OMA DM-based remote software fault management for mobile devices

Joon-Myung Kang\textsuperscript{1,*}, Hong-Taek Ju\textsuperscript{2}, Mi-Jung Choi\textsuperscript{3}, James Won-Ki Hong\textsuperscript{1} and Jun-Gu Kim\textsuperscript{4}

\textsuperscript{1}Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea
\textsuperscript{2}Department of Computer Science, Keimyung University, Daegu, Republic of Korea
\textsuperscript{3}Department of Computer Science, Kangwon National University, Chuncheon, Republic of Korea
\textsuperscript{4}R&D Center, WebSync Corporation, Daegu, Republic of Korea

SUMMARY

Mobile devices (e.g. mobile handsets or PDAs) have gained much functionality and intelligence with the growth of mobile network technologies and the increased use of mobile services. As a consequence, mobile devices have become more complex and many related problems have occurred. Specifically, sudden rebooting and freezing problems caused by software faults decrease the availability of the mobile device and cause inconvenience to end-users. To solve such problems, academia and industry have focused on Open Mobile Alliance (OMA) Device Management (DM); this method is the international \textit{de facto} standard for mobile device management. In this paper, we propose a software fault management method to remotely determine and correct problems of mobile devices based on OMA DM. We present a definition of management objects and a method to collect them from mobile devices using the OMA DM protocol. We also present a method for debugging and correcting software faults using the collected information. Finally, we present a prototype implementation and performance evaluation to validate our proposed method. Results of the performance evaluation show that our proposed method is efficient and scalable in regard to network traffic overhead and response time. Copyright © 2009 John Wiley & Sons, Ltd.

1. INTRODUCTION

The growth of mobile networks and services has accelerated the introduction of various mobile devices such as mobile handsets and PDAs. These mobile devices have become more sophisticated and intelligent in order to satisfy the end-user’s various needs in terms of technology convergence, quality of service (QoS), and quality of experience (QoE) [1]. Consequently, these devices have become more complex and many problems with both hardware and software do occur [2]. In recent times, the major source of failure in mobile devices is software related rather than hardware related because it is very difficult for manufacturers of mobile devices to completely remove software bugs before selling them to end-users. Therefore, many mobile devices are sold with pre-existing software bugs, which cause the device to self-reset, freeze, or have system malfunctions. Cinque et al. [3] reported that the types of software failures in Symbian-based smart phones are output failure (36.3%), freeze failure (25.3%), unstable behavior (18.5%), self-shutdown (16.9%), and input failure (3%).

Currently, the most common method of solving these software faults is to reset (or reboot) the system, but this is only a temporary solution; users will still experience them as ongoing problems. Furthermore,
manufacturers often cannot find the root cause of these software faults even when they are reported to a service center by users, since it is difficult to reproduce the same faults in experiments. This makes it difficult to find the exact location of the fault in the source code of the software. The availability of the mobile device is also decreased when end-users encounter these problems because they must visit a service center for repairs. In addition, we should not forget the expense involved, not only for users to visit the service center but also for manufacturers to operate and maintain the service centers. These problems inevitably lead to a decrease in the credibility of service providers, network operators, and manufacturers of mobile devices. Although this type of remote software fault detection and correction service can be provided by mobile device vendors, service providers, or a third party, we believe mobile device vendors are the most appropriate party to provide such service.

In this paper, we propose a remote software debugging method to find and correct software faults of post-sale mobile devices. Using this method, software faults can be corrected by manufacturers and updated software versions can be provided to users via service centers; the corrected software can be applied to future mobile devices as well. This will alleviate inconveniences to the users and increase the perceived reliability of mobile devices.

One common method for software debugging is to dump an execution image and debug it using debugging tools when a fault occurs [4]. The execution image includes such things as registers, stack, and key events. For example, we debug using ‘core dump’ in the UNIX system, and report collected errors and logs to the Microsoft software management server in the Microsoft Windows system. We also use this debugging method to correct software faults with the execution image. That is, software faults of the mobile device can be fixed by using the dumped execution image when the device is automatically or manually reset. However, it is very difficult to collect the execution image of the mobile devices once they are in users’ hands, due to the low bandwidth and high error rate in the wireless network environment.

When debugging software faults, the mobile device’s system information such as CPU type and memory size is required as well as the execution image. It is also important that the information included in the execution image should be managed with high security, because it is very critical to the mobile device. To satisfy these constraints, the Open Mobile Alliance (OMA) Device Management (DM) framework [5,6] (formerly known as SyncML Device Management [7]) can provide an appropriate solution to manage mobile devices in the wireless network environment. OMA acts as the international de facto standard for mobile device management [8]. Much research [8–17] from areas of academia and industry has addressed issues of mobile device management based on the OMA DM framework. OMA DM defines management protocols and mechanisms that enable robust management during the entire life cycle of the mobile device. DM protocol of the OMA DM framework can efficiently collect large-scale data in the wireless network environment, with high security, since it was originally designed to accommodate the mobile environment and service requirements. The system information of the mobile device has already been defined as standardized management objects (MOs) [18]. We merely need to add new MOs to define all of the information required for remote software debugging. We therefore set out to define the MOs needed for creating the execution image and to design a process for collecting it.

