Evaluation of Ericsson Service Enabler SDK (Software Development Kit) s for Mobile Internet Application Development

X I A O Y I N G W A N G

Master of Science Thesis
Stockholm, Sweden 2005

IMIT/LECS-2005-103

Evaluation of Ericsson Service Enabler SDK (Software Development Kit) s for Mobile Internet Application Development

X I A O Y I N G W A N G

Master of Science Thesis
Stockholm, Sweden 2005

IMIT/LECS-2005-103

Student: ***Xiaoying Wang*****, Internetworking Master Program, Department of Microelectronics and Information Technology, Royal Institute of Technology(KTH), Sweden.**

Examiner: ***Vladimir Vlassov*****, Associate Professor, Department of Microelectronics and Information Technology, Royal Institute of Technology (KTH), Sweden.**

Industrial supervisor: ***Peter Yeung*****, Technical Service Group, Mobility World, Ericsson**

Abstract

Internet and Mobile communication technologies have been evolving phenomenally in recent years. Mobile Internet technology, which integrates the traditional Internet applications and telecom network capabilities, has been approved as a new profitable business model for network operators, content providers, application service providers, and application vendors.

Ericsson Mobility World has aimed to provide the Ericsson proprietary SDKs and solutions for Mobile Internet application development. By using these SDKs, developer can develop innovative applications and get access to the key capabilities of the telecom networks, so to provide the end user with new services that are secure, reliable and efficient. The end user can access the Mobile Internet through either traditional web interface or the portable devices such as mobile phone and PDA via WAP/WML.

In order to provide the best mobile Internet technologies among the market, Ericsson Mobility World initiated this thesis project, whose mission is the Evaluation of Ericsson Service Enabler SDKs, in order to have better quality insurance for the SDKs. During the process of evaluation of Diameter Charging SDK, a higher-level API was proposed and subsequently implemented commercially to eliminate the complexity of use and further provide an applicable API structure for integration with EJB through a Resource Adapter. While MPS and Parlay SDK were evaluated, instructional improvements were recommended regarding the SDKs' outlook, usability, and feasibility and so on.

The Mobile Internet is gaining more and more popularity because of the innovative, easy-to-access and easy-to-use services. So this thesis project is also aimed on introduction of the commercial structure of Mobile Internet.

Key words

Mobile Internet, Charging, Diameter, Parlay, NRG, MPS, JAVA, JAXB, J2SE, J2EE

Acknowledgement

First of all, I would like to give the most respectful gratitude to my beloved paternal grandparents, who are very important source of spirit of my life, for the selfless love and invaluable virtue given to me. Without them, I could not have been like today. I wish they could continue being happy and healthy so to enjoy and share my happiness and success.

I would also like to express my sincere gratitude to my thesis supervisor, Associate Professor Vladimir Vlassov of IMIT, KTH, for academically guiding me through the whole thesis process. Great thanks are given to my industrial thesis advisor Peter Yeung, SDK manager in Ericsson Mobility World, for the endless patience and precise technical instruction to my thesis project.

Special thanks are given to Daniel Freeman, manager of Technical Service group in Ericsson Mobility World, for giving me this precious opportunity to carry out the project. I would also like to give my thankfulness to Kristoffer Högren, Tony Vlachos and Peter Skaphagen, who I cooperated with for this project and also other colleagues of Ericsson Mobility World for the supportive cooperation.

A thousand thanks are given to my friend Catherine Mulligan for her bright smile and the enthusiasm given to me by saying “Xiaoying, because you are an excellent Engineer!”

Special appreciations are given to my dear friends Wang Xiaobin, Xiang Hui, Zhang Hong, Zhang Lejun, Li Hui and Wu Junliang for helping me out of the most difficult times of my life. I am so lucky to share the precious friendship and those beautiful moments with you all.

Last but not least, I want to acknowledge my dear auntie Fang, uncle and Cousin Lucy for their support of my study in Sweden. I’d also like to thank my beloved father for always being there for me and other family members for their unconditional help and love.

Table of Content

1 Introduction

This chapter contains the brief background information regarding why the project is initiated, what results to be achieved and which Ericsson SDKs to be evaluated.

1.1 Background

1.1.1 Participants

This master thesis is initiated by the technical service group of Ericsson Mobility World, whose mission is to assist operators, application and content providers to define and deploy solutions for traffic and revenue growth, by using its global network, partners and expertise.

Successful accomplishment of this thesis project will lead the author to a Master of Science degree specializing in Internetworking in Royal Institute of Technology (KTH).

1.1.2 About Ericsson Service Enabler SDK

Ericsson Service Enabler SDKs provide Mobile Internet Applications with interfaces towards Ericsson Service Enablers that consists of a group of application servers. The provided interfaces allow application developers to access the key capabilities of a mobile telecom network, for example, sending and receiving SMS and MMS, Mobile Positioning service, and charging controls.

Before the Ericsson Service Enabler SDKs being published, the pre launch verification of the Server side SDKs for Ericsson Enablers is carried out by Ericsson Mobility World Technical Services group to ensure the usability of the Ericsson Service Enabler SDKs. The reason is to keep the SDKs at a high market standard so that mobile application developers feel comfortable with using them and can easily start developing applications towards Ericsson Service Enablers.

This project involves the evaluation of three Ericsson Service Enabler SDKs: Diameter Charging SDK, MPS SDK and NRG Parlay SDK.

Diameter Charging SDK provides real time charging interface towards Ericsson PPS (PrePaid System) 3.6 for both prepaid and postpaid subscribers. Through the Diameter Charging SDK, the application could charge the end user for one atomic service, e.g. SMS synchronously or charge for one continuous service, e.g. online gaming, asynchronously.

MPS SDK supports the Mobile Location Protocol (MLP) - the standardized interface between a Location Based Service application and the Mobile Positioning System.

Ericsson NRG SDK provides a standard interface for accessing telecom network capacities independent of the underlying networks. It has one Core API in Java for creating applications fast and conveniently.

1.2 Objective

The objective of this project consists of two parts:

- To evaluate and ensure the quality of Ericsson Service Enabler SDKs by finding out usability and pedagogic improvements as a developer that uses the Ericsson Service Enabler SDKs.

- To provide the test specifications and classes to the application developers to simplify and shorten the effort of understanding the SDKs and the protocols that the SDKs are built on. This aims to help them to develop a portable, robust and user-friendly Mobile Internet Application that can not only interact with the SDK emulators, but also, which is more important, can work with the real service node (Service Enabler). This objective leads to a product CDK (Complement Development Kit) that only applies to Diameter Charging SDK.

- To provide the design and implementation of the high level API for Diameter Charging SDK.

The object of this thesis project has vast significance to both Ericsson and the thesis student.

For Ericsson

The evaluation of SDKs provides suggested improvements and references for the new releasing of the future Ericsson Service Enabler SDKs. This will help Ericsson to produce future SDKs with better usability and feasibility for mobile internet application development so that to attract and persuade more and more Mobile Internet developers to develop applications using Ericsson Service Enabler SDKs. In the end this will generate traffic on Ericsson service enabler in the operator site, which will in turn motivate operators to buy the future Ericsson Service Enabler.

In a word, the ultimate goal of this project for Ericsson is to exceed the competitors and gain the mass market regarding Mobile Internet.

For the student

First of all, the student is given the chance to experience being a part of the world-leading Telecom Company, Ericsson, by working with professional Ericsson employees and the most up to date Ericsson technologies.

