Institutionen för datavetenskap
Department of Computer and Information Science

Final Thesis
A Reporting System for a Device Management Application

by
Marcus Svensson

LIU-IDA/LITH-EX-A--2009/001--SE

2009-01-27

Linköpings universitet
SE-581 83 Linköping, Sweden
Final Thesis

A Reporting System for a Device Management Application

by

Marcus Svensson

LIU-IDA/LITH-EX-A--2009/001--SE

2009-01-27

Supervisor : Dr. Pål Nilsson
OMSI Forum
at Cybercom Sweden South AB

Examiner : Prof. Henrik Eriksson
Dept. of Computer and Information Science
at Linköping University
Abstract

Device Management Applications are applications which are used to manage software on devices such as mobile phones. OMSI Forum provides such an application which is used to update the software on a phone. Software updates can be done at device management stations placed in stores or other service locations. Whenever a phone’s software is updated, information about the update process is stored in a log. These logs can then be analyzed to generate statistics about updates such as the number of successful or failed updates or which faults that are common.

This master thesis project solves the problem of manually collecting and compiling logs from several stores by making this process automatic. Rather than collecting logs manually, each device management station sends its logs to a centralized server which stores all collected logs in a database. This makes it possible to generate charts and statistics in a simple manner from a web application. This solution makes the analysis more effective, allowing users to concentrate on analyzing data by removing the work task of collecting logs.

Keywords: Device Management, Reporting System, OMSI, remote reporting, remote logging, hierarchical tree structures in databases, nested set model, adjacency list model, buckets, Enterprise JavaBeans, Web Services, JBoss, MySQL
Contents

1 Introduction .................................................. 1
   1.1 Background ............................................. 1
   1.2 Problem Definition and Purpose ....................... 2
   1.3 Methodology ............................................. 2

2 Theory ................................................................ 3
   2.1 Open Mobile Service Interface ......................... 3
       2.1.1 OMSI Reference Client ............................. 3
   2.2 Web Services .............................................. 5
   2.3 Java Enterprise Edition 5 ............................... 5
       2.3.1 Java Application Server ............................ 6
       2.3.2 Enterprise JavaBeans ............................... 6
       2.3.3 Stripes Framework ................................. 7
   2.4 Databases .................................................. 7
       2.4.1 Database Management System ..................... 7
       2.4.2 Database Modelling ................................. 8
       2.4.3 Managing Hierarchical Tree Structures .......... 8
       2.4.4 Normalization ...................................... 11
       2.4.5 Buckets ............................................. 14

3 Architecture and Design ...................................... 15
   3.1 Conceptual System Model ............................... 15
   3.2 Architecture .............................................. 16
   3.3 Interfaces and Data Types .............................. 18
3.3.1 OMSI Reporting ................................. 18
3.3.2 Centralized Logging ............................. 18
3.3.3 Statistics and Administration .................. 20
3.4 Client Module .................................. 23
3.5 Centralized Logging Server ........................ 23
3.6 Statistics and Administration .................... 25
3.7 Database ...................................... 26
  3.7.1 Modeling .................................. 26
  3.7.2 Log, Phonemodel and Manufacturer tables .... 28
  3.7.3 Store and Territory tables .................... 28
  3.7.4 User, Role and Role_Acl tables ................. 28
  3.7.5 Bucket tables ................................ 29
3.8 Security ...................................... 29
  3.8.1 Centralized Logging ........................... 29
  3.8.2 Statistics and Administration ................ 30
4 Measurements and Simulations .......................... 33
  4.1 EJB versus Web Services .......................... 33
  4.2 Measuring the Application ........................ 35
    4.2.1 Data Size .................................. 35
    4.2.2 Reporting Performance ......................... 36
    4.2.3 Statistics Performance ......................... 39
5 Discussion ..................................... 41
  5.1 Conclusions .................................. 42
  5.2 Future Work .................................. 42
A Abbreviations .................................. 43
B Guides ........................................ 45
  B.1 Installation of Client Module ....................... 45
    B.1.1 Preparation .................................. 45
    B.1.2 Copying Files ................................ 45
    B.1.3 Configuration ................................ 46
  B.2 Installation of Database .......................... 47
    B.2.1 Preparation .................................. 48
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.2.2 Configuration</td>
<td>48</td>
</tr>
<tr>
<td>B.2.3 Loading Database Dump</td>
<td>48</td>
</tr>
<tr>
<td>B.3 Installation of Centralized Logging Server</td>
<td>49</td>
</tr>
<tr>
<td>B.3.1 Preparation</td>
<td>49</td>
</tr>
<tr>
<td>B.3.2 Configuration</td>
<td>49</td>
</tr>
<tr>
<td>B.3.3 Deployment</td>
<td>50</td>
</tr>
<tr>
<td>B.4 Installation of Statistics and Administration</td>
<td>51</td>
</tr>
<tr>
<td>B.4.1 Preparation</td>
<td>51</td>
</tr>
<tr>
<td>B.4.2 Configuration</td>
<td>51</td>
</tr>
<tr>
<td>B.4.3 Deployment</td>
<td>51</td>
</tr>
<tr>
<td>B.5 Maintenance</td>
<td>52</td>
</tr>
<tr>
<td>C Source Code and Java Documentation</td>
<td>53</td>
</tr>
<tr>
<td>Bibliography</td>
<td>54</td>
</tr>
</tbody>
</table>
List of Tables

2.1 PhoneUpdateReport specification [1] ....................... 4
2.2 Example of an unnormalized database ......................... 12
2.3 Example of a database in NF1 ................................ 12
2.4 Example of a database in NF2 ................................ 13
2.5 Example of a database in NF3 ................................ 13

3.1 Enhanced PhoneUpdateReport specification ..................... 19
3.2 Remote logging interface storePhoneReport ................... 20
3.3 LogManager interface ............................................. 20
3.4 StoreManager interface ............................................ 21
3.5 TerritoryManager interface ......................................... 21
3.6 UserManager interface ............................................. 22

4.1 Average row size in database tables ........................... 35
4.2 Server setups used in measuring of logging performance .... 38

A.1 List of abbreviations ............................................... 43

B.1 Client module file listing ......................................... 46
B.2 Database file listing ................................................ 48
B.3 Centralized logging server file listing .......................... 49
B.4 Statistics and administration file listing ....................... 51
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Overview of the OMSI system</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Tree representation example</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>Nested set model example</td>
<td>10</td>
</tr>
<tr>
<td>3.1</td>
<td>Conceptual view of the system</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>System architecture</td>
<td>17</td>
</tr>
<tr>
<td>3.3</td>
<td>Sequence diagram of the centralized logging server</td>
<td>24</td>
</tr>
<tr>
<td>3.4</td>
<td>Overview of the statistics and administration subsystem</td>
<td>25</td>
</tr>
<tr>
<td>3.5</td>
<td>EER model of the database</td>
<td>27</td>
</tr>
<tr>
<td>4.1</td>
<td>Performance comparison between EJB and Web Services</td>
<td>34</td>
</tr>
<tr>
<td>4.2</td>
<td>Number of reports when 50 stores are using the system</td>
<td>36</td>
</tr>
<tr>
<td>4.3</td>
<td>Average service time (uS) in reporting performance measurements, JBoss time values are measured inside the actual application and total values are measured on the client side.</td>
<td>37</td>
</tr>
<tr>
<td>4.4</td>
<td>Total service time (uS) in reporting performance measurements, JBoss time values are measured inside the actual application and total values are measured on the client side.</td>
<td>37</td>
</tr>
<tr>
<td>4.5</td>
<td>Graph showing measured query time when using buckets</td>
<td>40</td>
</tr>
</tbody>
</table>
Listings

3.1 Example of access control ........................................ 26
4.1 Query used to measure performance without buckets ........ 39
4.2 Query used to measure performance using buckets .......... 40
B.1 OMSI reference client manifest file ............................ 46
B.2 Client module configuration file ................................. 47
B.3 Commands for load database dump ............................... 48
B.4 Centralized logging server database connectivity configuration 50
B.5 Example of code which rebuilds the “total” bucket .......... 52
B.6 Example of code which rebuilds the “histogram” bucket .... 52
Chapter 1

Introduction

This report is the result from a master thesis project carried out at Cybercom Sweden South in Malmö, autumn 2008. The author is a student at the Computer Science and Engineering programme, at the University of Linköping. It is recommended but not required that the reader has some basic knowledge about web technology and databases to be able to benefit from the content of this master thesis report.

