Design and implementation of a mobile database for Java phones

Eric Jui-Lin Lu*, Yung-Yuan Cheng

Department of Information Management, Chaoyang University of Technology, 168 Gifeng E. Rd., Wufeng, Taichung County, 413, Taiwan, ROC

Received 29 September 2003; received in revised form 28 November 2003; accepted 8 December 2003

Abstract

Due to the popularity of electronic commerce and the maturity of wireless technology, mobile users can access the Internet anytime anywhere without having to physically hook up any access point. To be able to manage and manipulate data downloaded from the Internet, there is a strong need for developing mobile database systems for mobile devices. Therefore, in the paper, we designed and implemented a mobile database system for Java phones. XML is adopted to describe databases. In addition, XSLT is used to define processing rules for databases so that only a general processor is required to process databases of different schemata. It is demonstrated that the proposed mobile database system is portable and interoperable.

© 2004 Elsevier B.V. All rights reserved.

Keywords: Mobile databases; Mobile commerce; XML; XSLT

1. Introduction

Due to the maturity of wireless technology, users can now use mobile devices such as PDAs, cellular phones, and laptop computers to access the Internet anytime and anywhere without having to physically connect to the fixed network. With mobile devices, mobile users can download data from the Internet or share their data (or a subset of the data) with other users [22]. To make data management and sharing easier, there is a strong need for developing mobile database systems for mobile devices. Mobile database systems are different from fixed-host distributed database systems for mobile devices. Mobile database systems are shared between mobile databases and fixed-host databases [3]. Traditionally, the research on mobile databases is centered around mobile devices that have full capabilities as of those desktop computers [1,16]. Until recently, the development of small mobile database systems for PDAs, cellular phones, and smart cards is starting to become an important research area [17,19].

The mobility and limited resources on mobile devices pose new challenges in the design of mobile database systems [17–21]. In summary, mobile database systems (1) should have small footprints to fit into resource-limited devices, (2) should support simple insert, delete, update, and query function even in disconnection mode and do not have to have complex functionality compared to their desktop counterparts, (3) are able to run without the services of a database administrator, and (4) are interoperable and portable between mobile databases and large enterprise databases. Among these requirements, interoperability and
portability are the biggest challenges faced by developers [17]. Since almost every business person carries a cellular phone, and since cellular phones with Java capability (called Java phones) are increasingly growing, the major goal of this paper is to design and implement a portable and interoperable mobile database system for Java phones.

Currently, the development of small database systems for mobile devices can be divided into two categories. One is to design database systems for PDAs. Because PDAs have less resource constraints, commercial light-weighted database systems such as IBM DB2 Everyplace [10], Oracle 9i Lite, and Microsoft SQL server CE are available for PDAs. However, because cellular phones generally have fewer resources than PDAs, these light-weighted database systems cannot be used on Java phones [2]. In addition, even with these commercial databases, it is extremely difficult to manipulate an arbitrary database downloaded from other devices because of the differences in database schemata [13,17].

The other category is to design database systems for smart cards [2,8,9,12]. Database smart cards generally contain a database engine (i.e. a dedicated smart card operating system) which processes and manages data stored in smart cards. To query or update data in a database smart card, the card has to be connected to a host computer and users interact with the card through a host program. Because it is required that a database smart card has to be attached to a host computer, the mobility is greatly restrained.

To develop a portable and interoperable mobile database system for Java phones, several issues have to be addressed. To ensure database can be exchanged and reused between various types of mobile devices, Extensible Makeup Language (XML) is employed to describe data. In addition, processing rules are extracted from application programs so that only a generic processor is required to be installed for each Java phone. Extensible Stylesheet Language Transformation (XSLT) is adopted to define processing rules. The processor manipulates XML data based on the definitions of processing rules. The proposed mobile database has the following benefits: Since both data and processing rules are encoded in XML (i.e. XSLT is actually an application of XML.), and since the processor is developed in Java, the developed mobile database is interoperable and portable. In addition, because any Java phone with network connectivity such as GPRS or dial-up modem can download both data and processing rules from the Internet or other mobile devices, there is no need for an administrator to manage the mobile database. Furthermore, with the processor, mobile user can perform simple query, update, insert, and delete operations on the received database.