Second, by carrying out this project, the student will gain the precious knowledge and experience in the following technical field: Mobile Network, JAVA related technologies, web technology, and so on. Besides, the experience of project management and the ability of working both cooperatively and independently are also what the student will achieve.

Last, but not the least, the success of this thesis project will lead the student to a Master of Science degree in KTH (Royal Institute of Technology), Sweden.

2 Technologies Overview

This project involves three major technology areas: Internet, Telecom and Java related OOP programming. Internet acts as an interface for the end user to access the traditional telecom network capabilities, such as Charging, MMS, etc. The web pages can be accessed either by computers explorers through HTML or by mobile phone and PDAs using WAP/WML technology.

2.1 Mobile Internet Architecture introduction

Mobile Internet combines the flexibility of IP-based Internet /web technology and the rich services that telecom wireless network provides. Mobile Internet applications are built using mobile service programming SDKs which provides access to the real mobile service-enabler nodes in the telephony network. With Mobile Internet technology, end users can send MMS/ SMS to a target mobile phone or download a ring tone through a web interface. Besides, the WML-compatible mobile phone user can also access the web interface of various applications using WAP/WML technology.

One Mobile Internet application could integrate services of several functionally related service-enablers together, for example, when a Mobile positioning service is delivered through MPS node, the end user's mobile credit should also be deducted through the Diameter Charging node. Figure 2-1 shows an ideal structure of putting all the SDKs evaluated in this thesis project together.

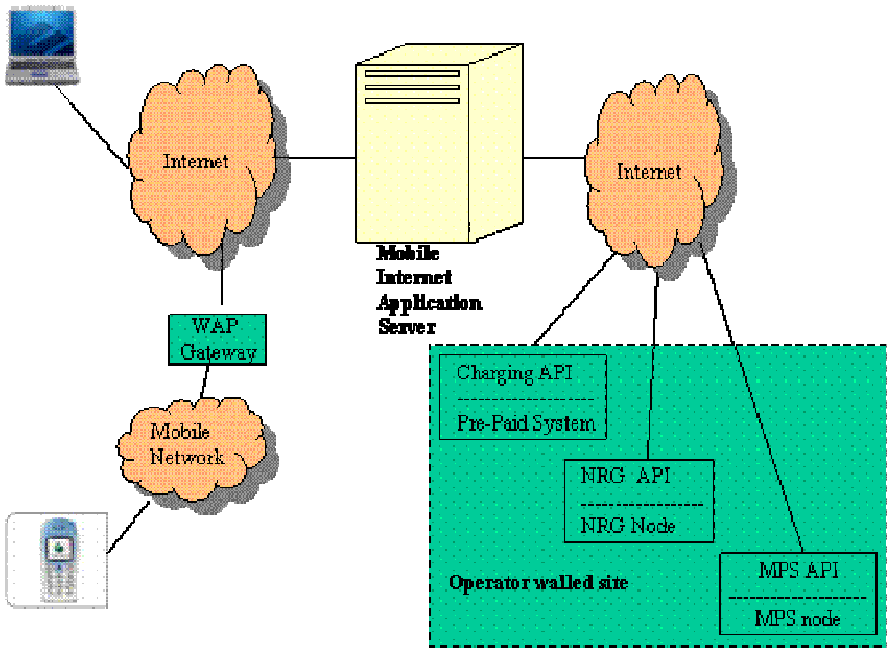

Figure 2-1, putting all SDKs together in the Mobile Internet architecture.

All the real service-enabler nodes are hosted on the Operator site, which provides secure access for Mobile Internet Application server. The application server could be hosted by either the Mobile Internet service providers or the operators.

2.2 Java and related technologies

Java has been a widely-used object-oriented programming language during the last ten years. Java's platform independent feature and web and enterprise technologies make a unique and advanced programming language, compared to other OOP languages, for example, c++ together with .NET platform.

In this thesis project, all the SDK libraries have Java interfaces, which make these Ericsson SDKs, could be implemented on all major operating systems, such as Windows, UNIX, Linux and Macintosh.

In this chapter, all major Java technologies and their relationships with the SDKs will be introduced briefly.

2.2.1 J2SE

J2SE (Java 2 Platform Standard Edition) is a principle product from SUN , released in the form of JDK (J2SE Development Kit), and includes JRE (Java Runtime Environment) and the necessary tools for e.g. debugging and compiling applications and applets. JDK consists of APIs (Application Programming Interface), JVM (Java Virtual Machine) and other components used for running Java applications. JDK is the foundation of J2EE (Java 2 Platform, Enterprise Edition) for enterprise Java application developments.

With J2SE, developers can develop the two most common types of programs: Java application and Java Applet. Java application can run stand alone, while applet can only run within a Java-compatible web browser.

JVM is the fundamental concept of Java technology and provides the platform-independent feature of Java language. With the help of the compiler in JDK, the .java source files are compiled into .class files, which are called java bytecode files and can be interpreted by JVM, and thus can be run at any OS as long as JVM is supported and installed.

J2SE API provides the programming interface for access to many fundamental features of Java platform such as:

- OOP basics: class, thread, file input and output, error and exception handling, etc.
- Networking: socket, TCP/IP, UDP, URL, etc
- Database : Java Database Connectivity (JDBC)

- Security: electronic signatures, access control, public and private keys, etc
- JavaBeans: essential J2EE, JSP/Servlet component.
- Object serialization: RMI (Remote Method Invocation)

2.2.2 J2EE (Java 2 Platform, Enterprise Edition)

J2EE provides component-based multi-tiered enterprise application architecture, as shown in Figure 2-2. A web or business middleware tier residing in J2EE server between the client tier and the legacy EIS tier enables the fast deployment, security, portability, scalability of enterprise solutions.

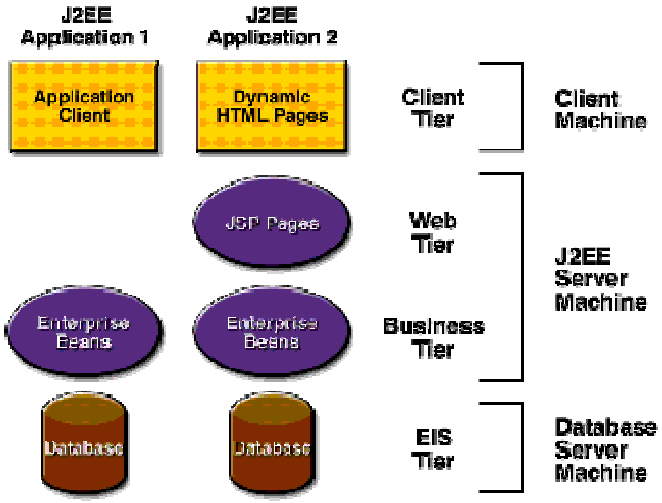

Figure 2-2 [*ref 1*] Multi-tired structure

The Client Tier could consist of either application or Java applet. The Web Tier can be implemented by either JSP pages or Servlet. The Business tier consists of three different kinds of JavaBeans: Session Bean, Entity Bean or Message-driven Bean. Client application can access these beans directly or through the Web Tier. The Business Tier can access the EIS tier through for example database connectivity.

EJB (Enterprise JavaBeans) is a portable business logic component. Session bean represents a transient interaction with the client, that is, all the data will disappear after the J2EE server is shut down. Entity Bean represents persistent data storage in the EIS system, for example a database. The database item will remain in storage even if the server is shut down. Both the Session and Entity bean exchange data with the Client Tier synchronously, while Message-driven bean provides the possibility to exchange JMS (Java Message Service) messages with Client Tier asynchronously.