1.1 Background

In 2007, 1.15 billion mobile phones were sold throughout the world [2]. Some phones may at some point malfunction and will then need a repair or other service. If the problem is software related, one measure that can be taken is to perform a software update of the phone, updating its firmware to a new version or restore it to factory default settings. This measure often solve problems with phones and can be performed in device management stations placed in e.g. stores or other service locations. Each service computer is referred to as a device management station. The application which is used to update software in mobile phones is called device management application.

In many cases the software updates are successful and solve problems
with a phone. When an update is not successful, the phone can be sent away for further diagnostics and repair. The result from updating a mobile phone in a device management station is stored in a log. The log contains information such as which phone that was updated, whether the software update was successful or not and the time and date when the update was made. This type of information can be very important to various organizations, as it will make it possible to trace all phone updates and see e.g. statistics on which phone models that are commonly having problems or how many successful or failed software updates that are done in each store.

1.2 Problem Definition and Purpose

In a big organization with a large number of device management stations, manually collecting and analyzing logs from individual stations can become a burdensome work task. Instead of doing this manually a more elegant solution would be to have a centralized server to which all device management stations send their logs and from which statistics can be generated.

The purpose of this project is to design and implement such a solution where device management stations send their logs to a centralized server, from which statistics can be generated. This would eliminate the need to manually collect logs and limit the administrative task to only generate and analyze statistics in reports.

This master thesis project will produce source code together with related documentation and manuals of a system that solves the problem outlined in this section. The result will be presented at seminars and this report will serve as documentation for this master thesis project.

1.3 Methodology

Most of the theory used in this master thesis report has been obtained through reading technical manuals, specifications and other literature related to the area. Prototypes were developed to test techniques and implementations before the full implementation of the reporting system was done.
Chapter 2

Theory

This chapter summarizes the theory used in this master thesis report and gives the reader background information about the technology, techniques and frameworks that were used throughout the project.

2.1 Open Mobile Service Interface

Open Mobile Service Interface (OMSI) is a web service interface specification for device management applications managed by OMSI Forum [3]. The idea with OMSI is to minimize the number of applications service-point PCs need to maintain. Instead of having several applications and/or PCs for different phone models and manufacturers, only one device management application is needed. Manufacturers contribute with plug-ins which enable their phone models in the device management application. Figure 2.1 visualizes the OMSI concept.

2.1.1 OMSI Reference Client

Cybercom develops the reference client application for OMSI Forum which implements the most current version of the OMSI standard, version 2.2 [3]. This master thesis project used this reference client application for im-
2.1. Open Mobile Service Interface

Figure 2.1: Overview of the OMSI system

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>userName</td>
<td>String</td>
</tr>
<tr>
<td>phoneIdentifier</td>
<td>PhoneIdentifier</td>
</tr>
<tr>
<td>modelName</td>
<td>String</td>
</tr>
<tr>
<td>dateTime</td>
<td>DateTime</td>
</tr>
<tr>
<td>reportedError</td>
<td>String</td>
</tr>
<tr>
<td>warranty</td>
<td>int</td>
</tr>
<tr>
<td>selfTestResult</td>
<td>int</td>
</tr>
<tr>
<td>fullSelfTestResult</td>
<td>base64Binary</td>
</tr>
<tr>
<td>refurbish</td>
<td>boolean</td>
</tr>
<tr>
<td>backup</td>
<td>boolean</td>
</tr>
<tr>
<td>updateOK</td>
<td>boolean</td>
</tr>
<tr>
<td>restoreOK</td>
<td>boolean</td>
</tr>
<tr>
<td>oldFirmwareVersion</td>
<td>FirmwareVersion</td>
</tr>
<tr>
<td>newFirmwareVersion</td>
<td>FirmwareVersion</td>
</tr>
</tbody>
</table>

Table 2.1: PhoneUpdateReport specification [1]
implementing the reporting system. The reporting interface and data types from the OMSI specification was used in communication between the device management application (the reference client application) and the client module. This can be seen in Figure 3.1. Each report is contained in a data structure which is shown in Table 2.1. The implementation of the OMSI reporting interface is further discussed in Section 3.3.1.

2.2 Web Services

The reference client application and the client module use Web Services for communication. Web Services is a communication protocol which is used to integrate systems and standardise how systems communicate with each other. Web Services use Extensible Markup Language (XML) technologies such as Simple Object Access Protocol (SOAP) and Web Service Description Language (WSDL) for its implementation. SOAP is used for remote invocation of methods on a server and WSDL is used to describe the services provided by a Web Service [4]. World Wide Web Consortium (W3C) provides the definition of Web Services [5]:

“A Web Service is a software system identified by a URI [RFC 2396], whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.”

Java 6 has a collection of tools that eases Web Services development. When a WSDL-file already exists which defines services, one of the tools, wscompiler, can generate Java stubs which can be used as a starting point for developing Web Services [4].

2.3 Java Enterprise Edition 5

Java Enterprise Edition 5 (Java EE) is a platform which is used to implement Service-Oriented Architectures (SOA). It is a widely used platform for developing enterprise applications [6].
2.3. Java Enterprise Edition

2.3.1 Java Application Server

To deploy enterprise applications an application server is needed. A number of application servers are available on the market today, both through commercial licenses and open-source licenses. Some popular choices are JBoss, GlassFish and JOnAS [7].

JBoss is an application server which is used to run Java applications. Two flavours of JBoss are available, one with a commercial license (JBoss Enterprise Middleware) which includes support and one open-source edition (JBoss Application Server) [8, 9].

Sun GlassFish Enterprise Server is the application server which Sun Microsystems provides. Both an open-source and a commercial version of the software is available [10].

Java Open Application Server (JOnAS) is a Java application server which is published under an open-source license [11].

In this master thesis project JBoss was selected as the application server. JBoss is easy to use and is currently one of the most popular open-source Java application servers on the market according to a recent Gartner research note [7]. It was also an recommendation from a Cybercom Group employee to use JBoss for this project since it is a commonly used platform in similar projects.

2.3.2 Enterprise JavaBeans

Enterprise JavaBeans (EJB) is a framework for developing distributed objects and components. It is used to develop server-side business logic in enterprise applications. Since the Enterprise Beans are portable and distributable it is a well suited technique for developing scalable and distributed applications [4].

Local interfaces are used for defining which methods in an EJB that should be publically accessible from other local beans or applications. Remote interfaces are used for defining which methods that should be publically accessible from remote applications or other remote enterprise beans [4].

It is also possible to deploy the same enterprise beans on several different servers to provide replication for higher performance and reliability [4].
2.3.3 Stripes Framework

The Stripes framework is a Java presentation framework with focus on requiring little or no configuration to get started with developing web applications [12].

Stripes realise the Model-View-Controller (MVC) design pattern in its framework [12]. This decouples the data (Model), presentation (View) and business logic (Controller) and makes it easier to develop applications [12, 13]. All presentation code (Hyper Test Markup Language (HTML) and XML) is put in Java Server Pages, business logic in Servlets (Stripes Action Beans) and data is fetched from a database or other data sources which may be encapsulated in e.g. Enterprise JavaBeans [12].

2.4 Databases

Databases are used to store and retrieve large amounts of data in a structured manner. Today, practically every computer user makes use of databases in one way or another. Examples of this could be when you withdraw or deposit money of your bank account. Databases are then used to keep track of accounts, balance and all transactions made. Examples of other areas where databases are commonly used are: ticket reservation systems, library systems and inventory systems for retail stores.