The proposed remote software fault management method for mobile devices is based on the OMA DM framework. This method remotely debugs software faults, using an execution image when they occurred. First, we defined the MOs needed for reproducing the execution image. Second, we designed a method to collect them and debug the software faults using the execution images. Third, we designed and implemented a prototype to validate our proposed method; the prototype consists of a client and server. The client collects the system data related to the defined MOs, while the server retrieves such data and uses it to debug the software faults with a Joint Test Action Group (JTAG) debugger [19], which is capable of debugging target programs on mobile platforms. Lastly, we present the results of performance evaluation in terms of network traffic overhead and response time. Results show that our proposed method is efficient and scalable in regard to network traffic overhead and response time.

The organization of this paper is as follows. Section 2 describes an overview of the OMA DM and OMA DM Diagnostics and Monitoring (DiagMon) standard. Section 3 presents a review of previous
related work on mobile device management and general software debugging methods for mobile platforms. In Section 4, we describe the management architecture of our prototype system; we present a definition of the MOs needed to collect relevant information and explain the collecting processes. Section 5 illustrates the design and implementation of our prototype and the performance evaluation to validate our proposed method. Finally, we summarize our work and discuss directions for future research in Section 6.

2. OMA DEVICE MANAGEMENT

In this section we explain the specification of the OMA DM and the OMA DM DiagMon standard. We present a bootstrapping, a device description framework (DDF), and an OMA DM protocol, as well as functions defined by the OMA DM DiagMon Working Group (WG). We also describe the functions and the current status of the OMA DM DiagMon WG.

OMA [5] was established in June 2002 by a collaboration of mobile operators, information technology companies, wireless equipment vendors, and content providers. OMA DM WG [6], one of the major WGs in OMA, is responsible for the management of mobile devices, such as remotely managing and maintaining their basic configurations and applications. Currently, OMA DM is the international de facto standard for mobile device management [8]. OMA DM is a technology to improve Wireless Application Protocol (WAP) [20] and SyncML DM [7] technology and provides methods to efficiently manage mobile devices and their applications in various network environments as follows: (1) provisioning; (2) configuration maintenance and management; (3) software management; (4) fault detection, query and reporting; (5) non-application software download; and (6) configuration of user preferences. OMA DM specification includes management information for mobile devices in the form of a DM tree [18,21,22] and a management protocol for remotely managing mobile devices, known as the OMA DM protocol [23]; this is a SyncML-based protocol aimed at enabling remote synchronization of mobile devices. OMA DM standard is beneficial to many interest groups such as end-users, service providers, and manufacturers. It is convenient for end-users to operate due to automatic configuration, invisible performance and remote troubleshooting. It allows wireless network operators and enterprises to maintain all mobile devices in their networks. Using this standard, service providers can provide services with high reliability by remotely setting the appropriate configuration of their software, and device manufacturers can maintain the high availability and customer satisfaction of their mobile devices while the devices are in use. Thus OMA DM can alleviate operators’ and end-users’ problems and consequently improve their satisfaction with the quality of the product.

The OMA DM standard includes three logical components: a device description framework (DDF) [21], a bootstrapping process [24], and an OMA DM protocol [23]. DDF provides necessary information on MOs in devices for communicating between a DM server (which is managing the device) and a DM client (the device being managed). The bootstrapping process configures the initial settings of devices. OMA DM protocol defines the sequence of communicated packages by the DM server and the DM client. Each device that supports OMA DM must contain a management tree, to provide information [22] which organizes all available MOs in the device in a hierarchical tree structure where all nodes can be uniquely addressed with a uniform resource identifier (URI) [25]. OMA allows each device manufacturer to easily extend the management tree in their devices by defining and adding management nodes to the existing management tree and to define new management functions to added nodes. We show how this can be performed in our proposed method in Section 4.

Figure 1 shows OMA DM management sessions. Bootstrapping is needed when a DM server initiates a management session to a new DM client. OMA DM protocol is used when a DM server and a DM client exchange messages in management sessions. DDF and managed application components are only used within the OMA DM protocol. Technically, OMA DM uses XML for data exchange, more specifically it uses the subset defined by SyncML. As we explained in the previous paragraph, device management occurs by communication between a DM server and a DM client. OMA DM is designed to support
and utilize any number of data transports, such as: (1) physical layers over wired lines (USB, RS-232) as well as wireless media (GSM, CDMA, IrDA, IEEE 802.11 WLAN or Bluetooth); and (2) transport layers implemented over Wireless Session Protocol (WSP)/WAP [20], HTTP [26], OBEX [27] or similar transports [28] (see Figure 2). As illustrated in Figure 2, the OMA DM protocol consists of two parts: a setup phase (authentication and device information exchange) and a management phase, which is also a request–response protocol. In the setup phase, authentication and challenge of authentication are built in to confirm both server and client, and enable communication only after proper validation. Both the server and client are stateful, meaning specific sequences of messages are to be exchanged only after authentication has been completed to perform any task.

