert Code 1200 for server initiated session [2]) in order to trigger a specific application on the phone. Triggering the Client application is automatic and it has been implemented in C code. The auto-trigger code allows the KVM running on the phone to wake up automatically in order to run the specific terminal management application.

The Client application might be run in a silent mode if the user has enabled the auto-start and auto-exit options for the MIDlet. The auto-start and auto-exit features allow the client application to connect back with the server, perform a terminal management section and exit without requiring any user interaction. In accord to Figure 4, once the client application is started, after opening a HTTP connection towards the server, it sends its own credential and device capabilities information to the server in order to perform Package 1. With the Package 1 the terminal application provides the server with device capabilities such as Manufacturer name, Model name, Device serial number, Language setting and Device management software version using the DevInfo class implementation defined by the Sync Engine. On receiving Package 1 from the Client, the Server application can identify and locate the terminal in its own internal database (MIB) or, if this terminal has never registered with the server, the server application creates a new database entry for it. In both cases, the server replies to Client agent sending its credential information with Package 2. The Package 2 might already include a first management operation. Thus the management actions might start from Package 2 of SyncML message flow and can continue for Package 4, Package 6 and so on. For any management action received by the terminal application, a response command will be sent back to the client containing the result or the status for the corresponding management action required by the Server application.

We implemented three different types of management actions in order to allow our prototype to support the use-cases applications: Tracking operations, Configuration operations and User Interaction operations. The tracking operation allows the Device Management Server to query the mobile terminal for device parameters: it has been implemented making use of the SyncML **Get** command. **Figure 5**-a shows an example of a tracking operation requested by the Server querying an active ring on the mobile. At the same time, when the Server application wants to change parameter values on management objects residing on the mobile device, the server has to use a configuration operation. This operation is required to provide a requested service to the end customer. The SyncML **Replace** command has been used to implement this type of operation. **Figure 5**-b illustrates the command to replace an active ring tone to value 2. If a classic ring tone is mapped to number 2, as it is on T280i phone, then with this operation the classic ring tone becomes the selected or an active ring tone. The last type of operation supported by our prototype is the User Interaction operation. This kind of operation allows the Server application to interact with the user to notify and obtain confirmation for a particular management action on the mobile device. The SyncML **Alert** command has been used for sending custom content information to the recipient. **Figure 5**-c shows a SyncML Alert asking the user to confirm the operation of updating to fewer ringer tones.

a) Active Ring Tracking Operation

```
<Replace>
  <CmdID>7</CmdID>
  <Item>
    <Target><LocURI>
    ./Sync/DM/Settings/ActiveRing
    </LocURI></Target>
    <Data>2</Data>
  </Item>
</Replace>
```

b) Ring Tone Configuration Operation

```
<Replace>
  <CmdID>7</CmdID>
  <Item>
    <Target><LocURI>
    ./Sync/DM/Settings/ActiveRing
    </LocURI></Target>
    <Data>2</Data>
  </Item>
</Replace>
```

c) User Interaction Operation

```
<Alert>
  <CmdID>2</CmdID>
  <Data> 1101</Data>
  <Item> MINDT=10</Item>
  <Item>
  <Data> Do you want to update the selected
         ringer?</Data>
  </Item>
</Alert>
```

Figure 5 Management Actions Examples

6. TERMINAL PROFILE

In designing our prototype software architecture, we had to extend the current set of profiles already available on mobile devices with a new one in order to access to actual hardware and software of our T280i phone. Actually most of the information a network operator wants to set/get from a mobile is provided in native platform dependent format: any manufacturer supports his own internal format for data and the way to retrieve and set it on a device. In order to access this kind of information we implemented in native code, but provide a standard interface to Java applications: the *Terminal Profile* defines this standard way to access underlay platform.

The *Terminal Profile* in Figure 2, along with *Platform Access Software,* defines a set of classes and interfaces, supported by native code implementation, allowing the Client Sync Engine to interact with the native platform of the mobile and to manage/configure it.

It also contains device specific classes and code that allows SyncML Engine software to access device-specific functionality such as persistence storage as well as platform-dependent parameter values. The Terminal Profile itself has been implemented providing a set of Java APIs compatible with other profiles running on the terminal. These Java APIs consist of a set of Java classes designed to handle domain-specific functionalities that greatly enhance the capabilities of terminal management application. All the applications developed using these classes are portable across different MIDP devices. The Terminal Profile Java APIs make use of the Platform Access Software to exploit native functionality such as querying device information as well as sharing data with native applications. The Platform Access Software is a platform specific component in CLDC written in platform dependent language. It is a not a portable component and is totally dependent on platform. In order to support the management operations required by the Client Sync Engine through the Terminal Profile, the Platform Access Software executes the newly designed AT command on the native platform. The Terminal Profile provides the interface to this native function from Sync Engine and any developed application. Our choice of implementing Terminal Management operations in terms of executing newly defined GSM AT commands has been driven by the will of minimizing the impact of porting the prototype on different platforms.

7. SERVER PROFILE

The server profile provides a management tree representation for the devices being managed (as stored in MIB). The current implementation of SyncML Server is that it is stateless (i.e., it does not maintain any state of terminal). The Replace operation is normally preceded by a Get operation to check whether the SyncML operation is required.

8. PERFORMANCE

Table 1 shows some Sync session performance benchmarking. The table shows the cost of an OTA sync section in time, when different management object types have to be configured over the phone on GPRS networks provided by two different Europe operators.

The timing results outline how more complex management objects require more execution time to complete a SyncML DM session between the Server application and our phone. By comparing the time requested for the same type on the two different networks, it's clear that the network delays impact the overall operation time. A 20% network delay time is due to the GPRS network provided by the second operator in the table.

Table 1 Sync Session Performance Measurement in msec

Management Object Type	**No of bytes exchanged**	**Sync session time (1)**	**Sync session time (2)**
DevInfo	2,144	18473	25465
Simple (plain/text)	3,729	32994	43037
WAP	4,315	35900	46923
Image	6,089	46584	57750

9. CONCLUSIONS

SyncML for device management has potential to become a standard way of managing millions of mobile devices. Our experiences with SyncML software architecture and implementation suggest that the protocol is very much suited for mobile devices having very limited capability in terms of memory and networking.

The extension of SyncML protocol and Sync Engine to perform device management tasks was considerably simple programmatically and sufficiently powerful to support all tracking and configuring needs. By enabling the Mobile Equipment (ME) to enhance its features over the air with the help of the servicing server, the duration for maintenance is reduced drastically, and also the user can chose a suitable time for maintenance.

10. REFERENCES

[1] SyncML Representation Protocol Version 1.0.1, 2000-05-30

[2] SyncML Representation Protocol, Device Managemet Usage, Version 0.8, 2002-02-13

[3] SyncML Device Management Demonstration, 2002-01-12

[4] SyncML Device Information DTD, Version 1.0

[5] SyncML Management Protocol, Version 0.9

[6] SyncML Notification Initiated Session, Version 0.92

[7] SyncML Device Management Tree and Description, Version 1.1

[8] IETF RFC 1521: MME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies

[9] IETF RFC 1321: The MD5 Message-Digest Algorithm

Formatted: Font: Not Bold

Deleted: Table 1