J2EE server is a component-based application server. It consists of EJB container accommodating all EJB components and web container for web components, as shown in Figure 2-3. The EJB components should be written in accordance to the EJB specification by SUN. There are some restrictions on what an EJB bean cannot do. For example, EJBs cannot handle threads nor file I/O processing. This is an important issue for creating J2EE application using Diameter Charging SDK, which involves thread underneath the provided API. In the evaluation process of Diameter Charging SDK, one suggestion is raised to solve this problem: to create a Resource Adapter, based on the J2EE Connector Architecture as shown in Figure 2-4.

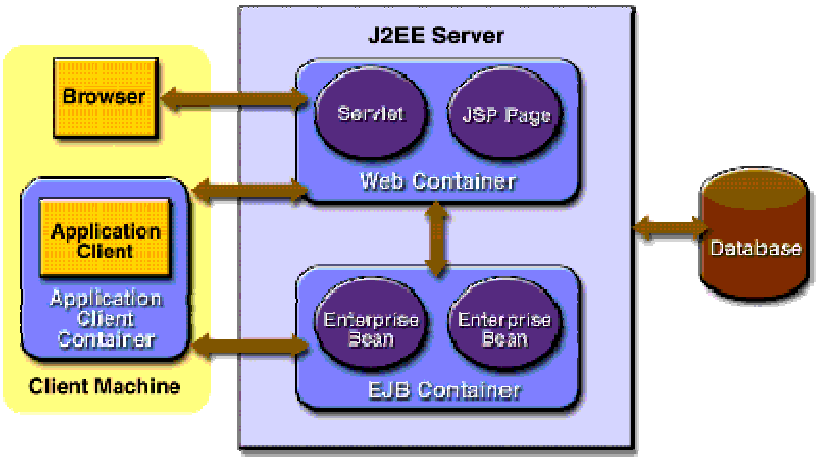

Figure 2-3 [*ref 2*] J2EE Server and Containers

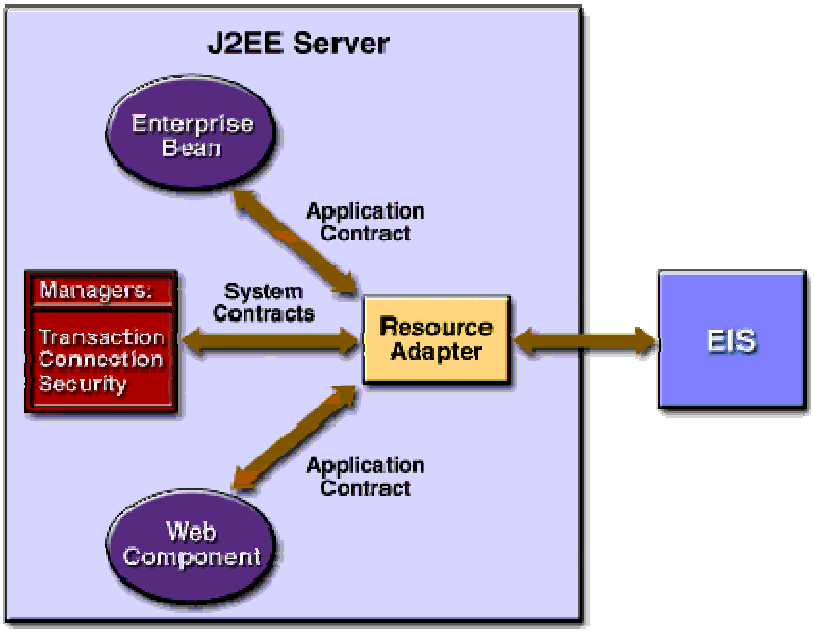

Figure 2-4 [*ref 3*] Resource Adapter Contracts

The development of Resource Adapter is not included in this thesis project and is handled by another master thesis in parallel.

2.2.3 Servlet and JSP

Servlet and JSP provide the possibility to build dynamic and platform-independent web applications with Java programming language. JSP has the Java code embedded in the HTML file, while Servlet is ordinary Java class which generates HTML tag in the code. JSP provides more flexibility by separating the design of HTML layout from dynamic content creation. JSP files are compiled into Servlet files at runtime. Both JSP and Servlet can work with JavaBeans, JDBC, EJB and RMI (Remote Method Invocation) technology.

Servlet and JSP applications need to be deployed in a compatible web server, such as, Tomcat, BEA Logic, or SUN J2EE server, etc.

2.2.4 XML and JAXB

XML (EXtensible Markup Language) is a markup language which describes data with user-defined tags. The structure of the tags (grouped in elements, attributes and entities) can be described in an XML Schema or a DTD (Document Type Definition) file. XML Schema file is XML file itself and follows XML syntax.

XML file is used to store, transfer and manipulate information across platform independent software and hardware system. Many software vendors have implemented XML standard in their product to retrieve and process information, such as Microsoft Office Software and JAXB (Java™ Architecture for XML Binding) technology from SUN.

JAXB provides the flexibility of processing XML data for Java applications. Given an XML schema file that describes the XML document, JAXB compiles the schema file into structured Java classes, whose instances can be used to validate, read from or write to the XML document, as shown in Figure 2-5.

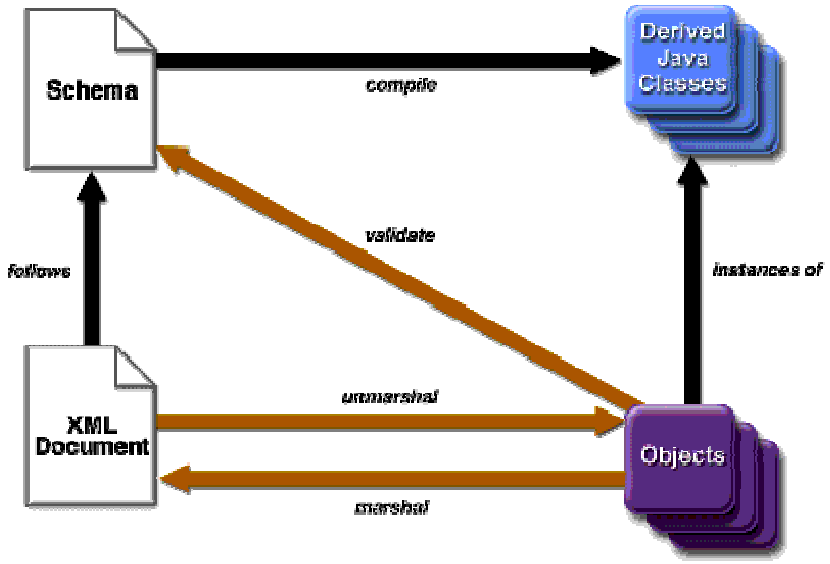

Figure 2-5 [*ref 4*] JAXB Binding Process

Charging Server and Client emulators in Diameter Charging SDK utilize XML file to store system and custom information such as: Charging Client and Server URI, subscriber name and mobile numbers, etc. The Charging CDK (Complement Development kit), one of the result of this thesis work, also uses XML together with JAXB technology to store business cases parameters.

2.2.5 Ant build tool

Within this thesis project, Ant build tool is frequently used to compile and execute java program.

Ant is an open source java-based build tool from Apache Software Foundation. The build file of Ant is an XML file conventionally called “build.xml”. Build.xml file consists of one pair of <Project></Project> tag, which in turn embraces several <target></target> tag pairs, specifying the target program to run respectively.