2.4.1 Database Management System

Database Management Systems (DBMS) is the term which defines programs that are used to create, manage and maintain databases [14]. These programs are required to use a database. Applications which use a database send queries to the DBMS. The DBMS then return a result which hopefully contains the data the application asked for.

There are many DBMS products available on the market, both commercial and open-source products. The most popular commercial database management systems are Oracle Database products and Microsoft SQL Server [15]. Two popular open source alternatives are MySQL and PostgreSQL.
2.4. Databases

The Oracle Database products are mainly used in large enterprise applications and are known for the high performance they provide in large deployments. Oracle is the industry leader when it comes to database management systems [16].

Microsoft SQL Server is a popular DBMS used in Windows environments. In fact, MS SQL server is one of the top products on the DBMS market on the Windows platform [17].

MySQL is the most popular open source DBMS on the market today. It has commercial enterprise as well as open source licenses and is a popular choice for many developers [18].

PostgreSQL is another open source DBMS which is marketed as an enterprise class database. It is only available under an open source license [19].

In this master thesis project the MySQL DBMS will be used. This product was chosen because it has an open-source license and is easy to get started and work with.

2.4.2 Database Modelling

Modelling a database before it is implemented is a very important step in the database design process. There are conceptual modelling tools available for this task. Some popular choices are the Entity-Relationship (ER) model and the Enhanced-Entity-Relationship (EER) model [14].

Conceptual models are used to design a database and define relationships between entity types. Such a model can be of great help to the database designer in finding anomalies in the database design before it is actually implemented [14].

EER modelling extends ER modelling with important design features such as inheritance, subclassing and superclasses, amongst other things [14].

2.4.3 Managing Hierarchical Tree Structures

Relational databases were originally intended for storing flat data structures, such as one-dimensional arrays of integers. When it comes to hierarchical structures such as tree-structures it is possible to store them in a database as well [20]. Since data in a relational database is stored in a flat list and tree structures are not, tree data must be translated into a
database model where operations on tree entities can be done in a simple and efficient manner. Three techniques for handling hierarchical tree structures in a relational database are described in this section; the adjacency list model, the nested set model and frequent insertion trees.

**Adjacency List Model** The adjacency list model is the simplest way to implement tree structures in a database. In this model each entity has a field for referencing a parent entity. This has the obvious side effects that an entity does not have direct knowledge about its children or siblings, it is only aware of its parents. Even though it is possible to work with this implementation in a database, it will become inefficient when data structures grow large. The complexity of queries will also grow with the size of the data structure. Each level of depth in the data structure will require a new sub query (or nested join) to work [20].

**Nested Set Model** A model which is better fitted for storing hierarchical data in databases is called the nested set model. As the name suggests the entities are modelled as sets and subsets. Each parent node is translated to a set. All children of a node are translated to subsets of the parent set. Figure 2.2 presents an example where two cities in Denmark and Sweden respectively have been put in a hierarchical tree. The corresponding sets for the nested set model are illustrated in Figure 2.3 [20, 21].

When working with the nested set model a method called the pre-order tree traversal algorithm is used. All nodes (sets and subsets) are numbered
in ascending order with a left and right value from left to right. This numbering is used to keep track of nodes in the hierarchical tree. The magic in this is that when you look at the left and right values of a node, all its children exist between those values [21]. Figures 2.2 and 2.3 have this numbering printed on the left and right side of each node or set.

An inconvenience arises with the nested set model when frequent updates or inserts are done. Each insert or update of data will lock the table during the time it takes to complete the operation which can result in a big overhead. When updates of the table are not very frequent this will not be a problem but there are techniques for handling this issue as well [20].

**Frequent Insertion Trees**  The problem with overhead that the nested set model can impose may be solved by “creating space” between nodes in a nested set model. The left and right values that are used for constructing the tree do not need to be directly consecutive to the adjacent node values. If a gap is applied between all nodes’ left and right values, an update of an already existing node or insertion of a new node in the tree will not require updates of all nodes left or right values, provided that the gaps between left and right values are large enough [20].

How do we make sure the gaps are large enough? With large data structures even the full range of 32-bit integers can easily be filled if we add gaps between all nodes. Floating-point numbers cannot be used due to precision loss when working with very large (or small) numbers. One of the solutions that Joe Celko discusses in his book is to use rational numbers
for the left and right values [20]. Then the upper boundary for left and right values would be very hard to reach as there always exists a value between two others. This method will create some computational overhead due to the computations that will need to be made for each left and right value, but hopefully this overhead is small compared to updating the left and right values of all nodes in a tree [20].

2.4.4 Normalization

Normalization is a notion used to describe the efficiency of a database. It is a process which can be used to analyze a relation to minimize redundancy, update anomalies and reduce data size. The normalization criteria are described by a number of different normal forms. Some common ones are normal form 1 through 3 and the Boyce-Codd normal form. Following the rules of normalization alone when designing a database does not guarantee a well-designed database. Other more informal requirements may have to be taken into consideration such as minimizing the number of NULL values in tuples. However there can exist cases when normalization to a higher normal form is not the most efficient solution and a denormalization is required [14, 22, 23]. An example of this is described in Section 2.4.5. It is said that database designers should try to achieve NF3 or BCNF to have a good database design [14]. NF1, NF2, NF3 and BCNF are briefly described below.

**Functional Dependency**  The term functional dependency is an essential part of the normalization theory. Functional dependency can be defined as a relation between attributes where a set $X$ uniquely defines a set $Y$. This means for each combination of values in $X$ there exist exactly one value in $Y$. If this criterion holds, then we say that $Y$ is functionally dependent on $X$ [23].

**Normal Form 1**  NF1 forbids multi-value attributes. When a relation contains no multi-value attributes it is said that it is in NF1 [22, 23]. Tables 2.2 and 2.3 show examples of an unnormalized database and a database in NF1, respectively.
2.4. Databases

<table>
<thead>
<tr>
<th>Employee table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee Id</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

Table 2.2: Example of an unnormalized database

<table>
<thead>
<tr>
<th>Employee table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee Id</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

Table 2.3: Example of a database in NF1

**Normal Form 2** The second normal form implies the first and adds a constraint which says that any attribute which is not a key attribute may be functionally dependent on a part of any candidate key [22]. An example of an NF1 database can be seen in Table 2.3. Table 2.4 shows the normalized database where attributes has been split into two tables. Now we no longer have any functional dependency from a candidate key on a non-key attribute.

**Normal Form 3** The third normal form states that there should not exist any functionally dependent non-key attributes from other non-key attributes in the relation [14, 22]. In practice this means that if there exists any dependency within a relation that does not involve a primary key, then the relation should be split. An example of a NF3 database can be seen in Table 2.5.

**Boyce-Codd Normal Form** The Boyce-Codd normal form requires that in all nontrivial functional dependencies $X$ to $Y$, that $X$ is a superkey. BCNF implies NF3 but is more strict. Most relations that are in NF3 are also in BCNF but a relation may be in NF3 and not in BCNF [22]. No
### Employee table

<table>
<thead>
<tr>
<th>Employee Id</th>
<th>Employee Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
</tr>
<tr>
<td>2</td>
<td>Sarah</td>
</tr>
</tbody>
</table>

### Works at table

<table>
<thead>
<tr>
<th>Employee Id</th>
<th>Department Id</th>
<th>Department Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>Sales</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>HR</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>Development</td>
</tr>
</tbody>
</table>

Table 2.4: Example of a database in NF2

### Employee table

<table>
<thead>
<tr>
<th>Employee Id</th>
<th>Employee Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
</tr>
<tr>
<td>2</td>
<td>Sarah</td>
</tr>
</tbody>
</table>

### Works at table

<table>
<thead>
<tr>
<th>Employee Id</th>
<th>Department Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

### Departments table

<table>
<thead>
<tr>
<th>Department Id</th>
<th>Department Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Sales</td>
</tr>
<tr>
<td>2</td>
<td>HR</td>
</tr>
<tr>
<td>3</td>
<td>Development</td>
</tr>
</tbody>
</table>

Table 2.5: Example of a database in NF3
example of BCNF will be provided here as NF3 was decided to be sufficient for this project’s implementation. This is further discussed in Section 3.7.