Communication is initiated by the OMA DM server asynchronously, using any of the methods available, such as a WAP Push or SMS (Package 0 in Figure 2). The initial message from server to client is
referred to as a notification, or an alert message. Once communication is established between the server and client, a sequence of messages might be exchanged to complete a given device management task using several OMA DM commands as follows [29]:

- **Add** (*adding an MO or MO content*) – Server creates a new node and returns error if there is an existing node; is not allowed to create a node at the Add target URI, or if the specified URI cannot be resolved.
- **Alert** (*management session start*) – This conveys notifications, such as device management session requests, to the client.
- **Atomic** (*processing commands atomically*) – Set of commands inside Atomic must be processed in the same way as commands inside Sequence, with all subordinate commands to be executed as a set or not at all.
- **Copy** (*copy MO content*) – This command treats the data of the copy and the data of the original independently after the copy is complete.
- **Delete** (*removing MO(s]*) – One or more MOs are removed from a management tree.
- **Exec** (*executing a process*) – New process is invoked and returns a status code or result.
- **Get** (*reading MO content or an MO list*) – Server retrieves the content from the DM Client or the list of MOs residing in a management tree.
- **Replace** (*updating MO content*) – Existing content of an MO is replaced with new content.
- **Results** (*return MO or MO contents*) – Client returns the MO or MO contents requested by the server.
- **Sequence** (*processing commands sequentially*) – One or more of Add, Replace, Delete, Copy, Get, Exec or Alert element types must be specified, and these element types must be processed in the specified sequence.
- **Status** (*return the status of processing*) – Client and the server return the status code after processing requested commands.

The server sends all commands except Status to the client. Status command is sent by both of them. We can design the data collection process for a mobile device by using a combination of these commands.

OMA DM WG has introduced device diagnostics and monitoring functionality to remotely solve some problems of mobile devices using the OMA DiagMon [39]. The overall goal of OMA DM DiagMon is to enable management authorities to proactively detect and repair problems even before they impact on users, or to determine actual or potential problems with a device as necessary. The management authority is an entity that has the rights to perform a specific DM function on a device, or manipulate a given data element or parameter. For example, the management authority can be the network operator, handset manufacturer, enterprise, or device owner. OMA DM DiagMon includes the following management areas: diagnostics policy management, fault reporting, performance monitoring, device interrogation, remote diagnostics procedure invocation and remote device repairing [40]. OMA DM WG publishes the standard documents in the following sequence: Work Item Document (WID), Requirement Document (RD), Architecture Document (AD), Technical Specification (TS), and Enablers Package (EP). The OMA DM DiagMon WG is currently working on TS. This only defines MOs for common cases of diagnostics and monitoring. We have now expanded MOs, however, to perform remote software debugging based on MOs defined by the OMA DM DiagMon WG. This paper therefore contributes a useful case study to demonstrate this capability.

3. RELATED WORK

In this section, we describe some previous research related to device management and general software debugging methods for mobile platforms.

Currently, several management frameworks for device management have been proposed. Simple Network Management Protocol (SNMP) [30,31] is the most widely used method for network
management on the Internet and on enterprise networks that use IP, and it has been standardized by the Internet Engineering Task Force (IETF). It is based on the manager–agent paradigm. The SNMP manager provides the interface to a human network manager to monitor and control the network, and interact with the SNMP agent through SNMP. SNMP agent is equipped in the managed device and responds to requests for information and actions from the SNMP manager. The Management Data Model, the Management Information Base (MIB), is organized as a tree structure with individual variables that have associated Object Identifiers (OIDs). SNMP supports simple commands like Set, Get, Get-Next and Trap, which combined with OIDs and Variable Bindings are exchanged between the SNMP agent and manager. Web-based device management solutions are widely deployed and share the popularity of web browsers. The Distributed Management Task Force (DMTF) defines the Web-Based Enterprise Management (WBEM) standard [32], which is a set of management and Internet standard technologies developed to unify the management of distributed computing environments. WBEM defines a Common Information Model (CIM) as a fundamental data model, an encoding specification and a transport mechanism for CIM operations over HTTP.