Ant makes building Java program platform-independent, compared to the platform-specific shell-based commands.

3 Evaluation procedure

The evaluation work of Ericsson SDKs is carried out by the author, a moderate Java developer at the time, to draw an object and intensive feed back regarding the SDK products' usability, quality, and etc.

The author concludes all the evaluation results through testing and implementing the main use cases supported by each SDK. The evaluation results include the pros and cons of each SDK and also recommendations and references on how to improve the cons.

The evaluation results are put into separate evaluation documents and will be studied by the SDK designed organizations as an important reference for the improvement of the future product release.

3.1 Evaluation criteria

The Ericsson Mobility World has created several guidelines to set up a common standard for the SDK design organizations. These guidelines are the major criteria for the evaluation work, as listed below.

- SDK_technical_guideline
- SDK_delivery_guideline
- SDK_documentation_guideline
- SDK_packaging_guideline

From a developer's point of view, a SDK for a specific system should have several general properties for it to be called a "good" SDK. These properties vary from system to system, but the common requirements are the same. For a SDK designed for Mobile Internet, it should be, first of all, easy to use, which means that a moderate developer needn't study a great volume of documents and materials before the developer can develop an applicable and deployable application. Second, the feasibility and usability are also of great importance, because it's the supported functions and use-cases that attract the developers to use the SDK. Third, the compatibility or the inheritability of each version of the SDK is another important factor to concern. Of course, there are also some other important features that a good API is supposed to have - high efficiency, stability, etc. [Ref 5].

3.2 Evaluation technique

The main measures and techniques of the evaluation work include:

- Implement the main use cases of each SDK and give feedback on the usability, inheritability, compatibility and other aspects regarding APIs.

- Read all the product documents (User's Guide, javadoc, etc.) to give feedback on readability and accuracy of documentation.
- Test the usability of Emulators included in the SDK, if any.

3.3 Evaluation platform

The evaluation work is carried out on a PC with following system configurations:

- Operating System: Windows 2000
- Hard disk size: 40G
- Memory Size: 512M
- CPU: 1.4G Hz

4 Evaluation of Ericsson Diameter Charging SDK

The **Ericsson Diameter Charging SDK** 1.0 D12E for PPS 3.6 is an important part of Ericsson's charging solutions offering. It could be downloaded from Ericsson Mobility World free of charge. The Charging SDK enables standardized charging for content and services in an IP-based network. In a live environment, Ericsson Charging System 1.6 / Ericsson PPS (Prepaid) System 3.6 is the system communicating with the Charging SDK.

The Mobile Internet charging application uses Diameter interface to exchange charging information with the Ericsson PPS, as shown in Figure 4-1[Ref 6]

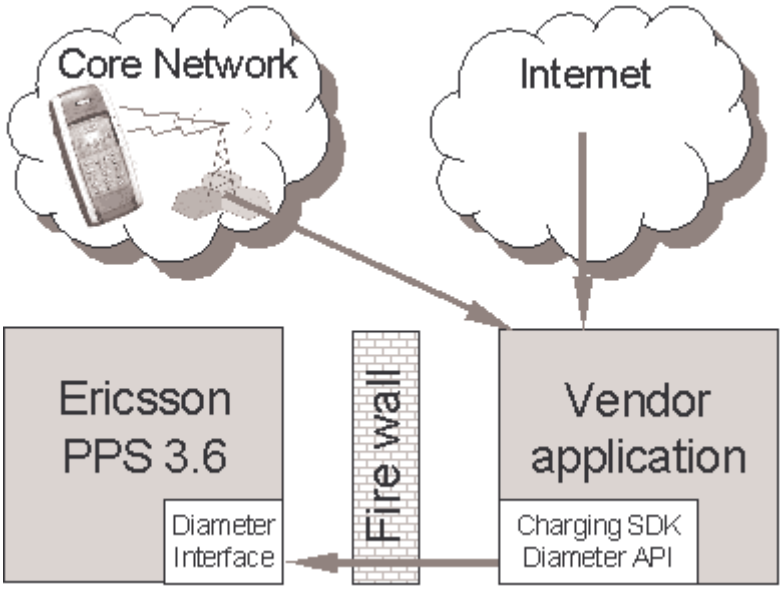

Figure 4-1[Ref 6] Diameter Charging SDK system architecture

4.1 Inside the SDK

The Diameter Charging SDK includes the following main components:

- Diameter API specification and the javadoc for the API
- Diameter Charging Server (PPS) emulator
- Diameter Charging Client emulator
- Product documentations for Diameter API, Server and Client emulator.

Diameter API is the programming interface for development of charging applications. The API isolates the protocol implementation from the developers and allows them only concentrate on implementing their own business cases based on the supported use cases.

Diameter Charging Server Emulator is developed with Diameter API and simulates the behavior of Ericsson PPS, against which the charging application could be tested. Diameter Charging Server Emulator uses XML file to store the emulator configuration data and three system databases for Currency, Users and Tariff.

Diameter Charging Client Emulator is an example Diameter charging application, which can perform the main use cases supported by the SDK. Both the Server and Client simulator have the logging functionality to keep track of the traffic between the server and client applications.

4.1.1 Ericsson PPS (Pre-paid System)

Ericsson PPS works as a real time pre-paid charging system for mobile telephony networks. It allows the mobile subscribers to access the personalized mobile internet services within the subscribers' account credit limit. It ensures the cost control for the mobile users and consequently ensures the profit for the operators and service providers.

PPS offers flexible charging functionalities in scalable mobile networks. It supports both debiting and crediting the user's account so that the users could get the deducted credits back if the requested services are not successfully delivered.

4.1.2 Diameter base protocol

Diameter Charging SDK uses Diameter base protocol (Draft 8) for exchanging charging information over the transportation network layer.

The Diameter base protocol is intended to provide an Authentication, Authorization and Accounting (AAA) framework for applications such as network access or IP mobility. Diameter is also intended to work in local Authentication, Authorization & Accounting and roaming situations [Ref 7].

One of Diameter base protocol's main features is the delivery of AVP (Attribute Value Pair), which is implemented in Ericsson Charging SDK as well. The charging Client sends charging request in format of AVP code, and receives the charging result in format of result AVP code. For example, result code 201 indicates the success of requested charging service.

4.1.3 Evaluation deliverables

The evaluation result of Diameter Charging SDK consists of three main deliveries:

- **Evaluation Report** of Charging SDK 1.0 D12E for PPS 3.6

Evaluation Report is intended to give the Charging SDK design organization recommendations and references on how to improve the performance and usability of future released Charging SDKs, as well as suggestions on how to make the SDK conform to the specifications of the most commonly used enterprise JAVA technologies, such as J2EE.

Extracts of this document are also intended for the Java application developers who will use the Charging SDK in their applications. This document can foresee what problems they will encounter during the developing process and also give the corresponding suggestions and solutions, if possible.

- **CDK (Complement Development Kit V1.0**) for Charging SDK 1.0 D12E

Diameter Charging CDK acts as a complementary kit to Ericsson Diameter Charging SDK. Diameter Charging CDK provides more background information about Charging and Charging use cases to facilitate the development of efficient and flexible charging applications. In the CDK, an example J2SE application is included to demonstrate the main use cases that the Charging SDK supported and the flexibility of combining Diameter SDK with different Java technologies, for example, JAXB (Java Architecture for XML Binding). In the User's Guide, detailed instructions are provided on how to modify the XML file, how to compile and run the example application.