2.4.5 Buckets

When managing large sets of data, performance issues become relevant. Complex queries which work well on small amounts of data may degrade in performance. One solution is to refactor data into new efficient data structures which are optimized for the type of queries which are being used. This can result in a denormalized database [14]. A technique for this is to put data into conceptual buckets which are designed to keep data in a format which increases performance of queries. The original data is left untouched and additional tables which keeps aggregated data values are created. These tables are called buckets. The bucket tables are updated whenever a new record is inserted and will always have updated aggregated values which will allow for high performance real-time statistics to be generated.
Chapter 3

Architecture and Design

The reporting system, which is the main result of the master thesis project, can be described as a multi-tier client-server architecture where clients handle presentation and servers handle all business logic and data management [24]. The communication between servers and between clients and servers is realised using Web Services and EJB. These techniques are further described in Sections 2.2 and 2.3.2. Data management is accomplished using a database (see Section 2.4) and business logic in the servers. The implementation of these parts is described in this chapter.

3.1 Conceptual System Model

The project can be subdivided into subsystems which represent different parts of the system. These are described by the text below and the conceptual view of the system as seen in Figure 3.1.

Client Module and Device Management Stations

The client module is an integrated part in device management stations and act as a bridge between the device management application and the server. The device management application communicates with the client module which in turn communicates with the centralized logging server.
3.2. Architecture

Centralized Logging Server The centralized logging server’s job is to save reports coming in from remote device management stations in a database. These reports would then be saved in the database for future analysis and generation of statistics.

Database The database acts as a common point of communication between the centralized logging server and the statistics and administration subsystems. This is where all data is stored such as logs or user data.

Statistics and Administration The statistics and administration subsystem is a web application which is used to generate and present statistics from the logs stored in a database. This application contains administrative features such as administration of users and stores and generation of statistics.

3.2 Architecture

The architecture serves as a further refinement of the system model presented in Section 3.1. Figure 3.2 shows how the different parts of the system are connected to each other. The two main parts of the reporting system, the centralized logging server and the statistics and administration, are independent of each other. What ties them together is a common database.
backend where all data is stored. They communicate with the database using the Java Persistence Advanced Programmable Interface (API). The design decision to separate the centralized logging server and the statistics and administration was made to increase both scalability and reliability. In a small deployment of the Reporting System, all subsystems can be placed on the same physical server. In large deployments the subsystems can be spread out over several physical servers to allow for increased performance. In [24], Sommerville brings up other advantages with spreading the subsystems over several physical servers (and even on geographically different locations). One is that it increases redundancy and overall reliability of the system. A failure in one subsystem might otherwise propagate and eventually result in an overall system failure.
3.3 Interfaces and Data Types

The reporting system uses a number of common interfaces and data types for communication. The most important communication interfaces are those used in communication between the different modules of the system. The reference client communicates with the reporting module(s) through a Web Services interface which is described in Section 3.3.1. The client module communicates with the centralized logging server using Enterprise Java Beans and remote procedure calls. This is outlined in Section 3.3.2. Stripes action beans also use Enterprise Java Beans which serve as an abstraction layer for the database in the statistics and administration system. These interfaces are described in Section 3.3.3.

3.3.1 OMSI Reporting

In Section 2.1.1 the Web Services reporting interface, which the OMSI standard defines, is presented. This interface is used in communication between the device management application and the client reporting module.

During the work of the master thesis project it became clear that the reporting interface did not have all the necessary fields to distinguish phone models from different manufacturers. An enhanced interface was proposed which contained an additional field for manufacturer name. This modification to the reporting specification will hopefully be introduced in the next version of the OMSI standard and is the one used in the current version of the Reference Client. The new reporting interface is presented in Table 3.1.

3.3.2 Centralized Logging

The client module communicates with the centralized logging server using the interface presented in Table 3.2. The pUsername and pPassword fields in the storePhoneReport interface (Table 3.2) are used for authentication as described in Section 3.8.1. Enterprise Java Beans requires that all parameters are serializable on remote procedure calls [4]. Therefore, a serialized version of the PhoneUpdateReport data type (Table 3.1) had to be used when communicating with the centralized logging server.
<table>
<thead>
<tr>
<th><strong>Element</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>String</td>
<td>Username field from the OMSI DM application.</td>
</tr>
<tr>
<td>phoneIdentifier</td>
<td>PhoneIdentifier</td>
<td>Phone identifier field, eg. IMEI-number.</td>
</tr>
<tr>
<td>manufacturer</td>
<td>String</td>
<td>Name of the manufacturer.</td>
</tr>
<tr>
<td>modelName</td>
<td>String</td>
<td>Model name.</td>
</tr>
<tr>
<td>dateTime</td>
<td>DateTime</td>
<td>Date and time of the performed update.</td>
</tr>
<tr>
<td>reportedError</td>
<td>String</td>
<td>Error message from the DM application</td>
</tr>
<tr>
<td>warranty</td>
<td>int</td>
<td>Field for specifying whether it was a warranty service or not</td>
</tr>
<tr>
<td>selfTestResult</td>
<td>int</td>
<td>Test result from mobile device</td>
</tr>
<tr>
<td>fullSelfTestResult</td>
<td>base64Binary</td>
<td>Full test result from a mobile device</td>
</tr>
<tr>
<td>refurbish</td>
<td>boolean</td>
<td>Indicates whether the device was refurbished or not</td>
</tr>
<tr>
<td>backup</td>
<td>boolean</td>
<td>Indicates whether a backup was made or not</td>
</tr>
<tr>
<td>updateOK</td>
<td>boolean</td>
<td>Indicates success of the update</td>
</tr>
<tr>
<td>restoreOK</td>
<td>boolean</td>
<td>Indicates success of the restore</td>
</tr>
<tr>
<td>oldFirmwareVersion</td>
<td>FirmwareVersion</td>
<td>Version information of the old mobile device firmware</td>
</tr>
<tr>
<td>newFirmwareVersion</td>
<td>FirmwareVersion</td>
<td>Version information of the new mobile device firmware</td>
</tr>
</tbody>
</table>

Table 3.1: Enhanced PhoneUpdateReport specification
3.3. Interfaces and Data Types

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pUsername</td>
<td>String</td>
<td>Username to authenticate with.</td>
</tr>
<tr>
<td>pPassword</td>
<td>String</td>
<td>Password to authenticate with.</td>
</tr>
<tr>
<td>pReport</td>
<td>S_PhoneUpdateReport</td>
<td>Serialized PhoneUpdateReport.</td>
</tr>
</tbody>
</table>

Table 3.2: Remote logging interface storePhoneReport

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleteBucketDataByTerritory</td>
<td>Deletes bucket data associated to a territory.</td>
</tr>
<tr>
<td>deleteLogsByStore</td>
<td>Delete logs by store.</td>
</tr>
<tr>
<td>getLogChart</td>
<td>Get chart data based on the logs in the database.</td>
</tr>
<tr>
<td>getManufacturers</td>
<td>Get a list of manufacturers.</td>
</tr>
<tr>
<td>getPhonemodels</td>
<td>Get a list of phone models.</td>
</tr>
<tr>
<td>storePhoneReport</td>
<td>Stores a phone report on the server.</td>
</tr>
</tbody>
</table>

Table 3.3: LogManager interface

3.3.3 Statistics and Administration

The Stripes action beans used in the statistics and administration subsystem use Enterprise JavaBeans to access the database. Four EJBs act as abstraction layer towards the database providing public accessible functions through local interfaces; LogManager, StoreManager, TerritoryManager and UserManager. Each of these interfaces provides access to different parts of the database.