Chakravorty and Ottevanger [1] presented the Smart Box Management (SBM) system, which is an end-to-end remote management framework for Internet-enabled devices. In SBM, client devices securely communicate over the public Internet for device management-specific services such as remote registration, remote configuration, dynamic updates (downloads) and device diagnostic uploads with the SBM server. SBM uses HTTP to leverage a web-based device management infrastructure that offers several benefits: ubiquity, security, reliability and a high degree of user friendliness. However, they do not operate with a standard framework or protocol such as OMA DM; instead, they have devised their own proprietary protocol. Adwankar et al. [2] proposed a universal manager that manages both mobile and non-mobile devices in an enterprise. Their implementation is based on SyncML-based mobile devices integrated with SNMP-based enterprise manager. Also, they developed a multi-protocol gateway, which is a software entity that represents a terminal in the enterprise management system. The software makes the enterprise management system believe that the terminal is like a manageable entity in the enterprise. This study acted as a good trial for applying a SyncML solution to the question of mobile device management. Van Thanh et al. [14] presented the Device Management Service (DMS), which is considered as one big ‘virtual terminal’, with multiple input and output capabilities to suit every communication device. They also presented some user case studies. State et al. [15] presented an open-source agent toolkit built around the SyncML model. This was also a trial to test the SyncML device management framework. However, they considered only the framework required for the agent. Ma et al. [8] presented the device management architecture for IP Multimedia Subsystem (IMS) based on OMA DM. They presented the requirements of device management and management architecture which utilized distributed design by separating management and executive functionalities, offering service provisioning and tracing functionalities to application servers using some IMS capabilities. This work was a good example of the need for OMA DMs to manage mobile devices. Shin et al. [9] presented the design and implementation of an OMA DM-based mobile device management agent. This model provided more effective device management with a small amount of network traffic by using Tree and Description Serialization for acquiring, adding, and editing mobile devices’ objects. Lim et al. [10] presented a design for a network security framework based on a context-aware access control mechanism, which was interoperable with OMA DM protocol for device management. Ryu et al. [11] proposed and implemented a remote phone management system based on the OMA DM protocol. They designed the structure of an OMA DM client to adapt the existing terminal platform and validated it by using the SyncML Conformance Test Suite (SCTS) DM [7].

In the wireless communications industry, standardization bodies lead the research activities on the specified definition and mechanism of device management. For example, 3GPP proposed a technical report to study Universal Mobile Telecommunications System (UMTS) device management issues, known as a User Equipment Management (UEM) feasibility study [16]. A detailed management protocol is not defined in the UEM work but the OMA DM is strongly recommended. Nokia has developed and deployed many mobile devices that support the OMA DM standard [17]. They have developed a mobile device management solution in which an administrator is able to manage mobile devices remotely. They
have provided their solution for managing their mobile phones, based on the OMA DM standard, to the Nokia Forum. HP has also developed a Mobile Device Management (MDM) solution, which is part of the HP Service Delivery Platform (SDP) [33]. MDM allows operators, service providers, and handset manufacturers to manage mobile devices remotely. Innopath has developed two mobile device management suites; both the iMDM Server Suite and the iMDM Device Suite are based on OMA DM [34]. Both suites are modular, with integrated capabilities to remotely configure settings and new services, to diagnose and fix problems, and to update software and firmware over-the-air (OTA).

Currently, research activities in academic and industrial areas have launched OMA DM solutions for mobile device management as previously discussed. All of the work mentioned above focuses on remote mobile device management based on their proprietary management protocol or a standard protocol. However, current solutions are used for configuring or retrieving simple management information. Our previous work proposed a remote software debugging method to find and correct software errors in post-sales mobile devices and a remote RF monitoring method for improving the QoS of mobile devices based on OMA DM [12,13]. This work proposed practical and useful applications of mobile device management. We defined management objects and management operations for two useful cases using the OMA DM framework. Our current work is an extended version of [12] in providing a more practical definition of MOs and also performance evaluation of data collection processes.

In a typical software development process, debugging is one of the key activities. The purpose of debugging is to locate and fix the offending code responsible for a symptom violating a known specification. Debugging typically happens during three activities in software development: the coding process, the later stages of testing, and in production and deployment [4]. We are focusing on the third activity. One of the traditional debugging methods for mobile platforms, the JTAG debugger [19], is usually run before deploying mobile devices in the market. However, it is difficult to debug mobile devices once deployed because it is difficult to reproduce errors in experiments. No method has previously been proposed to record relevant information and remotely collect it in order to debug it. In this paper, we propose a practical method, using OMA DM, to record information about when a software fault occurs and then remotely collect it for debugging.

4. REMOTE SOFTWARE FAULT MANAGEMENT METHOD

In this section, we present our method for software fault management based on OMA DM. First, we present our overall management architecture. Then, we define MOs required for debugging software faults and design processes to collect relevant information from mobile devices. Figure 3 shows the overall management architecture of our proposed solution, which is composed of the DM Server and DM Client. DM Server sends the initialization and execution request to start a fault-detecting function in the DM Client. DM Client, which is equipped in various mobile devices such as PDAs, handsets, and laptops, replies with the result of the request from the DM Server. The analysis server obtains the data related to the fault and debugs it using the JTAG debugger. The presenter displays the process, the collected data, and the result of our management method.

4.1 Definition of management objects

We present a useful case study illustrating our proposed method: the software reset fault. The software reset fault, one of the major faults in mobile devices, causes rebooting of the device without the user’s intention. One method for debugging the software reset fault is to dump the execution image when the software reset fault occurs and then make use of the execution image, which includes such things as registers, stack, and key events. This method is commonly used in software debugging. We have defined the DM trees for software fault management (especially the software reset fault) as illustrated in Figure 4. We did this by expanding the DiagMon and DiagFunc nodes defined by the OMA DM DiagMon WG, and then creating the Reset node specifically for managing the software reset fault. Each node has its own
access control list (ACL), format, and scope attributes specified in the DDF [21], which are denoted in parentheses in each node of Figure 4. For example, the DM server can request GET, DELETE, and EXEC commands using the Reset node because its ACL is (G, D, E). The format defines the type of node and the format of the Reset node is a node type. The scope specifies whether the node is permanent or dynamic. The scope of the Reset node is dynamic. We have defined seven child nodes under the Reset node: DiagMonConfig, ServerID, DFID, Description, Operations, Status, and DiagMonData, according to the specification of OMA DM DiagMon. DiagMonConfig node is a placeholder for the configuration information of diagnostics and monitoring. This interior node has the following three child nodes:

- **DefDuration**: this node will stop the Diagnostics and Monitoring function if a time-limit threshold is reached. This time is expressed in seconds.
- **DefMemory**: this node will stop the Diagnostics and Monitoring function if a memory size threshold is reached. Amount of memory is expressed in bytes.
- **ReportCondition**: This node acts as a placeholder to contain different reporting conditions indicating when Diagnostics and Monitoring data are reported back to the Diagnostics and Monitoring System via the Generic Alert mechanism. Frequency node contains an integer value indicating the frequency at which an alert about Diagnostics and Monitoring data is reported back to the DM Server. The Diagnostics and Monitoring System uses the GET command on the ‘DiagMonData’ node to retrieve diagnostics data. For example, if the value is 0, the device should not send a generic alert to the Diagnostics and Monitoring System when diagnostics data are collected. If the value equals 1, the device must send a generic alert when diagnostics data are collected. If the value is greater than 1, the device must send a generic alert to the System when the ‘nth’ collection of diagnostic data occurs.
The ServerID leaf node is used to identify which DM Server the collected Diagnostics and Monitoring data should be sent to if the node is instantiated. The DFID leaf node contains a unique identifier that identifies the Diagnostics and Monitoring function. The value of the leaf node must be a unique registered identifier. The Description node provides a description of the Diagnostics and Monitoring function. The Operations node is a placeholder for operations that can be executed to start or stop the Diagnostics and Monitoring function. It has the following two child nodes:

- **Start:** this node is the target of an ‘EXEC’ command to start the Diagnostics and Monitoring function resident on the device.
- **Stop:** this node is the target of an ‘EXEC’ command to stop the Diagnostics and Monitoring function resident on the device.

The Status node specifies the operational state of the reset. Its value is one of the following:

- **NONE:** collection of reset data is halted.
- **PREPARED:** Exec command of data collection is received.
- **ACTIVE:** collection of data has started.
- **PROCESSED:** data have been collected.
- **REPORTED:** collected data have been sent to the DM server.

The DiagMonData node acts as a placeholder for the data. As shown in Figure 4(a), the child nodes of the DiagMonData node contain the relevant information for analyzing the software reset fault. We can reproduce the software reset fault case on the JTAG debugger using these data. It includes the following child nodes:

![Diagram](image_url)

Figure 4. DM tree definition: (a) DM tree for the software reset fault; (b) an XML document to define management objects for a software reset fault
• **Register**: register dump when the reset occurred.
• **Stack**: stack dump when the reset occurred.
• **UIPrimitive**: last UI primitives on the MMI (man–machine interface).
• **KeyEvent**: event value of KEY_EVENT and data on MMI.
• **ScenCb**: scenario and call-back data dump.
• **ExitCode**: pre-defined exit code value of the device.

Figure 4(b) shows the management object description according to the OMA DM DDF. We have defined it based on the hierarchy and attributes of Figure 4(a). This includes the definition and properties of the node. In the case of the **Reset** node, it is a sub-node of ./*DiagMon/DiagFunc as described in the *<Path>* tag which specifies the URI. The <DFProperties> tag aggregates the elements for describing framework properties. The <AccessType> tag specifies which commands are allowed on the node. The <Description> tag is the human readable explanation of the node. The <DFFormat> tag is the data format of the described node. <Occurrence> specifies the number of instances that may occur in the node. <Scope> specifies whether this node is permanent or dynamic. The <DFTitle> tag is the human readable name of the node. <DFType> is for leaf nodes, the MIME type of the node value.

### 4.2 Management processes

We have designed management processes for the software reset fault based on the DM tree defined in Section 4.1. We divide the management processes into three phases: initialization, execution, and data-gathering phase (gathering phase). In the initialization phase, the DM Server checks whether the mobile device has functionalities related to diagnostics and monitoring (diagnosis of a software reset fault) or not by adding functional nodes such as the **Reset** node. If the mobile device has the functionality to diagnose the software reset fault, it returns ‘OK.’ Otherwise, it returns ‘Not Found.’ In the execution phase, the DM Server applies policies and executes the Reset detecting program, which can identify whether the software reset fault is present or not. Finally, in the gathering phase, the DM Server collects relevant data from the DM Client when it notifies about the problem event. By dividing the management operation into three phases as shown in Figure 4, we achieve an efficient management process. First, each management phase is composed of the same management commands. Hence a single management command can process an operation of many MO addresses, which decreases the length of the management session. Second, each phase can be independently used for its own purposes. That is, to diagnose the software reset fault, all three phases do not need to be repeated. Once the initialization is processed, it does not need to be repeated. Also, after the execution phase, there is no need to repeat it to process data gathering, as long as the policy for collecting the data remains unchanged. Therefore, it is more efficient than processing all three phases. We present the results of performance evaluation for validating our proposed method in Section 5.3.