The generic processor is in fact an XSLT processor. However, currently, there is no XSLT processor which can run on Java phones [15]. Therefore, a tiny-XSLT processor was also developed. The current version of the tiny-XSLT processor supports the following XSLT grammars: xsl:stylesheet, xsl:variable, xsl:template, xsl:apply-templates, xsl:element, xsl:text, xsl:copy-of, xsl:value-of, and xsl:if. Because the supported XSLT grammars are a subset of the W3C XSLT recommendation, the developed mobile database can also be executed on devices such as PDAs, laptops, and desktop computers that have full installation of any XSLT processor.

The rest of paper is organized as follows: The current development of mobile database systems are discussed in Section 2. In Section 3, the design of the proposed mobile database system is described in details. And then the implementation of the tiny-XSLT processor and the mobile database system are described in Section 4. Finally, we shall conclude this paper and future works in Section 5.

2. Related works

Because of the compactness, convenience, and mobility of small mobile devices such as cellular phones and smart cards, the development of small mobile database systems for these mobile devices has emerged as an important research area. In 1996, the ADEPTE project [8] developed a small database for smart cards acting as a part of a large federated database. Each smart card stored local data related to its owner and references to remote data. When remote data are queried, the smart card generates remote requests which are passed through its host computer to remote database and the queried results are returned and displayed on the host computer.

Bobineau et al. [2] designed a small database for smart cards called PicoDBMS. Due to limited memory
resources on smart cards, the design of PicoDBMS was concentrated on the compactness of storage model and query executions without consumption of memory. To reduce space consumption on data storage, a storage model based on a combination of flat storage, domain storage, and ring storage is used. To determine the best storage for each attribute, the database developers have to know whether or not attributes need to be indexed as well as the average length and selectivity factor of the attributes. In query processing, PicoDBMS constructs an extreme right deep tree and then uses pipeline technology to avoid unnecessary memory consumption for intermediate results.

Kuramitsu and Sakamura [12] proposed an ubiquitous database which allows various applications to access data objects in a contactless smart card. The database manages data objects from different organizations. To represent various types of data objects, each data object is composed of multiple properties and each property is described by its corresponding object identifier, the property name, type, and value. When an object is queried, the query language is transformed into primitive commands on a host computer and these primitive commands are then executed on the smart cards.

By taking advantages of both database and multi-applications smart card models, Jean et al. [9] proposed a hybrid smart card model. The hybrid smart card model uses the embedded database to manage and control data and programs (called card applets) sharing. Users can query through either the internal database engine (i.e. ISO 7816-7 RDBMS) or card applets in smart cards. Each card applet can accomplish its tasks individually or cooperatively with other card applets. Additionally, each card applet can be seen as a “package” that can be created, deleted, or executed.

Although the reviewed database models for smart cards are small enough so that they may be considered to be ported to Java phones, there are a few drawbacks. The majority of smart cards have a hierarchical file system defined in ISO 7816-4 [7,12], while the file structure on Java phones is record stores [6]. Therefore, the data models designed for smart cards cannot be used in Java phones. Furthermore, because it is required that a database smart card has to be attached to a host computer to accomplish a task, the mobility is greatly restrained. As of writing, to our knowledge, there is only a set of APIs, called JavaPhone [5], designed for developing databases for Java phones. However, because what JavaPhone provides is a set of APIs, it is extremely difficult for mobile users to simply download data from the Internet or other mobile devices and start manipulating the data. Thus, we attempt to design and implement a mobile database for Java phones to fulfill all requirements stated in the previous section.