The LogManager interface contains methods related to logs stored in the system. A full method summary can be seen in Table 3.3. Similar interfaces for the StoreManager, TerritoryManager and UserManager which contain methods for accessing store-, territory- and user data, respectively, are also available and are presented in Tables 3.4, 3.5 and 3.6.
### Method Description

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addStore</td>
<td>Add a store.</td>
</tr>
<tr>
<td>deleteStore</td>
<td>Delete a store and its associated logs.</td>
</tr>
<tr>
<td>getAllStores</td>
<td>Get all stores.</td>
</tr>
<tr>
<td>getStoreById</td>
<td>Get a store by its id.</td>
</tr>
<tr>
<td>getStoreByUsername</td>
<td>Get a store by its associated username.</td>
</tr>
<tr>
<td>getStoresByTerritory</td>
<td>Get all stores by their associated territory.</td>
</tr>
<tr>
<td>saveStore</td>
<td>Edit a store.</td>
</tr>
</tbody>
</table>

Table 3.4: StoreManager interface

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addTerritory</td>
<td>Adds a territory.</td>
</tr>
<tr>
<td>deleteTerritory</td>
<td>Delete a territory.</td>
</tr>
<tr>
<td>editTerritory</td>
<td>Edits a territory.</td>
</tr>
<tr>
<td>getAllTerritories</td>
<td>Returns a sorted list of all territories.</td>
</tr>
<tr>
<td>getAllTerritoriesIndented</td>
<td>Returns a sorted list of all territories indented by specified string.</td>
</tr>
<tr>
<td>getParentTerritories</td>
<td>Returns a list of parents to the selected territory.</td>
</tr>
<tr>
<td>getTerritories</td>
<td>Retrieve territories defined by its ids.</td>
</tr>
<tr>
<td>getTerritory</td>
<td>Retrieve a territory.</td>
</tr>
<tr>
<td>getTerritoryStores</td>
<td>Retrieve all store territories in a given territory.</td>
</tr>
</tbody>
</table>

Table 3.5: TerritoryManager interface
<table>
<thead>
<tr>
<th><strong>Method</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>addRole</td>
<td>Add a role.</td>
</tr>
<tr>
<td>addUser</td>
<td>Add a user.</td>
</tr>
<tr>
<td>deleteRole</td>
<td>Delete a role.</td>
</tr>
<tr>
<td>deleteUser</td>
<td>Delete a user.</td>
</tr>
<tr>
<td>editUser</td>
<td>Edit a user.</td>
</tr>
<tr>
<td>encryptPassword</td>
<td>Encrypts a password with a salt.</td>
</tr>
<tr>
<td>generateSalt</td>
<td>Generates a salt to be used with password encryption.</td>
</tr>
<tr>
<td>getAllUsersAndRoles</td>
<td>Retrieves a list of all users and the roles that they belong to.</td>
</tr>
<tr>
<td>getRole</td>
<td>Get a role.</td>
</tr>
<tr>
<td>getRoleByName</td>
<td>Get a role by its name.</td>
</tr>
<tr>
<td>getRolePermissions</td>
<td>Get the permissions for a role.</td>
</tr>
<tr>
<td>getRoles</td>
<td>Get all roles.</td>
</tr>
<tr>
<td>getRolesByTerritory</td>
<td>Get all roles by territory they have permission on.</td>
</tr>
<tr>
<td>getUserById</td>
<td>Get a user by its id.</td>
</tr>
<tr>
<td>getUserByUsername</td>
<td>Get a user by its username.</td>
</tr>
<tr>
<td>getUsersByRole</td>
<td>Gets a list of users associated with a given role.</td>
</tr>
<tr>
<td>refreshUserPermissions</td>
<td>Refresh the privileges of a user.</td>
</tr>
<tr>
<td>saveRolePermission</td>
<td>Saves a permission for a set role.</td>
</tr>
<tr>
<td>setRole</td>
<td>Change a role, the id in the Role object should be the id of the role that should be changed.</td>
</tr>
<tr>
<td>validateUser</td>
<td>Validates (authenticate) a user.</td>
</tr>
</tbody>
</table>

Table 3.6: UserManager interface
3.4 Client Module

The client module implementation was based on a reference module implementation provided by OMSI Forum. This reference module was originally intended for vendors who wished to create their own device management plug-in module to the OMSI Reference Client. It was modified to use the reporting interface and provide logging functionality to the OMSI Reference Client.

Instead of limiting the work to creating a remote reporting module, a concept of a generic plug-in system for logging was created. This plug-in system implemented the reporting interface and allowed for different types of reporting modules to be used such as remote reporting and local file reporting. This concept was adapted to the OMSI Reference Client by Cybercom.

When this plug-in system was in place, creating a reporting module was as easy as creating an external “jar”-archive and implement a predefined Java interface to connect a reporting module to the OMSI Reference Client.

The client module receives reports from the OMSI Reference Client through Web Services. The reports in the client module are stored in a buffer until a successful connection to the centralized logging server can be made. After that a connection is established the report is sent to the centralized logging server using a remote EJB interface.

3.5 Centralized Logging Server

The centralized logging server relies on a database which handles logs and device management station information. The server endpoint consists of an Enterprise JavaBean which is accessed through a remote interface which is further described in Section 3.3.2.

Logs which are sent to the centralized logging server are stored in a database. When a log entry reaches its destination, the client module is first authenticated by matching the store’s login information with information in the database. Then manufacturer and phone model information is extracted from the log entry and compared to already existing entries in the database to avoid duplicate entries of this data. After this has been
Figure 3.3: Sequence diagram of the centralized logging server
Figure 3.4: Overview of the statistics and administration subsystem

The statistics and administration subsystem successfully completed the log is committed to the database. The flow of data is visualized in Figure 3.3.

3.6 Statistics and Administration

The statistics and administration subsystem is implemented using the Stripes Framework, EJB and a database. A detailed overview of this subsystem can be seen in Figure 3.4. The statistics and administration subsystem contains functionality to manage (create, read, update or delete) users, roles, stores and territories. It also provides the ability to set permissions for roles and to generate statistics in charts.

The Stripes Framework is used to implement a client web application. In Stripes all logic is put in action beans where also access control, data management, computations and other things reside. Each action bean has several methods where each method represents different actions that a user can perform in the application. When a user requests a function in the system, the request is automatically mapped to a method in an action bean by Stripes. The action bean also decides what should be presented to the user by selecting a view (an HTML template) and then injects data into the view.

The data which is stored in a database is accessed through EJBs. The EJBs act as an abstraction layer towards the database. When an action bean needs information from the database a method call is made on a local enterprise bean. The EJB will create database queries to retrieve the information that the action bean requested.
if (!getContext().getUser().isAllowed(1, Permission.ADMIN_USER_CREATE)) {
    return new RedirectResolution("/error/403a.jsp");
}

Listing 3.1: Example of access control

Access control is used to allow or deny users from accessing functions in the statistics and administration subsystem. Each action bean method is responsible for checking if a user has permission to access the method. An example of this can be seen in Listing 3.1 where it is checked whether the current user is allowed to create a new user account. The access control feature is further discussed in Section 3.8.2.

3.7 Database

In this section the database implementation is described. First the database model is presented and then each part of the database implementation is described.

3.7.1 Modeling

An EER model of the database was created which can be seen in Figure 3.5. It was used as starting point when implementing the database in MySQL. The goal was to make a database which was in BCNF as described in Section 2.4.4. Most databases which are in NF3 are also in BCNF, so no effort will be put into verifying BCNF in this project. By making the database NF3 the overhead and anomalies from removed data is minimized, whereas BCNF guarantees that there are no anomalies in the database.

Unfortunately some tables had to be denormalized due to performance issues with the NF3 implementation when generating statistics. The implementation of statistics is further discussed in Section 3.7.5. The performance issues appeared when measuring performance of the application and are described in Section 4.2.3.
Figure 3.5: EER model of the database
3.7.2 Log, Phonemodel and Manufacturer tables

The Log, Phonemodel and Manufacturer tables are tightly coupled and together represent the full view of a log entry. To achieve BCNF as described in Section 2.4.4 the phone model and manufacturer attributes information are put in their own tables. New foreign keys are introduced for the Log to Phonemodel “Many-to-One” (M:1) association and Phonemodel to Manufacturer M:1 association. Logs are also related to stores by an M:1 association. This association is used to see where logs originate from and is realised by using a foreign key between the Log and Store tables.