- Charging High Level API specification

As stated in Chapter 4.3.2.1 of the evaluation report, Charging API 1.0 D12E is still a low level API since it still requires the knowledge of the underlying protocols. So a new Charging High Level API was proposed within this thesis project and eventually it was adopted by the design organization for the later release of Charging SDK.

4.2 Evaluation result Overview

Generally speaking, the Charging SDK is a good SDK that makes it possible for the developers to handle real time high performance charging in their J2SE applications towards Ericsson PPS 3.6 system, in despite of the fact that it is poor in some aspects.

The Charging SDK includes all the basic tools needed for the application development using Charging SDK along with the html documentation being so clear and cool in Look& Feel. It also has some drawbacks and deficiencies. For example, the Diameter Charging API is a low level API and doesn't conform to the J2EE specifications, etc.

The detailed pros and cons of the components included in the Charging SDK will be covered in the later chapters. More efforts will be put on the following items: the Diameter Charging API, the documentation and the Diameter Emulator.

4.3 The Diameter Charging API

4.3.1 Pros

Well-designed structure

The structure of the API is based on the hierarchy of the Diameter Base and SCAP protocols.

Package com.ericsson.pps.diameter.api.base is mainly intended to work against the Diameter implementation found in PPS 3.6 and is not intended to be a full implementation of the Diameter Base Protocol for general use.

Package com.ericsson.pps.diameter.api.base handles peer specific diameter commands such as CER/CEA, DPR/DPA and DWR/DWA. Handling of other commands and new diameter applications are done through plugins that are implemented outside of this package. The package com.ericsson.pps.diameter.api.scap is an example of such a plugin.

The API supports sending and receiving messages both asynchronously and synchronously. A single application can send multiple Diameter messages and it can simultaneously process multiple incoming messages as well.

The Charging API isolates the core protocol

The API isolates the core Diameter protocol implementation from the developer and allows the application to implement the pre-defined Diameter interface with operations relevant to the application, for example, the DiameterAnswerListener interface.

Developers do not have to worry about details regarding transport protocols, network addressing and so on.

4.3.2 Cons

The Charging API is a too low level API so a high level API is needed

Even though this API isolates the core Diameter Protocol from the developers; it's still a low level API since it requires the developers to know the knowledge of SCAP. The experience the author gained when developing some test classes using the Charging SDK also lead to the conclusion that it's even necessary for the developers to study a small section of the Diameter Base protocol.

The part mentioned above relates to the “Terminology” in Section 1.3, Page 7. Knowing the precise explanation of the listed terms that appear quite often in the context of the Diameter Charging SDK will reduce the ambiguity and the chance of misunderstanding of the terms during the process of using Charging SDK especially when reading the Diameter API. Special effort should be made to fully explaining the following terms: Accounting record, AVP, Home Realm, Local Realm, Realm, etc. So I suggest that a “Terminology” section to be included in the HTML documentation to explain the common terms that appear in the SDK. I think this will help the developers to understand the document and enable them to begin developing their own applications more quickly.

It is of great importance and convenience for the developers to have a high level API, which lower the knowledge threshold and shorten the time needed for the developers to be able to start developing applications using Charging SDK.

For example, for a charging request on sending one SMS, if the incoming request is not pre-rated, the Emulator must find the price of the service. The Emulator finds the price for the requested service by searching for a match of the incoming parameters in the tariff list. What we need to do now is:

```
ServiceParameterInfo serviceProviderId =

 new ServiceParameterInfo(ServiceParameterInfo.SERVICE_PROVIDER_ID,
"500");

 myAcr.addServiceParameterInfo(serviceProviderId);
```

In case of a high level API, what we need to do in order to charge a SMS service could be as simple as:

```
myAcr.addService(“SMS”);
```

Here “SMS” is a service name in the tariff of the Diameter Server emulator.

It would be much easier to code and understand in a high level API. While we may reduce flexibility in the use of the Charging API, we would certainly be helping the vast majority of developers by providing more high-level capabilities to support them in developing charging support for the more common functions.

Improvements needed for the Result Code AVP

I suggested that in the html document the result code could be elaborated. And in the API, the result code has both a number and a clear text (String) format.

The result code AVP is useful for developers when debugging the application with the Diameter emulator to find out the cause of a certain problem. In the Diameter API, the SCAPResultCodes and DiameterResultCodes are given in the forms of AVP. That is, a static parameter name, which stands the meaning or reason of the result, corresponds to a specified result code.

For example, DIAMETER_END_USER_NOT_FOUND corresponds to result code 5241

```
int  para1= SCAPResultCodes.DIAMETER_END_USER_NOT_FOUND;

System.out.println(”Result code of DIAMETER_END_USER_NOT_FOUND is : ” +para1);
```

Running result:

Result code of DIAMETER_END_USER_NOT_FOUND is: 5241

In the API, I suggested that a certain method is added to the Class SCAPAca or the toString() method is overridden in the Class AVP so that the static parameter name could be derived from the returned result code. This will make it easier for the developers to do the debugging and logging of the program.

Suppose there is a method “String getResult() “added to the Class SCAPAca;

```
SCAPAca answer;
int resultCode = -1;
String resultString=””;

   try {
     resultCode = answer.getResultCode();
     resultString= answer.getResult();
   }
   catch (AvpDataException e)
   {……}

   if (resultCode != DiameterResultCodes.DIAMETER_SUCCESS)
   {
     System.out.println("The Session Failed. Result-Code "
                    + resultCode + " was returned.");
     System.out.println(resultString);}
```

Suppose resultCode= 5241

Running result:
The Session Failed.Result-Code 5241 was returned.
DIAMETER_END_USER_NOT_FOUND.

About the result code, it is also suggested that in the html documentation, some important result codes could be elaborated. This is useful for the developer to handle the errors associated with some result codes. For example, the reason for the result code 4241(DIAMETER_END_USER_SERVICE_DENIED) could be “the account balance for the end user is not sufficient for the requested service”.

A last credit indicator interface is needed

For the session based charging, it is necessary for the end user to be notified before the user’s account balance reaches zero. Then the application using charging functions can notify the end user before stopping the delivery of the service. One phone call, for example, of the end user can still last for the last minute instead of being interrupted right away.

So I suggested that an interface is added to the package “com.ericsson.pps.diameter.api.scap”.

public interface SCAPLastCreditIndicatorListener {

public void processIndicator(long moneyUnit, String currencyName);

}

Then in the client application that implements this interface, the processIndicator() method should be implemented:

Class clientProgram implements SCAPLastCreditIndicatorListener

{……

public void processIndicator(long moneyUnit, String currencyName) {

System.out.println(”There is only “+moneyUnit+currencyName+”left in your account, please terminate your requested service and credit your account. Otherwise your service will be terminated when your account balance runs out.”);}

}

The Charging SDK could not be used within EJBs(J2EE platform)

According to the EJB specification, enterprise beans should not create or manage threads. This means that synchronization primitives include the synchronized keyword, as well as methods wait(), notify() and notifyAll() of class Object, are disallowed in enterprise beans. The reason for this is that the EJB specification assigns to the EJB container the responsibility for managing threads, not the bean providers.

The Charging Diameter API contains synchronized blocks, who inherit from class Object of methods wait(), notify() and notifyAll(). So the developer could not use Charging API within EJBs.