3. Mobile database design

Extensible Markup Language (XML) was officially released in February 1998 by W3C. Due to its simplicity, flexibility, and richness in structure, it is believed that XML is the de facto standard for data interchanges [15]. Therefore, XML is adopted to describe databases in the proposed model so that databases can be easily exchanged and reused between various types of mobile devices. Fig. 1 shows an example of phone book encoded in XML. As shown in the example phone book, three individuals including cheng, iron, and cheng-rex are recorded in the phone book and each

```
<phonebook>
  <person>
    <name>cheng</name>
    <number>0935000001</number>
    <age>31</age>
    <sex>male</sex>
    <address>Taipei</address>
    <email>cheng@mail.cyut.edu.tw</email>
  </person>
  <person>
    <name>iron</name>
    <number>0935000002</number>
    <age>24</age>
    <sex>male</sex>
    <address>Taipeii</address>
    <email>iron@mail.cyut.edu.tw</email>
  </person>
  <person>
    <name>cheng-rex</name>
    <number>0935000003</number>
    <age>25</age>
    <sex>male</sex>
    <address>Taichung</address>
    <email>cheng-rex@mail.cyut.edu.tw</email>
  </person>
</phonebook>
```

Fig. 1. An example of XML Phone Book.
individual is described by using elements such as name, number, age, sex, address, and email.

Because XML documents can be parsed by a Document Object Model (DOM) parser, developers can design programs to query, update, insert, and delete elements on XML documents. We utilized kXML [4] which is a small DOM parser for Java 2 Micro Edition (J2ME) and designed a mobile database for Java phones [14]. This mobile database system is small and can be installed on any Java phone. However, it lacks both flexibility and interoperability. Whenever a database with different schema is downloaded, all of its associated programs (called MIDlets) have to be downloaded and installed. This is time consuming. In addition, because there is no defined mechanism that allows different MIDlet suites to share persistent data [6], kXML has to be included in every MIDlet suite which is space consuming.

To overcome the above problems, a new approach is taken in this paper. Instead of embedding processing rules in programs, the processing rules are extracted from the programs. The processing rules are then described in XSLT. This approach has the following benefits: (1) XSLT is an application of XML and thus can be easily exchanged and reused, (2) because the processing rules are extracted from programs, only one general processor called DBEngine is required for different databases which makes maintenance easy, (3) the DBEngine has to be installed only once in each device, and (4) any database (in XML) and its associated processing rules (in XSLT) can be downloaded and processed by users. The last benefit increases the portability of the mobile database system. Furthermore, since any user can download any database and its associated processing rules and start processing the database without having to know its schema in advance, this significantly increase the interoperability of mobile databases.

The architecture of the proposed mobile database system for Java phones is shown in Fig. 2. The mobile database system contains a database which is encoded in XML, a set of processing rules, and a processor called DBEngine. The DBEngine contains a controller, a DOM parser, a tiny-XSLT processor. The controller is in charge of database management and user interface handling. The DBEngine has to be installed into a mobile device before the mobile database system becomes functioning. A user with the DBEngine installed on her/his mobile device can receive a database and its associated processing rules from the Internet or other mobile devices. After saving the database and processing rules into their mobile devices, the user can start manipulating database based on the provided processing rules.

To delete an element from the database, the processing rule encoded in XSLT is designed as shown in Fig. 3. In general, lines 1, 2, and 3 as well as the last line of processing rules are identical for all processing rules. In line 4, xsl:variable is used to define the element and its value which will be deleted. As shown in Fig. 3, the Person element whose name is “cheng” is to be removed. To locate the element, lines 5 and 7 are first

Fig. 2. The architecture of the proposed mobile database system.
used to locate the root element of the database. Once the root element is located, line 6 is used to process all of the children nodes of the current node which is actually the root element. Similarly, all of the children nodes of /PhoneBook will be processed in lines 8 to 12. Note that lines 9 and 11 will output /PhoneBook/i and /PhoneBook/i, respectively. Everything that will be processed or generated in line 10 will be placed between /PhoneBook/i and /PhoneBook/i. In line 13, all children nodes of /PhoneBook/Person will be processed based on the rules defined between lines 14 and 18. As shown in line 14, if the value of /PhoneBook/Person/name (which is denoted as name) is not equal to "cheng" (which is denoted as $name), the whole Person element will be generated. In other words, the Person element whose value is "cheng" will be removed from (or not generated in) the result tree. Elements are copied into the result tree. Note that any Person element whose values is not "cheng" can also be removed. This is because the controller will parse the processing rule in the beginning and replace "cheng" with whatever entered by users. This step will be performed for every processing rule. The processing rule for query is very much like the processing rule for deletion except that line 14 is now (xs1:if test = "name=$name").