3.7.3 Store and Territory tables

DM stations information such as login credentials and store name are saved in a table. The DM stations are also organized under territories. This relationship is identified by an M:1 association and realized by a foreign key in the Store table to the Territory table.

The territory table is designed to hold hierarchical data as discussed in Section 2.4.3. The implementation is a mixture of the nested set model and the adjacency list model. The nested set model requires the left (lft) and right (rgt) values for building a hierarchical tree and the adjacency list model uses a reference to a parent node (parent_id). The reason for using both these models is for convenience reasons only. It can at some times, such as when working with single nodes in the tree and not the complete tree, be very useful to be able to know which the parent of a node is.

In Chapter 2 the frequent insertion trees model is presented as a solution to some performance issues that can arise when updating the tree structure frequently. In this application the territory tree structure will rarely be updated, therefore it was decided not to implement the frequent insertion trees model.

3.7.4 User, Role and Role_Acl tables

User accounts in the statistics and administration system are stored in the User table. All users are associated with a role on which permissions are set. Many users may share a role meaning that the same users will also
share permissions. Access control and permissions are discussed in Section 3.8.2. Roles are stored in the Role table which has a “One-to-Many” (1:M) association with the User table and a “Many-to-Many” (M:N) association with the Territory table. The M:N association with territories introduces an additional attribute to store permissions called permission_id and is realized in a table called Role_Acl.

3.7.5 Bucket tables

To take on the performance issues which appear when generating statistics from large data sets, bucket tables were created. The process of measuring performance of statistics generation is described in Section 4.2.3. The technique to use Buckets is described in Chapter 2.

Two subsets of Bucket tables were created, one for keeping aggregated statistics on a daily basis, Bucket_Total, and one for covering the daily distribution of updates carried out, Bucket_Histogram. Both bucket tables are updated whenever a new report is added to the system to avoid inconsistencies of aggregated data.

3.8 Security

When working with the implementation some decisions concerning system security were taken. Access to the centralized logging server and statistics and administration subsystem had to be restricted to prevent unauthorized usage.

3.8.1 Centralized Logging

The centralized logging is designed to be used by a number of device management stations where each station will need to identify itself when submitting a report. There are a number of different techniques which can be used to prevent the system from unauthorized access.

Transport encryption  It is possible to make EJB or Web Services communicate over an encrypted connection using SSL/TLS. This sets up a se-
cure transport between client and server endpoints which encrypts all data that is sent over the connection. Such an approach requires some configuration on both server and client applications but makes all communication secure and can be useful when using a public network like Internet for data transport [4, 8, 25].

Limiting access When using a closed network such as a VPN, corporate network or other private networks, only device management stations on the same network can access the logging server. When using a public network like the Internet, allowed client IP addresses could be set in the server to restrict access to the centralized logging server [25].

Authentication Another security measure that can be taken is to require each device management station to authenticate itself, using its user credentials on the centralized logging server when sending in a report. Then the centralized logging server can decide whether or not to allow a certain client to send reports.

Authentication is the only security measure that has been implemented in the centralized logging system. It was decided that the security it provides is sufficient for the system at this point. Further security measures can be implemented when it is considered to be needed.

3.8.2 Statistics and Administration

The security measures in the statistics and administration subsystem consist of user authentication and user permissions. To access statistics and other administrative features of the system a user needs an account. User accounts are created by users with certain administrative rights.

Features in the statistics presentation system require different permissions. These permissions are set up in a hierarchical manner where all stores are organized under territories. This can be visualized as a tree structure as described in Section 2.4.3, where a leaf in the tree represents a store. This gives us the flexibility that user permissions can be set at either global level or limited to territories or even individual stores. This allows us to create user permissions so that a user is allowed or denied to access statistics or administrative features for any store in the system.
To further increase the flexibility of user management, users inherit permissions from a role. Multiple users can be assigned to a role and permissions be set for this role which results in permissions being inherited down to those users.
The purpose of doing measurements on the system is to determine how well the system performs over time. Real measurements will be carried out together with simulations on the system.

4.1 EJB versus Web Services

A performance comparison between the two techniques Enterprise Java Beans and Web Services were performed to evaluate the performance for logging results to the reporting server. This was done in an early stage to help in making a decision on which technique to use throughout the system. The assumption made was that only small differences in performance were going to be seen since both techniques involve transporting data over the network in some form of intermediate format [4, 5].

The analysis was performed using a single client which communicated with a local server using one of the techniques at a time. The client reported 100 consecutive logs at a time. The procedure was repeated twelve times to get a data set with as little faults as possible. The first set with both techniques showed inconsistent values. The reason for this is that it takes
4.1. EJB versus Web Services

Figure 4.1: Performance comparison between EJB and Web Services

some time for server side initialization. The full results are presented in Figure 4.1.

The result from the analysis showed that using Enterprise Java Beans were often over 50% faster than Web Services. This information together with the facts presented in Section 2.3.2 showing that EJB is easy to implement and use, was enough to make a decision on which technique to use in the project.

Web Services offer interoperability between different programming languages, however in this project the client module is closely coupled with the server. The server and client module are together stand-alone and not dependent on any specific device management application. The only requirement is that the logging interface specification for reporting is used to communicate with the client module.
4.2 Measuring the Application

To get a better idea on the actual performance and requirements of the system, some measurements have to be made. The results of these measurements are presented in this section.

4.2.1 Data Size

The amount of hard disk space the data stored in the database takes up may be necessary to know when planning for deployment of an application.

The average row size measured on a database filled with data gives an estimate on how much disk space that will be required as the database is populated with new reports. Since the Log table will hold the largest amount of records, one for each report, this will be the place which will take up most space in the database.

The database was populated with data using statistics from an OMSI Pilot Project. The average row sizes for tables in the database is presented in Table 4.1.

Using the rough estimate that an average of 100 updates are done in a store each month we can generate a graph showing the total number of

<table>
<thead>
<tr>
<th>Table</th>
<th>Average row size (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bucket_Histogram</td>
<td>263</td>
</tr>
<tr>
<td>Bucket_Total</td>
<td>308</td>
</tr>
<tr>
<td>Log</td>
<td>145</td>
</tr>
<tr>
<td>Manufacturer</td>
<td>1820</td>
</tr>
<tr>
<td>Phonemodel</td>
<td>186</td>
</tr>
<tr>
<td>Role</td>
<td>2340</td>
</tr>
<tr>
<td>Role_Acl</td>
<td>199</td>
</tr>
<tr>
<td>Store</td>
<td>963</td>
</tr>
<tr>
<td>Territory</td>
<td>655</td>
</tr>
<tr>
<td>User</td>
<td>4096</td>
</tr>
</tbody>
</table>

Table 4.1: Average row size in database tables
4.2. Measuring the Application

logs that have been saved in the database. A graph displaying the amount of reports when 50 stores are using the system can be seen in Figure 4.2.

When assuming that each report is about 145 bytes in size, twelve months worth of reports by 50 stores will occupy approximately 8.3MB of hard disk space. This number will then be easy to use when estimating the hard disk space required in larger deployments. As an example would a deployment with 1.000 stores running for three years require about 500MB of hard disk space. This amount of data is small compared to what is available on todays hard drives and should therefore not require any special planning for deployment.

4.2.2 Reporting Performance

One of the major parts of the system that has been created is the reporting subsystem. It is essential to know how the system performs when more clients are added. To figure out how well the system performs when

Figure 4.2: Number of reports when 50 stores are using the system
Figure 4.3: Average service time (μS) in reporting performance measurements, JBoss time values are measured inside the actual application and total values are measured on the client side.