One possible solution to make Charging SDK conform to the EJB specifications is to provide a resource adaptor along with the Charging SDK by the Charging SDK designer. The resource adaptor is a J2EE component that implements the J2EE Connector Architecture who enables J2EE components such as enterprise beans to interact with enterprise information systems (EISs).

Account balance checking and price inquiries are not supported by CCN.

In fact, the Charging API supports another use case: account balance checking. But unfortunately at this moment we could not use it since the CCN (Charging Control Node) does not support the balance checking yet. But as a developer, the author hopes this function could be supported soon since it's an important part of the Charging service.

Besides, it would be great if the function of price inquiries could be supported by the CCN also.

Security issue

In the HTML documentation, it tells "The API only supports TCP/IP as transport protocol. It does not provide nor use any security mechanisms such as TLS or SSL. If a secure network environment is needed, it is up to the user of the API to take the appropriate actions, for example by using IPSec."

So the charging application developer and the operator have to cooperate on the security issue. The operator who owns the charging server should provide the authentication and authorization mechanism for the charging application. And the charging application should provide security mechanism for the end user to ensure that no subscribers are charged incorrectly.

4.4 The documentation of the Charging SDK

4.4.1 Cons

A real programmer's guide is needed

Programmer's Guide included in the SDK is a Service Charging Application Protocol guide and NOT a guide on how to code Charging SDK API features.

In the “User's Guide - PPS Diameter API” document, there is a chapter, called “Charging SDK Quick Introduction”, actually acts as a small Programmer’s guide on Charging API. It lists the main use cases that the SDK supports and also gives a brief introduction on the procedure of coding the two main groups of use cases: direct debit and charging session. But this is not enough for a real programmer’s guide. More content could be included in a programmer’s guide such as, the use of some essential classes, the complete code example of a business case, how to debug errors and handle exceptions, etc.

More example applications are needed

There are only three example code files included in the SDK: StateChange.java, DirectDebit.java and Session.java. DirectDebit.java and Session.java only show how to implement the main use cases. I suggest that several example applications are added to the SDK to show several complete business cases that are based on the use cases. For instance, example applications on how to do the charging with PPS on a regular phone call (use case: charging session based on used time) and a file downloading service (use case: charging session based on used volume).

Example business cases are needed.

When the SDK designers defined the supported use cases, they had some business cases in mind. Therefore the author suggested that these example business cases could be documented in the SDK. This will help the developer to understand the original intention of the designer on each use cases and also help the developer to create their own business cases.

There are two main groups of Charging services, Charging session and direct debit.

Charging session is used for the service that could not be completed once a time and the whole service process could be divided future, such as a phone call. During the delivery of the service, the charging application has to communicate with the charging server periodically to keep the charging session alive.

Direct Debit is used for the service that could be delivered all at once, such as sending an SMS. If the service is not delivered successfully, the user’s account could be refunded by the application using the “Direct refund” use case.

The following example business cases are what the author collected during the study of Charging SDK.

Use case 1: Charging Session based on used Time

Business case: Pre-paid phone card, Internet access service based on Time

Use case 2: Charging Session based on used Volume

Business case: Streaming of movie or news based on used bytes.

Use case 3: Charging Session based on used Event Services

Business case: Surfing the net, e.g. each click of the links generate one event; Restaurant reservation (session consists of several SMS services. In the Charging web cast, this is an example of charging session based on used volume, but the author thinks it's more appropriate to put it here.).

Use case 4: Charging Session for Pre-rate Services

Business case: Playing a poker game (like Jack Vegas), the user pays per game.

Use case 5: Direct Debit based on used Time

Business case: Downloading MP3 based on the duration of the music.

Use case 6: Direct Debit based on used Volume

Business case: Download movie file on byte volume, buy 300 ml volume coke at Mac Donald

Use case 7: Direct Debit based on used Event Services

Business case: Delivering MMS, SMS, MPS services, e.g. sending three SMSs at one request generates three events.

Use case 8: Direct Refund for Pre-rate Services

Business case: refund user's account

4.5 The Diameter Client and Server Emulator

4.5.1 Pros

These two are very useful tools

It's a very good idea to have the Diameter Server Emulator and the integrated test client. The server Emulator emulates the behavior of the Diameter implementation of the PPS so that to make it easier for developers to perform the basic test as if towards a real PPS system.

The Diameter Client is an example application using the Diameter Charging API. I think this as a very useful tool. The developers will get the basic idea on which parameters are needed for a SCAP request message and how the applications will interact with the Emulator. For example, the developer will get the first experience on how to perform a successful charging session that consists of the start, interim and stop request.

Emulators read the parameters using XML technology

It is very convenient that all the configuration parameters could be saved in and read from an XML file. It gives the developers the hint that they can use XML file in their own applications as well.

As we know, nowadays XML is more and more widely used to describe, manipulate and transmit data. With the XML file, the developer need not modify and change the source code all the time when the parameters need modification.

Powerful logging panel

In the logging panel, the developers can see the detailed peer-to-peer communication process and the sent messages between the client and the Emulator. And also different log levels can be set as needed.

4.5.2 Cons

The measurements of services unit need to be specified.

For clarity, all the Requested-Service-Unit should have a specified measure. On “Programmer’s Guide” page 42, we can know that the measure for Time is second and for volume is byte.

So it would be helpful if in the “Create Request” window of the Diameter Client, in “Type” drop down list of the Requested Service Unit and Used Service Unit items, the “0-Time” could be replaced with “0-Time (second)” and the same applies to “1-Volume” with “1-Volume (byte)”.

And, when a new service is added to the tariff list of the Diameter Emulator, the service should be rated based on per second, per byte or per event.

A bug found in Diameter Emulator should be corrected

The real cost could not be fetched when the result codes other than DiameterResultCodes.DIAMETER_SUCCESS are returned.

After a SCAPAca answer is returned from the Diameter Emulator, if the result code of the answer is 4241 (when the account balance of the user is not sufficient to cover the requested service units), the cost information couldn't be fetched from the answer. And an AVPdata exception is thrown. This means that when a charging session ends without a stop request/answer, the cost information available from the API differs from (less than) the actual cost deducted from the user's account in the Diameter Emulator.

5 Diameter Charging CDK

The CDK exists because current release of the Charging SDK does not include some practical information users might want, such as more examples based on some concrete business cases and more background information on charging, so the Complement Development Kit (CDK) was produced. It includes an example application that can demonstrate all the use cases the Charging SDK supports.

Charging CDK can be downloaded from Ericsson Mobility World website for free. The package includes:

- An example application demonstrates all the use cases supported by the SDK
- User's guide on detailed elaboration of building business cases based on the supported use cases and also how to install, run and modify the example application.

5.1 CDK: Real-time Charging of Pre-Paid System introduction

Real-time charging is used when the charging process is performed as a precondition for delivering a service. That is, the application won't deliver the service until assuring that the user's account balance is sufficient to cover the requested service. It will then debit or reserve money from the end-user's account, with all charges appearing on the user's phone bill.

A charging application composes and transmits the charging messages to the charging system. The roles of a charging application include:

• Identifying what service and how many service units the consumer has requested
• Requiring the charging system to charge the service
• Delivering the amount of service the charging system has granted
• Recording the transaction in a detail record, including the final cost deducted from the user's account.

There are two main charging mechanisms offered by the Charging API: Direct Debit and Charging Session.