The processing rule for modification is illustrated in Fig. 4. Like the previous processing rules, xs1:variable is used to define the value of an element to be modified. As shown in lines 2, 3, and 4, the values of elements—number and age—will be modified for the Person element whose name is "cheng-rex". Additionally, as shown in lines 8 to 19, if the value of the selected /PhoneBook/Person/name is equal to "cheng-rex", the values of its corresponding elements number and age will be replaced by the new values entered by users. Otherwise, as shown in lines 20 to 22, the selected /PhoneBook/Person will be copied into the result tree. Note that the content of line 5 in Fig. 4 actually includes all statements from lines 5 to 12 in Fig. 3. These statements are intentionally omitted from the figure for clarity.

To append a new /PhoneBook/Person element into the database, the controller will retrieve input values from users and then replace the corresponding values of the select attributes from lines 2 to 7 as shown in Fig. 5. Then, according to the processing rules defined in Fig. 5, the XSLT processor will locate the first /PhoneBook/Person element and apply the processing rules defined in lines 22 to 24 to the database. Line 23 in Fig. 5 contains the statements from lines 15 to 17 in Fig. 3 and is actually copying all existing /PhoneBook/Person

```xml
<%xml version="1.0" %>
<xs1:stylesheet version="1.0" xmlns:xs1=
    "http://www.w3.org/1999/XSL/Transform">
<xs1:variable name="name" select="cheng "/>
<xs1:template match="/" >
<xs1:apply-templates/>
</xs1:template>
<xs1:template match="PhoneBook">
<xs1:element name="PhoneBook">
<xs1:apply-templates/>
</xs1:element>
</xs1:template>
<xs1:template match="Person">
<xs1:if test="not(name=$name)">
<xs1:element name="Person">
<xs1:copy-of select="*"/>
</xs1:element>
</xs1:if>
</xs1:template>
</xs1:stylesheet>
```

Fig. 3. The processing rule for deletion.

```xml
<%xml version="1.0" %>
<xs1:stylesheet version="1.0" xmlns:xs1=
    "http://www.w3.org/1999/XSL/Transform">
<xs1:variable name="name" select="cheng-rex "/>
<xs1:variable name="number" select="0928000001 "/>
<xs1:variable name="age" select="26 "/>
<xs1:template match="Person">
<xs1:element name="Person">
<xs1:if test="name=$name">
<xs1:element name="number">
<xs1:text>$number</xs1:text>
</xs1:element>
<xs1:element name="age">
<xs1:text>$age</xs1:text>
</xs1:element>
</xs1:if>
<xs1:copy-of select="sex"/>
<xs1:copy-of select="address"/>
<xs1:copy-of select="email"/>
</xs1:if>
</xs1:element>
<xs1:if test="not(name=$name)">
<xs1:copy-of select="*"/>
</xs1:if>
</xs1:element>
</xs1:template>
```

Fig. 4. The processing rule for modification.
elements to the result tree. Finally, in lines 12 to 19, the new element is appended to the result tree.

To process processing rules encoded in XSLT, an XLST processor must be embedded in the DBEngine. Unfortunately, existing XSLT processors that we know for PC or PDAs are too big to fit into Java phones. Therefore, a small XSLT processor called tiny-XSLT was developed for Java phones in this project. To make the tiny-XSLT processor as small as possible, the number of required XSLT grammars is restrained to nine and they are xsl:stylesheet, xsl:variable, xsl:template, xsl:apply-templates, xsl:element, xsl:text, xsl:copy-of, xsl:value-of, and xsl:if.