Figure 4.4: Total service time (μS) in reporting performance measurements, JBoss time values are measured inside the actual application and total values are measured on the client side.
38  4.2. Measuring the Application

<table>
<thead>
<tr>
<th>Windows 2003 Server</th>
<th>Windows XP Professional SP3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xeon 2.33Ghz</td>
<td>C2D 2.13 Ghz</td>
</tr>
<tr>
<td>512 Mb RAM</td>
<td>2 Gb RAM</td>
</tr>
<tr>
<td>JBoss 4.2.3 GA</td>
<td>JBoss 4.2.3 GA</td>
</tr>
<tr>
<td>MySQL 5.0.67</td>
<td>MySQL 5.0.67</td>
</tr>
</tbody>
</table>

Table 4.2: Server setups used in measuring of logging performance

thousands of simultaneous clients are accessing the system, it needs to be modelled and simulated since this is virtually impossible to test in a lab environment. During the development process the measurements and simulation results will help in finding bottlenecks in the system. At deployment of the system the simulation models can also be used to determine requirements for hardware and software the system will run on.

The measurements were made using two types of client programs. One for putting load on the server to simulate multiple clients sending logs to the server. The second program type was used to measure times when storing a fixed amount of 20.000 logs on the server. Two different server setups were used to measure results. These are presented in Table 4.2. The hardware of the two servers are quite different and is expected to produce different result data. Using two systems which are this different will help us get an understanding of how the system works on different hardware setups and see if the systems behave differently under different loads.

One major difference which can be observed in the results presented in Figures 4.3 and 4.4 is the time a client spends on setting up a connection to the server. When using Windows XP Professional as the server platform, the time spent on connections is about five times as high as when the server platform is Windows 2003 Server. One could assume that there are differences in how the two operating systems handle server connections. Windows 2003 Server has higher network performance and is better fitted to work with server applications. This means that log requests will not arrive at the server as frequent on Windows XP Professional as on Windows 2003 Server. The hardware on which Windows XP Professional is running on is also superior to the one Windows 2003 Server is running on.
The default setting in JBoss for the number of concurrent threads is 10 and the maximum queue size is 1000. The number of concurrent threads setting directly influences the results seen in Figures 4.3 and 4.4. With ten or less concurrent client connections the average and total time for jobs are more or less the same. When more than ten concurrent clients are using the system we get a linear increase in service time. This behaviour is seen in both server setups used in the tests.

### 4.2.3 Statistics Performance

The most complex and computationally heavy operation in the statistics and administration subsystem is to generate charts from statistics of reports stored in the database. The first prototype of the statistics and administration subsystem did not implement any “smart” technique for optimizing generation of statistics. When using performance tests on large data sets (millions of reports), the query execution time rose to minutes instead of seconds. It became evident that something had to be done to increase performance of queries. In the following section the method and result from measuring performance are presented.

An optimization technique for working with large sets of data is to create buckets which hold aggregated statistical data. This technique is described in Section 2.4.5. After implementing buckets some performance measurements were performed using data sets of different sizes to see how much performance had improved.

```sql
SELECT SQL_NO_CACHE count(*)
FROM log l, store s
WHERE s.id=l.store_id AND
(l.dateTime BETWEEN "2008-01-01" AND "2008-12-31") AND
(s.territory IN (252));
```

Listing 4.1: Query used to measure performance without buckets

Listing 4.1 and 4.2 shows the queries which were used to perform the measurements using no buckets or buckets respectively. Caching was turned off in both cases to measure actual query times. If caching would have been
4.2. Measuring the Application

Figure 4.5: Graph showing measured query time when using buckets

left on, some measured query times would be very low since the query result could be fetched from the cache.

```sql
SELECT SQL_NO_CACHE sum(amount)
FROM bucket_total b
WHERE
    (date BETWEEN "2008-01-01" AND "2008-12-31") AND
    (territory IN (252));
```

Listing 4.2: Query used to measure performance using buckets

Figure 4.5 shows the execution time on different data set sizes. When not using buckets we get a linear increase in execution time, however when using buckets, all queries operate at constant time. The performance gain that we see using buckets for optimization is big and solves the problems that were experienced when working with large data sets.
Chapter 5

Discussion

This master thesis project provides a solution which makes collecting statistics from device management stations more efficient, requiring little or no human resources. As the burden to collect logs can be a heavy work task even in a small deployment of device management stations, imagine what it would be in a large deployment with thousands of device management stations.

The performance measurements showed that the reporting system would be able to serve hundreds of device management stations with a single server. The reporting system can be scaled up to handle more clients (device management stations) by deploying the different subsystems at different physical servers. Since the centralized logging server and the statistics and administration are stand-alone EJBs, it is also possible to replicate these enterprise beans to several physical servers for increased scalability and reliability. Even the database can be replicated over several servers if needed in a large deployment.

Users of the statistics and administration subsystem are able to perform basic tasks like user and store management or generate statistics. This subsystem has the advantage that many users can access information in an easy manner through a web browser. All data is collected in one place where only those who should have access to the information, have access to the information. The statistics and administration subsystem can
also be extended or replaced if new features are required since the data in
the database is separated from the implementation. Access control to all
features in the system can be controlled in a simplistic way by an admin-
istrator. Due to how the access control system is designed, users can be
granted or denied access to stores and features in the system on both an
individual basis or on territories, to cover several stores.

5.1 Conclusions

The gains of using a software system for tasks that can be automated can
be big. In this master thesis project one such application where a software
system could be used to make a work task more effective is presented.
Using a similar design to what has been presented here, other areas where
data is collected for analysis can be rationalized using a software system
for reporting. This master thesis project can serve as a proof-of-concept
for implementing a remote reporting system using the techniques described
throughout the report.

5.2 Future Work

Other implementations or further development of the statistics and admin-
istration system could be created depending on which features customers
want from the product. Using well documented techniques and frameworks
such as the Stripes Framework, it is easy for a developer to pick up and
continue developing the system.

One downside with the reporting system arises when any of the at-
tributes in the OMSI Reporting specification are added or removed. The
reporting system must then be adapted to match the specification. To
avoid this inconvenience and perhaps make the reporting system compat-
ible with other applications, one future work task could be to make this
reporting system generic. The reporting system could then be configured
to work with different deployments of device management applications or
even other systems that require remote logging.
## Appendix A

### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Advanced Programming Interface</td>
</tr>
<tr>
<td>BCNF</td>
<td>Boyce-Codd Normal Form</td>
</tr>
<tr>
<td>DBMS</td>
<td>DataBase Management System</td>
</tr>
<tr>
<td>DM</td>
<td>Device Management</td>
</tr>
<tr>
<td>NFx</td>
<td>Normal Form 1-3</td>
</tr>
<tr>
<td>OMSI</td>
<td>Open Mobile Service Interface</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>

Table A.1: List of abbreviations
Appendix B

Guides

This chapter contains guides for how to get started with the system.

B.1 Installation of Client Module

This section describes how to install the client module in the reference client environment.

B.1.1 Preparation

To install the client module a clean OMSI Reference Client distribution is required along with the client module “jar” file, the configuration XML file and some additional library “jar” files. A complete listing of client module files are listed in Table B.1.

B.1.2 Copying Files

The client module files need to be placed in the appropriate directories for the module to work.

All files described in Table B.1 need to be copied to the corresponding directory in the OMSI Reference Client. The reporting/remotereporting-module.jar file should go into the OmsiClient/reporting folder,
## B.1. Installation of Client Module

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reporting/remotereportingmodule.jar</td>
<td>The client module file</td>
</tr>
<tr>
<td>omsi_reporting_remote_ejb.xml</td>
<td>Configuration file</td>
</tr>
<tr>
<td>lib/jbossall-client.jar</td>
<td>JBoss client library file</td>
</tr>
<tr>
<td>lib/ReportingEjbClient.jar</td>
<td>EJB client library file</td>
</tr>
</tbody>
</table>

### Table B.1: Client module file listing

`omsi_reporting_remote_ejb.xml` into the `OmsiClient` root folder, `lib/jbossall-client.jar` and `lib/ReportingEjbClient.jar` into the `OmsiClient/lib` folder. Now all files are in place.