Direct Debit

• Based on Events
• Based on Time
• Based on Volume
• Direct refund

Charging Session

• Based on used Events
• Based on used Time
• Based on used Volume

Direct Debit is a simple event that can be understood as an atomic transaction. Direct Debit is chosen by the application when the full scope of the service to be delivered is known before the delivery and the application will not supervise the delivery, for example, sending an MMS. And Direct refund is chosen when it's necessary to return a certain amount of money to the user's account.

At a Charging Session, the final cost of the service is not known at the start. Charging Session is chosen when the full scope of the service to be delivered is not known before the delivery and the application will supervise the delivery. For examples, a phone call or file downloading service.

5.2 What's inside the Package

5.2.1 Package structure

```
+CDK
  +classes
    +jaxb
    +usecase
  +javadoc
    +jaxb
     +usecase
  +doc
    -User'sGuide
  +src
    +jaxb
    +usecase
      -ChargingMain.java
      -ClientListener.java
      -DirectCharging.java
      -DirectDebit.java
      -processXML.java
      -Session.java
      -SessionCharging.java
  -backup.bat
  -compile.bat
  -generatejavadoc.bat
  -run.bat
  -setenv.bat
  -startCMD.bat
  -testCase.xml
  -testCase.xsd
  -XSD.bat
  -XSDcompile.bat
```

The illustration above depicts the package structure for the "CDK" package. Directory "src" contains all the source files of this sample application. Directory "classes" contains all the class files generated from the source files. Directory "javadoc" contains all the javadoc

generated from the source files. “testCase.xsd” is the XML schema for the “testCase.XML” file, which stores the test cases that the sample application can demonstrate.

5.2.2 Introduction to batch files

“startCMD.bat” is used to start a command line window.

“setenv.bat” is used to set the environment variables.

“XSD.bat” is used for generating the classes from the schema file testCase.xsd.

“XSDcompile.bat” is used for compiling the generated classes from “XSD.bat”.

“backup.bat” is used for making a backup .zip file of the whole package in the parent directory of this package.

“generatejavadoc.bat” is used for generating Java doc from the source files

“compile.bat” is used for compiling the source files of the “testcase” package.

“run.bat” is used for running the sample application.

5.3 Source code elaboration

5.3.1 Source code structure

The “src “ directory consists of two packages: jaxb and usecase. The “jaxb” package includes the source files generated from the XML schema “testCase.xsd” using JAXB technology for reading the XML file into the application.

The “usecase” package includes all the source files that construct the application. The following illustration depicts the structure of these classes.

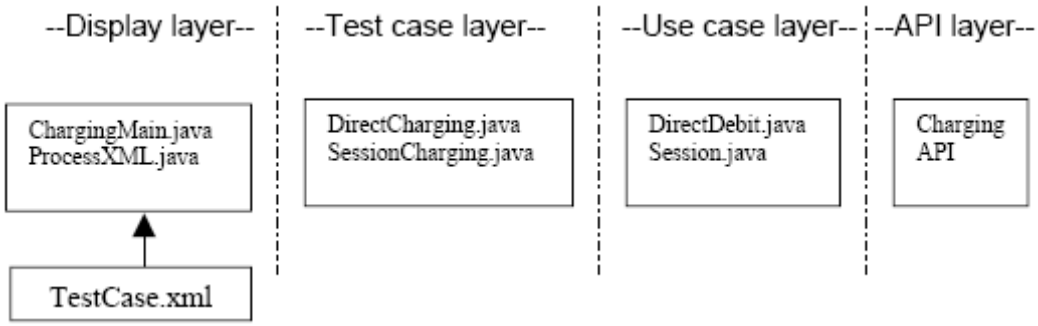

Figure 5-1 Source Code structure of CDK

ChargingMain.java
This is the main class of this charging application. It provides the interaction dialog between the user and the program, through a command line window, on which use case to choose and when to terminate the charging session.

ProcessXML.java:
The purpose of this class is to configure the Charging event with the parameters fetched from the XML file. JAXB technology is used in this class.

DirectCharging.java
This class is for completing a Direct Debit charging or Direct refund and gives the result string to the ChargingMain.java to show to the end-user.

SessionCharging.java
This class is for a complete session charging and gives the result string to the ChargingMain.java to show to the end-user.

DirectDebit.java
The purpose of this class is to give an example of how to do a Direct Debit by sending the charging request synchronously.

Session.java
The purpose of this class is to give an example on how to perform sessionbased charging by sending requests asynchronously

5.3.2 Flowchart of the main class

The following illustration depicts the flow control diagram of ChargingMain.java.