4. Implementation and analysis

Sun’s J2ME (Java 2 Micro Edition) wireless toolkit supports the development of Java applications that can be executed on mobile devices. Therefore, J2ME is adopted to implement the mobile database. To parse XML documents, a small DOM parser called kXML [11] is employed. As stated previously, there is no appropriate XSLT processor for Java phones. Therefore, a small XSLT processor called tiny-XSLT is developed. The tiny-XSLT processor is not intended to be a general-purpose XSLT processor with full functionality. Rather, it is mainly designed for supporting the transformations that are needed in the mobile database. The tiny-XSLT processor can process nine XSLT grammars including xsl:stylesheet, xsl:variable, xsl:template, xsl:apply-templates, xsl:element, xsl:text, xsl:copy-of, xsl:value-of, and xsl:if. Because the supported XSLT grammars are a subset of the standard XSLT recommendation, processing rules can also be run on PCs and PDAs that have a regular XSLT processor installed. The size of the current implementation of DBEngine is about 53 kb. For a phone book of 100 entries, the average time required to insert a new entry is little bit over 1 s, while the average time required to delete and update an entry is less than 1 s.

4.1. The prototype

To be able to download or share a database and its associated processing rules in one click, the developed prototype implements a source file of the format as

![Fig. 5. The processing rule for appending elements.](image-url)
shown in Fig. 6. The name of the to-be shared database is defined in line 1. Line 3 contains all XML data as of those XML data shown in Fig. 1. All functions provided for the database are defined using element names such as append, delete, modify, and query. For example, if two insert processing rules are designed for the database, there will be two \(<append\) elements in the file. Line 6 contains the processing rules for appending elements as shown in Fig. 5. Similarly, lines 9, 12, 15, and 18 contain processing rules for one deletion, two modifications, and one query, respectively. The values of attribute name of elements append, delete, modify and query represent the labels that will be displayed on Java phones.

Once the source file is downloaded, the controller will check whether or not a database of the same name exists. If so, the existing information will be removed.

Then, the source file will be processed and a new file as shown in Fig. 7 is created. Since the file structure in J2ME consists of record stores and each record store may contain records of different sizes, and since each record store can be named as a string and each record can be named as record identification number, each database is stored in one record store and the name of the record store is the name of the database as shown in line 2 in Fig. 7. The XML database by default is always stored in the record whose identification number is 1. Then, each of the processing rules for each function is processed and stored into records with identification numbers incremented by one. As shown in Fig. 7, the processing rules for the two append functions are stored in records 2 and 3. Similarly, the processing rules for the delete, modify, and query functions are stored in records 4, 5, and 6, respectively. The label for each function is enclosed as the content of the function elements. For example, the label for the modify function is modify-name. Note that, as shown in Fig. 7, the proposed mobile database system supports multiple databases, named PhoneBook and OrderList, in the system.

Fig. 8A shows the names of databases installed on a Java phone. When users choose to process the

```xml
1 <db name="PhoneBook">
2 <xmlData>
3  <!-XML Data-->
4  </xmlData>
5  <append name="append-all">
6    <!-Processing Rules for append-all Data-->
7   </append>
8  <delete name="delete-name">
9    <!-Processing Rules for delete-name Data-->
10  </delete>
11 <modify name="modify-name">
12   <!-Processing Rules for modify-name Data-->
13 </modify>
14 <modify name="modify-address">
15   <!-Processing Rules for modify-address Data-->
16 </modify>
17 <query name="query-name">
18   <!-Processing Rules for query-name Data-->
19 </query>
20 </db>
```

Fig. 6. An example of database to be shared.
PhoneBook database, the DBEngine lists all of its provided functions which is shown in Fig. 8B. Fig. 8C displays two modification processing rules for the PhoneBook database. Based on the variable names defined in lines 2 to 4 in Fig. 4, the DBEngine shows three text fields which allow users to enter values. The screen shot of this process is shown in Fig. 8D. As shown in Fig. 8D, after the values are entered, the DBEngine will search for the person whose name is “cheng-rex” based on the processing rules in Fig. 4. Once an element is located, the values of the number and age elements of “cheng-rex” will be replaced with the newly entered values. Fig. 9A shows users can query the PhoneBook database using a processing rules labelled as “query-name”. Fig. 9B demonstrates the screen shot which allows users to input a name to query the PhoneBook database. Fig. 9C shows the query result which also demonstrates the value of the number of “cheng-rex” had been changed to “0928000001”. The layout of the screen shots is controlled by tiny-XSLT. For example, to display a query screen as shown in Fig. 9B, tiny-XSLT first reads its associated XSLT file as shown in Fig. 10. Because there is only one xs:variable element, tiny-XSLT will use the value of its name attribute to be the label and create one text field. The value of its select attribute is set to be the default value of the text field. Similarly, to display the query result as shown in Fig. 9C, tiny-XSLT reads the result tree generated from the XSLT file in Fig. 10 and displays each element name as the label followed by the content of the element.