Figure 5 presents the sequence of management processes for diagnosing the software reset fault. Figure 5(a) shows the initialization phase. When the DM Server wants to initialize the function diagnosing the software reset fault, it needs to send a **NOTIFICATION** command to the DM Client [26]. When the DM Client receives the **NOTIFICATION** command, it sends the server-initiated **ALERT** (1200) command to the DM Server with the device name. Next, the DM Server sends the **ADD** command to initialize the diagnosis function and the **REPLACE** command to set ACL for the Reset node as **GET**, **DELETE**, and **EXEC**. For efficiency, as shown in package #4 of sequences 5, 7, and 9 in Figure 5(a), the DM Server adds many MOs by using one **ADD** command. If the mobile device supports the diagnosis function for the software reset fault, then it can add a **Reset** node to its DM tree and send a successful **STATUS** command (200) to the DM Server. If the **Reset** node is added successfully, then the DM Server adds **DiagMonConfig**, **Status**, and **DiagMonData** nodes step by step. Finally, the DM Server sends a ‘Completion’ message to the DM Client to complete this management session. After the initialization phase, the mobile device is ready to execute the reset diagnostics function.
Figure 5. Design of management processes for the software reset fault: (a) initialization phase; (b) execution phase; (c) gathering phase
Figure 5(b) shows the execution phase of diagnosing the software reset fault. When the DM Server wants to execute the reset diagnosis function, it sends the NOTIFICATION message as in the initialization phase. Then the DM Client sends the server-initiated ALERT command to the DM Server. The DM Server sends the REPLACE command to set the policy information. The DM Server sets a time limit threshold to the DefDuration node and a memory-size threshold to the DefMemory node. The DM Server also sets the frequency to Frequency node when an alert about diagnosis data is reported back to the DM Server. If the mobile device has initialized the reset diagnostics function, then it sends the status code as 200 (success). Otherwise, it sends the status code as error code. Then, the DM Server sends the EXEC command to execute the reset diagnosis function. The DM Client executes the reset collecting process, which is a module to collect data related to the reset problem in the mobile device. Finally, the DM Server sends the completion message to the DM Client and the management session is finished.

Figure 5(c) shows the gathering phase of diagnosing the software reset fault. When the software reset fault occurs, the DM Client stores all relevant information to its DM tree. When it is ready to report, it sends the generic ALERT command to the DM Server. If the DM Server needs to gather the information about this mobile device, it sends the GET command for the DiagMonData node to retrieve all information related to the reset. It saves these data to the database. As mentioned earlier, we can reproduce the software reset case on the JTAG debugger using these data and find the error in the source code.

5. VALIDATION

In this section we present the system development, based on the DM tree and management processes presented in Section 4, for validating our proposed method. Then, we present the result of the performance evaluation of management processes in terms of response time and network overhead.

5.1 System design

Our proposed system is composed of a DM Client and DM Server, as illustrated in Figure 6. DM Server sends initialization and execution requests for the diagnosis function to the DM Client. The DM Client, which is equipped in various mobile devices such as PDAs, cellular phones, and laptops, responds to the requests by the DM Server. The DM Server and DM Client support WSP over WAP [20], object exchange (OBEX) over IrDA [27], and hypertext transport protocol (HTTP) over TCP/IP [35] as transport protocols. The bootstrap protocol is used to process provisioning of the DM Client to a state where it is able to initiate a management session to a new DM Server. The representation protocol uses a type of OMA DM protocol such as Add, Get, and Replace commands.

The major component of the DM Server is the Reset Tester. This function processes the initialization phase and the execution phase on the user’s request based on the management processes outlined earlier in Section 4.2. When the DM Client notifies the reset, it runs the gathering phase to retrieve data. Major components of the DM Client are DM Tree Handler and the Reset Detecting Process. DM Tree Handler manages the MOs for software fault management by commands which follow the manager’s requests. Reset Detecting Process detects the software reset fault in the mobile device, collects the relevant information when the software reset fault occurs, and fills the Reset DM tree with the information. It can detect the software reset fault by the system flag, which shows 0 on normal rebooting and 1 on abnormal rebooting.

5.2 Implementation and testing

We developed the DM Client and the DM Server based on the design in Section 5.1. We implemented the DM Client on a mobile platform based on Intel PXA270 (624 MHz), which is a next-generation Intel® XScale® processor based on the ARMv4 instruction set architecture [36]. We have ported the embedded Linux, kernel version 2.6.15.7 and developed a DM Client based on the open-source project, SyncML
Figure 6. The architecture of the software fault management system: (a) architecture of DM Client equipped to the mobile device; (b) architecture of DM Server.
Conformance Test Suite DM Version 1.2 (SCTS) [7]. SCTS is a tool that can check if the Test Object implements all the mandatory features of the OMA Device Management protocol, (formerly called the SyncML protocol). SCTS provides core libraries for communicating with the OMA DM protocol. Since SCTS originally operated on the Microsoft Windows operating system, we ported it to operate on embedded Linux. In addition, we implemented extra functions for software fault management.