### B.1.3 Configuration

The last part of the installation is to configure the OMSI Reference Client and the client module. The first thing to do is to make the OMSI Reference Client aware of the additional library files that were installed. To do this the OMSI Reference Client’s manifest file need to be edited.

```manifest
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.7.0
Created-By: 10.0-b23 (Sun Microsystems Inc.)
Main-Class: org.omsirefclient.Startup
Class-Path: lib/log4j-1.2.15.jar lib/jbossall-client.jar lib/ReportingEjbClient.jar
```

### Listing B.1: OMSI reference client manifest file

Open the `OmsiClient/omsi-client.jar` file with a compression utility like WinZip or WinRar. Navigate to `omsi-client.jar/META-INF/MANIFEST.MF` and open the file for editing. At the end of the “Class-Path:” line add “lib/jbossall-client.jar lib/ReportingEjbClient.jar”, when done the line should look something similar to code listing B.1. Save the file and make sure the file in the archive is updated with the changes.
The client module should be updated with connection configuration settings so that the client module can connect to the centralized logging server. The file to be edited is `omsi_reporting_remote_ejb.xml`. This file can be seen in Listing B.2. The only entries in this configuration file that should need to be changed is `serverURL`, `username` and `password`.

The `serverURL` entry should point to the JBoss JNP used to look up the Enterprise Java Bean. The service entry holds the internal name in JBoss for the reporting bean, this should probably not have been changed. A username and password should be defined for each device management station, the exact same username and password must exist in the Centralized Logging Server database for logging to work.

### B.2 Installation of Database

The database is used to store all user information, store information, log data and statistics. The installation of the database is covered in three steps; Preparation, Configuration and Loading database dump.
B.2. Installation of Database

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema.sql</td>
<td>Holds a dump of the database schema</td>
</tr>
<tr>
<td>data.sql</td>
<td>Holds a dump of the initial data</td>
</tr>
</tbody>
</table>

Table B.2: Database file listing

B.2.1 Preparation

The system has been tested on MySQL Database Server 5.0.67 using InnoDB as the table engine. Instructions for how to install MySQL Database Server can be found at MySQL AB’s web site [18]. The files used in the database installation procedure are presented in table B.2.

B.2.2 Configuration

The omsi reporting system requires a database user and a database. How to create databases and users and setup permissions are not covered here, read the MySQL manual available at MySQL AB’s web site for information on how to do this [18]. The user credentials and the database name will be required when configuring the Centralized Logging server in Section B.3.2 and the Statistics and Administration system in Section B.4.2.

B.2.3 Loading Database Dump

The last part of the database installation is to create the tables and to populate the database with initial data. This is done by loading two files into the database, one which contains the table structure and one which contains the data. This can be done in many ways, of which one will be explained here. The commands that should be executed are presented in code listing B.3.

```
mysql -u username -p database_name < schema.sql
mysql -u username -p database_name < data.sql
```

Listing B.3: Commands for load database dump
The commands should be executed on the system where the MySQL server is installed. Replace *username* and *database_name* with the actual user credentials and database name respectively. Enter a password when prompted to. When this has been completed, the database should be properly configured and ready to use.

### B.3 Installation of Centralized Logging Server

The Centralized Logging server itself is a stand-alone Java Enterprise Bean packaged in an “ear” package together with the necessary utility classes and interfaces. The installation consists of three consecutive tasks, preparations, configuration and deployment.

#### B.3.1 Preparation

The Centralized Logging server has been successfully deployed and tested on both Windows Server 2003 and Windows XP Pro SP3 installation with JBoss 4.2.3 GA application server and MySQL Connector/J 5.1.6. The Centralized Logging server requires a database connection to a MySQL database and that JBoss 4.2.3 GA is available on the server system. The installation procedure of the OMSI database is outlined in Section B.2. Instructions for how to install JBoss can be located at the JBoss web site [8]. A complete listing of files that is used in the installation of the centralized logging server can be seen in Table B.3.

#### B.3.2 Configuration

For database connectivity to be available in JBoss 4.2.3, a JDBC driver needs to be added. The database provider used in this system is MySQL,
B.3. INSTALLATION OF CENTRALIZED LOGGING SERVER

and therefore the corresponding JDBC driver needs to be downloaded and installed in JBoss. MySQL AB provides this driver from their web site [18]. When the driver has been downloaded it should be placed in the folder server/default/lib relative to the JBoss installation folder.

The database connection configuration is done in reportdatabase-ds.xml. This file can be seen in Listing B.4. The only entries that should need to be changed here is connection-url, user-name and password.

Listing B.4: Centralized logging server database connectivity configuration

```xml
<datasources>
  <local-tx-datasource>
    <jndi-name>ReportDatabaseDS</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/omsi</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>MYSQL_USERNAME_HERE</user-name>
    <password>MYSQL_PASSWORD_HERE</password>
  </local-tx-datasource>
</datasources>
```

The jndi-name entry sets the JNDI name that the data source should bind to, this should not be necessary to modify. The connection-url entry tells the JDBC driver where the database is located. The default setting is a database named omsi located on localhost (the same machine as the Centralized Logging Server). Driver-class should not be changed either since it specifies which driver class that is used to connect to the database. The user-name and password fields should correspond to a database user with read and write permissions on the omsi database. This user is set up in the database server.

B.3.3 Deployment

To deploy the Centralized Logging server the OMSIReporting.ear and reportdatabase-ds.xml files should be put into the server/default/deploy directory relative to the JBoss installation folder. If everything was configured correctly, the Centralized Logging server is now up and running.
Table B.4: Statistics and administration file listing

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMSIReportingPresentation.ear</td>
<td>The statistics and administration package file</td>
</tr>
<tr>
<td>reportdatabase-ds.xml</td>
<td>Configuration file for database access</td>
</tr>
</tbody>
</table>

B.4 Installation of Statistics and Administration

The Statistics and Administration part of the system is stand-alone and consists of a number of Java Enterprise Beans, Stripes Action Beans and JSP pages packaged in an “ear” package together with the necessary utility classes and interfaces. The installation consists of three consecutive tasks, preparations, configuration and deployment.

B.4.1 Preparation

This part of the system share the same prerequisites as the Centralized Logging server as outlined in Section B.3.1. The files that are used in the installation of this module are presented in Table B.4.

B.4.2 Configuration

The procedure is the same as when configuring the Centralized Logging server. See Section B.3.2.

B.4.3 Deployment

To deploy the Statistics and Administration part of the system the OMSIReportingPresentation.ear and reportdatabase-ds.xml files should be put into the server/default/deploy directory relative to the JBoss installation folder. If everything was configured correctly, the Statistics and Administration part of the system is now up and running.
B.5 Maintenance

If it ever becomes necessary to rebuild the buckets which contain aggregated data for statistics the SQL code presented in Listings B.5 and B.6 can be used.

```sql
TRUNCATE TABLE bucket_total;

INSERT INTO bucket_total
    (dateTime, 'successful', 'territory', 'phonemodel', 'manufacturer')
    SELECT DATE_FORMAT(l.dateTime, "%Y-%m-%d"), l.updateOk, s.territory, p.id, p.manufacturer_id
    FROM log l, phonemodel p, store s
    WHERE p.id=l.phonemodel_id AND s.id=l.store_id
    ON DUPLICATE KEY UPDATE
        'amount'='amount'+1;

Listing B.5: Example of code which rebuilds the “total” bucket

TRUNCATE TABLE bucket_histogram;

INSERT INTO bucket_histogram
    (dateTime, 'successful', 'territory', 'phonemodel', 'manufacturer')
    SELECT TIME_FORMAT(CONCAT(HOUR(dateTime), ':', (FLOOR(MINUTE(dateTime)/15)+15), ':0'), '%H:%i'),
        l.updateOk, s.territory, p.id, p.manufacturer_id
    FROM log l, phonemodel p, store s
    WHERE p.id=l.phonemodel_id AND s.id=l.store_id
    ON DUPLICATE KEY UPDATE
        'amount'='amount'+1;

Listing B.6: Example of code which rebuilds the “histogram” bucket
Appendix C

Source Code and Java Documentation

This appendix chapter has been excluded from this version of the master thesis report.
Bibliography


Copyright

Svenska
Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.
Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådan sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

English
The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Marcus Svensson
Linköping, January 28, 2009