4.2. Analysis

In this section, the benefits of the proposed mobile database system are analyzed.

- Since XML is adopted to describe the database, and since processing rules are defined in XSLT, the proposed mobile database system allows databases to be exchanged and reused between various types of mobile devices. For example, a user can receive a phone book database from her/his friends and reuse this database. This benefit allows the user to reuse the phone book without having to re-type the whole phone book. Additionally, it also significantly reduces possible input errors entered by the user.

- With processing rules, users are able to process the received database without the need to develop new programs. Users are allowed to perform some simple query, update, insert, and delete operations on the received database. One major benefit of using processing rules is that users are able to process data without knowing the schema of the database. This feature significantly increases the interoperability of the proposed mobile database system.

- As long as a copy of the DBEngine is installed on a mobile device, it can process databases of different schemata. This is because processing rules are defined separately. For example, a mobile user may have downloaded a tourist guide and query the hot spots that the user is interested. And later, the user may download a shopping catalog. In spite of the
different schemata of these databases, the mobile device has the capability to process them without changing processor. Furthermore, the proposed mobile database system is so easy to use such that there is no need for an administrator to manage the database.

- The kXML parser and tiny-XSLT processor support only a subset of DOM and XSLT specifications. Therefore, the proposed mobile database system can also be used in PDAs, laptops, and desktop computers that have full DOM and XSLT support.
- The proposed mobile database system supports multiple databases. As shown in Fig. 8, the proposed mobile database system can install more than one database with different schemata.
- Since the resources on Java phones are generally limited, it is nice for users to be able to download only a portion of a database that is required. Because the database is described in XML, the proposed model can process and/or share a portion of a database, if necessary, with a server that can generate the required XML data and put them into a file in a format as shown in Fig. 6.

5. Conclusion

As Java phones are becoming more and more popular, users cannot only play games, but also accomplish business tasks on these phones. Due to frequent disconnections, there is a strong need for developing a mobile database system for Java phones so that users can manage and manipulate data even when they are travelling. In this paper, a mobile database system is designed and developed. In the proposed mobile database system, databases can be exchanged and reused between different mobile devices. In addition, a tiny-XSLT processor was developed to process processing rules. Although it has been demonstrated that the proposed mobile database system is portable and interoperable, the system can be further enhanced to increase the acceptance of normal users. For example, a user-friendly utility can be developed so that users can create a database and design its associative processing rules without the knowledge of XML, J2ME, and XSLT.

Acknowledgements

This research was partially supported by the National Science Council, Taiwan, ROC, under contract number: NSC91-2218-E-324-003.

References


Eric Jui-Lin Lu received his B.S. degree in Transportation Engineering and Management from National Chiao Tung University, Taiwan, R.O.C., in 1982; M.S. degree in Computer Information Systems from San Francisco State University, CA, in 1990; and Ph.D. degree in Computer Science from University of Missouri-Rolla, MO, in 1996. He is currently an associate professor of the Department of Information Management and the director of the Graduate Institute of Networking and Communication Engineering, at Chaoyang University of Technology, Taiwan, R.O.C. His current research interests include electronic commerce, distributed computing, and security.

Yung-Yuan Cheng received his M.S. degree in Information Management from Chaoyang University of Technology, Taiwan, R.O.C., in 2003. His research interests include mobile commerce and mobile database. He is currently a software engineer at Universal Scientific Industrial Co., Ltd.