The mobile platform which supports our DM Client has a WLAN interface (802.11 b/g), a Bluetooth interface, a CDMA interface, and an HSDPA interface for communication. We have also implemented the DM Server based on the open-source project called Funambol [37], which provides basic functions using the OMA DM protocol based on the JBoss Application Server [42]. We have extended it to manage software faults of the mobile device. Figure 7 shows a screenshot of our DM Server, which manages the software reset fault. We can start a diagnosis process setting limited duration, limited memory, and frequency for the report condition (Figure 7(a)). We can collect relevant information on the software reset fault, save it to the database, and show it on the web page (Figure 7(b)). The DM Server gathers the data from the DM Client and the data are used to create the execution image for debugging the software reset fault using the JTAG debugger, which is a debugging tool for embedded systems.

Figure 8 shows a screenshot of debugging the software reset fault on the JTAG debugger using our developed system. First, we generate a specific script file for debugging with the mobile device model; this is a configuration of the debugging environment and the debugging information collected in the gathering phase. Then, we input the script file to the JTAG debugger, which shows the location of the error in the source code in the Data.list window. Note that the name of the source file and function is shown in the Symbol.info window. We can now find the error location and correct it.

5.3 Performance evaluation
In carrying out the performance evaluation for our proposed method we have tested the performance between our DM Server and DM Client, also mentioned in Section 4.2. We have evaluated the latency performance, which is the roundtrip time for sending and receiving a message between the DM Server and the DM Client. We also evaluated the bandwidth performance, which is the overhead of network traffic. As described in Figure 9, we performed the test with the DM Server and DM Client. For the DM Server we used a Window server with a Pentium Dual Core 1.6 GHz CPU and 2 GB RAM, in which Apache Tomcat server was running. For the DM Client, we used Embedded Linux with a PXA270 Intel Scale 624 MHz CPU and 128 MB SDRAM. The DM Server and wireless access point were connected by a 100 Mbps Ethernet link, whereas the wireless access point and the DM Client were connected by a wireless network based on IEEE 802.11b. In this experiment, we show that our proposed method requires low network traffic overhead and fast response time in a wireless network environment. We performed two experiments by increasing the concurrent requests from 1 to 100 by 10 and investigated the processing time of each request. Concurrent requests were generated using a thread mechanism. When we performed the management process, we collected the generated OMA DM messages as a unit of a session. We measured the total bytes of collected messages and the response time by analyzing the packet header information of each session because it has the size information (length) of each message. We measured the average response time by the difference between a starting time and an ending time of each message at the DM Server.

In the first experiment, we tested how to compose each phase of the management session efficiently. As we mentioned in Section 4.2, we designed each phase to send the same commands through one management session simultaneously. At the initialization phase there are many ADD commands to initialize the related information variables. If each ADD command is transferred through its own management session, then it requires heavy network traffic overhead and long response times. Basically, the OMA DM protocol supports the transfer of many commands in one management session. We considered this fact and designed this phase of the management operation accordingly. The initialization phase is composed of 17 ADD commands and one REPLACE command, as shown in Figure 5. Network traffic overhead of the initialization phase, when each command is transferred through separate management
Figure 7. Implemented DM Server: (a) example of diagnosis configuration in the execution phase; (b) example of data collected during the data-gathering phase which can be viewed via a web page.
Figure 8. Screenshot of debugging software reset fault on the JTAG debugger using collected data from the mobile device

Figure 9. Experimental environment for performance evaluation of data collection in the proposed method

sessions, is 25,286 bytes. However, the network traffic overhead of the initialization phase by our proposed method, which generates only one initialization phase, is 3791 bytes. In the case of one ADD command, its network traffic overhead is 1336 bytes. Consequently, our proposed method is efficient because the network traffic overhead of the initialization phase requires the same network traffic overhead as triple ADD commands. Figure 10 shows the comparison of response times between our proposed method and an operation with commands treated separately in accordance with the number of requests.
As the number of requests increases, the response time of the operation with separate commands increases tremendously compared to our proposed method. On average, the response time of the initialization phase when each command is transferred through a separate management session is 4.68 s, but with the initialization phase of our proposed method it is 0.66 s. In the case of one ADD command, its response time is 0.23 s. This result shows that our proposed method is efficient because the response time only requires a couple of ADD commands.

The second experiment measured the network traffic overhead of the management operation by dividing it into three phases and comparing it with a single management operation. Network traffic overhead of the initialization phase was observed to be 3791 bytes; that of the execution phase was observed to be 2234 bytes; and that of the gathering phase was observed to be 3025 bytes. The single management operation uses 7750 bytes. Figure 11 shows the comparison of response times between our proposed method and each phase of the management process for mobile devices in accordance with the number of requests. As the number of requests increases, the response time of one session without dividing increases highly compared to our proposed method. On average, the response time of the initialization phase is 0.66 s, while that of the execution phase is 0.26 s, and that of the gathering phase is 0.48 s. The single management operation takes 1.81 s. As mentioned in Section 4.2, this result also confirms that our proposed method, which divides the management operation into three phases, is efficient in terms of both message size and response time. Both experiments show that our proposed management operation is efficient and scalable in regard to network traffic overhead and response time.

6. CONCLUSIONS

Software fault management in mobile devices has become an important area in the field of mobile device management. In this paper, we have presented a remote software fault management method for mobile
devices based on OMA DM. Our proposed method is appropriate for wireless network environments since our management protocol generates low-bandwidth messages and acquires fast response times, as shown in Section 5.3. Since it is based on the OMA DM framework, which is the international de facto standard protocol for managing mobile devices in the wireless communications environment, our proposed method can be widely used for managing mobile devices. First, we defined the MOs for managing the software reset fault; we then designed management processes in three phases (initialization, execution, and gathering phase) for efficiency of network traffic overhead and response time. Second, we presented the development of the software fault management system based on defined MOs and management processes and described how to analyze the software reset fault using the JTAG debugger. Finally, we demonstrated that our proposed method is efficient and scalable in regard to network traffic overhead and response time through a performance evaluation.

To date, much research has investigated SNMP-based management systems for IP-networked devices, but there has been little work on OMA DM-based management for mobile devices. Most previous work on mobile device management focused on development of a DM Client and DM Server based on the OMA DM framework. Our contribution is not only the development of a DM system based on OMA DM, but also the creation of a practical, useful case study on remote software fault management. We also illustrated how to define MOs in the form of a DM tree using DDF, and how to design the management process using management commands of OMA DM. By using this proposed method, system developers will be able to easily find and correct software faults of deployed mobile devices.

For future work, we plan to improve our prototype to repair problems automatically by merging the debugging system (JTAG debugger) and generating an appropriate patch. In addition, we will expand our method to apply it to other useful cases such as radio frequency signal monitoring [13], call monitoring and calibration of mobile devices as well as software reset fault. Ultimately, we will improve our prototype to enable it to self-diagnose problems in mobile devices and self-heal them using the concept of Autonomic Computing proposed by IBM [38].

Figure 11. Number of requests and response time of management processes
ACKNOWLEDGEMENT

This research was supported in part by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute for Information Technology Advancement) (IITA-2009-C1090-0902-0045) and by the WCU (World Class University) program through the Korea Science and Engineering Foundation funded by the Ministry of Education, Science and Technology (Project No. R31-2008-000-10100-0).

REFERENCES


**AUTHORS’ BIOGRAPHIES**

**Joon-Myung Kang** received his BSc degree in computer science and engineering from POSTECH in 2005. From 2000 to 2004, he worked at Alticast Corporation as a software engineer. Currently, he is a PhD candidate in the Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea. His research interests include autonomic network management and wireless mobile device management.

**Hong-Taek Ju** received his BSc degree in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 1989, and his MSc and PhD degrees in the Department of Computer Science and Engineering from POSTECH in 1991 and 2002, respectively. From 1991 to 1997 he worked at Daewoo Telecom as a senior software engineer. Currently, he is an assistant Professor in the College of Communication and Information, Keimyung University. His research interests include web-based network management, network monitoring, and data synchronization over wireless mobile communications. He is a member of KNOM.
Mi-Jung Choi is an assistant professor in the Department of Computer Science, Kangwon National University, Chuncheon, Korea. She received her BS degree in computer science from Ewha Womans University in 1998, and MSc and PhD degrees in the Department of Computer Science and Engineering from POSTECH in 2000 and 2004, respectively. She was a Postdoctoral Fellow at INRIA, France, from 2004 to 2005 and at the School of Computer Science, University of Waterloo, Canada, from 2005 to 2006. She worked in POSTECH as a research Professor from 2006 to 2008. Her research interests include XML-based network management and NGN and Future Internet management. She is a member of IEEE and KNOM.

James Won-Ki Hong is the head and professor in the Department of Computer Science and Engineering, POSTECH. He received a PhD degree from the University of Waterloo in 1991 and an MS degree from the University of Western Ontario in 1985. His research interests include network and systems management, distributed computing, network monitoring and analysis, and network planning. He has served as Technical Chair (1998–2000), Vice Chair (2003–2005), and Chair (2005–present) of the IEEE ComSoc Committee on Network Operations and Management (CNOM). He has also served as Director of Online Content for IEEE ComSoc (2004–2005). He is a NOMS/IM Steering Committee Member and a Steering Committee Member of APNOMS. He was technical co-chair of NOMS 2000 and APNOMS ’99. He was Finance Chair for IM ‘09, NOMS ’06 and IM ’05, and Finance Chair and Chair of the Local Planning Committee for NOMS ’04. He was General Chair of APNOMS ’06 and APNOMS ’08. He is a General Co-Chair of NOMS 2010, which will be held in Osaka, Japan, in April 2010. He is an editorial board member of JNSM, IJNM, TTM, and TNSM. He is also Editor-in-Chief of KNOM Review Journal. He is a member of KICS, KNOM, and KISS.

Jun-Gu Kim is a chief software engineer in the R&D Center, WebSync Corporation, Daegu, Republic of Korea. His research interests include device management and data synchronization based on OMA DM.