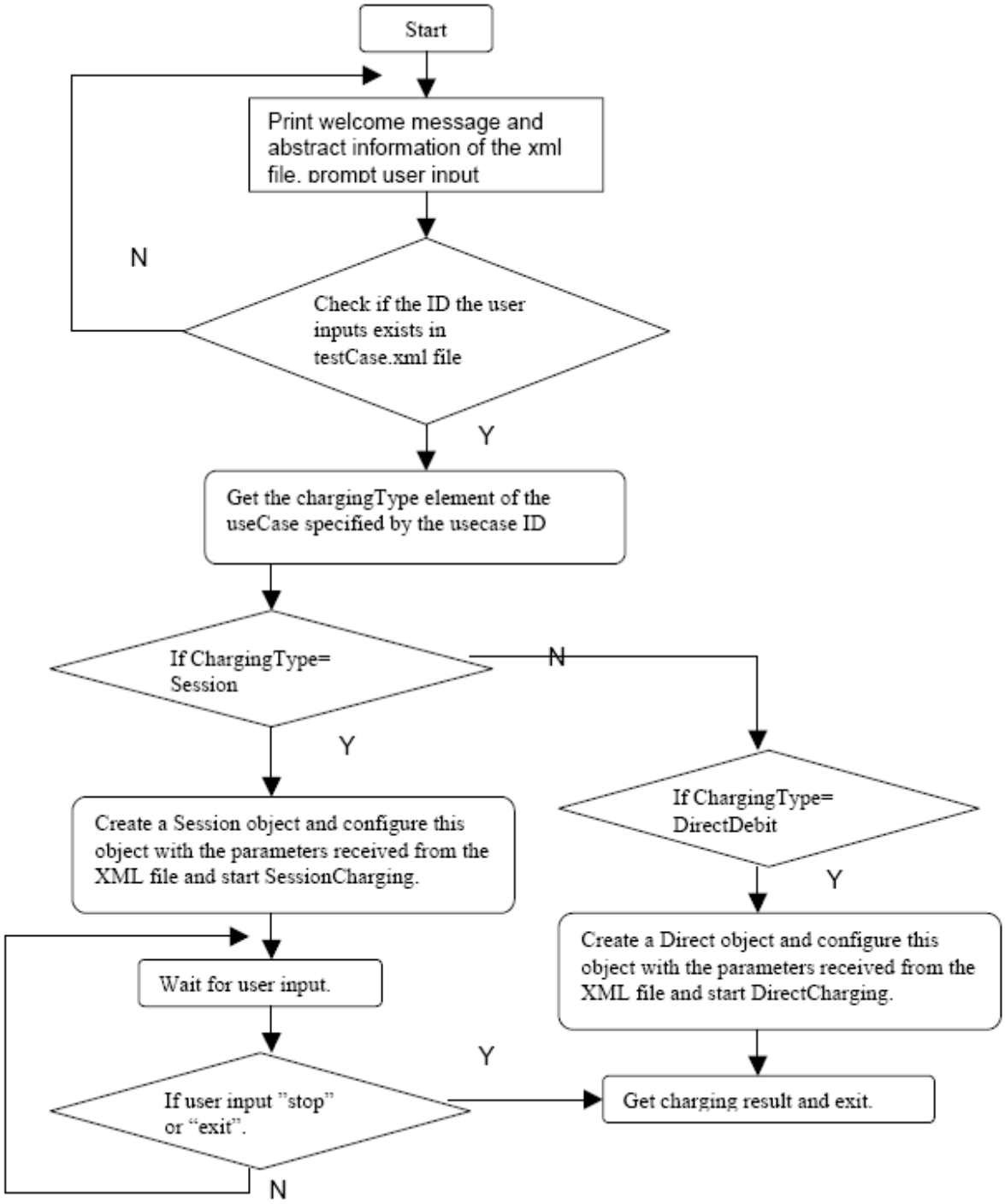

Figure 5-2 Main class flowchart

5.3.3 Coding skeleton of the main class

The skeleton of the main class described in this section is given as a reference on how to reuse the functionalities of **ProcessXML**.java, **SessionCharging**.java and **DirectCharging**.java classes.

```
// Create an XML processor
processXML XMLProcessor = new processXML(XMLName);

// Print the XML file abstract
```

```
System.out.println(XMLProcessor.getXMLAbstract() );

// Check if the usecase ID entered exists.
Boolean XMLFlag = XMLProcessor.checkUseCaseID(userInputLine);

//Check the ChargingType, session charging or direct debit
if ( (XMLProcessor.getChargingType()).equalsIgnoreCase("session")) {}
if ( (XMLProcessor.getChargingType()).equalsIgnoreCase("DirectDebit")) {}

/***Do a session charging*/

//create a session object
Session session = new Session();
// Configure the session object with the parameters received from the XML
file.
session = (Session) XMLProcessor.configChargingInfo(session);
// start the Session Charging
SessionCharging sc=new SessionCharging(session);
// print out the charging result
System.out.println(sc.getResultString());
```

Note! The command prompt for terminating the charging session is the task of SessionCharging object, so the user need not to implement it in the main method.

```
/*** Do a direct debit*/
//create a DirectDebit object
DirectDebit dd = new DirectDebit();
// Configure the DirectDebit object with the parameters got from XML file.
dd = (DirectDebit) XMLProcessor.configChargingInfo(dd);
// Start the DirectDebit Charging
DirectCharging dc = new DirectCharging(dd);
// Print out the charging result
String resultString= dc.getResultString();
System.out.println(resultString);
```

5.4 Executing the example application

For the moment, this package is only intended for Windows users. Linux users can also use the source file in this package, but there is no batch file provided for Linux.

This section contains short instructions on how to run the sample application and how to add additional test cases to the application through editing the XML file.

5.4.1 Start a command line window

By executing the “startCMD.bat” included in the directory, a command line window could be started with the root of CDK as default directory.

5.4.2 Set all the needed environment variables

First, the end user should edit the “setenv.bat” file by specifying the installation directory of the ChargingSDK and the Java Web Services Developer Pack (JWSDP) package.

set CHARGING_HOME=yourChargingSDKPath

set JWSDP_HOME=YourJWSDPPath

Note that the % *CHARGING_HOME* % path should be the parent directory of the “ChSDKapi.jar”. And the JWSDP can be downloaded free of charge from:
http://java.sun.com/webservices/downloads/webservicespack.html

Just executing the “setenv.bat” file can set all the required environment variables for the test application.

It is assumed the users of this package have already installed the JDK 1.3 or above and have already added the installation path of JDK to ***classpath***.

5.4.3 Add a new user and a tariff to Diameter Charging Server Emulator

If the developer wants to add new users to the account list and new tariffs to the tariff list of the Emulator, the developer can do it in one of two ways.

The first way is to create a new user account and tariff item by hand after starting the Emulator. But once the Emulator is closed, it has to be created again the next time, so the second way is recommended.

The second way is to save the user and tariff information to the “PPSDiamEmulConfig.xml” file in the ChargingSDK installation directory before starting the Emulator.

- Add new account to the Emulator

Open the “PPSDiamEmulConfig.xml” with Wordpad or other XML editors.

Add a new “Account” child element to the “Accounts” element. Suppose that a developer wants to charge a user whose name is John Smith and MSISDN is 0737762157. He has SEK 10,000 in his account.

```
<Accounts>
        <Account>
        ……..
        </Account>

        <Account>
        ……..
        </Account>

       <Account MSISDN="0731234567">
            <Name>John Smith</Name>
            <IMSI>0737762157</IMSI>
            <Balance>10000.0</Balance>
            <Currency>752</Currency>
        </Account>
</Accounts>
```

All the developer has to do is add the user information to the respective elements. For the “Currency” element, 752 refer to SEK, 987 refer to EUR and 840 refer to USD.

- Add new tariff to the Emulator

Open the “PPSDiamEmulConfig.xml” with Wordpad or other XML editors.

Add a new “Tariff” child element to the “Tariffs” element. Suppose the developer is providing the end-user with a new service, “CallSweden”. The rate of this service is SEK 0.001 per second and the Service Provider ID is 590.

```
<Tariffs>

        <Tariff>
        ……..
        </Tariff>

        < Tariff>
        ……..
        </Tariff>

       <Tariff Name="CallSweden">
            <Rate>0.001</Rate>
            <Currency>752</Currency>
            <Condition>
                <TrafficCase>20</TrafficCase>
                <ServiceProviderId>590</ServiceProviderId>
            </Condition>
        </Tariff>
```

</Tariffs>

Note that the TrafficCase element is normally set to 20 for normal end-user initiated services.

5.4.4 Explanations of XML tags and charging parameters

In the testCase.xml file, a complete use case element is shown as:

```
    <useCase id="9" description="A call within Sweden">
            <MSISDN>0737762157</MSISDN>
            <ChargingType>Session</ChargingType>
            <UnitType>Time</UnitType>
            <URI>aaa://localhost:9998</URI>
            <Realm>realm.com</Realm>

            <DestinationRealm>test.com</DestinationRealm>
            <DestinationHost>aaa://localhost:1812</DestinationHost>

            <ServiceName>CallSweden</ServiceName>
            <ServiceProviderID>590</ServiceProviderID>
            <TrafficCase>20</TrafficCase>

            <RequestedServiceUnit>5</RequestedServiceUnit>
            <!--UsedServiceUnit and InterimIntervalInSecond are used
ONLY for session based charging !-->
            <UsedServiceUnit>5</UsedServiceUnit>
            <InterimIntervalInSecond>5</InterimIntervalInSecond>
            <!--the CurrencyName is only used for DirectDebit for
REFUND-->
            <CurrencyName></CurrencyName>
    </useCase>
```

It's important for the users to understand the meaning of each child element and attribute so that they can create your own test case and run the application successfully.

ID: the identifier of each use case by the application, so it should be unique. It should not necessarily be numbers, so both "UseCase1" and "1" are valid IDs.

Description: shows description and comments of the test case to be created.

MSISDN: the subscriber ID of to be charged from, normally a mobile number.

ChargingType: in the test case, what kind of charging operation to be performed. Put **Session** or **DirectDebit** here.

UnitType: determines which unit type will be used for charging. Put one of the four types here: Event, Time, Volume, and Refund. Note that **Refund** is used for returning money to the user's account and can only be used when the **ChargingType** is set to **DirectDebit.**

URI: indicates where the charging application is hosted and tells the charging server where to return the charging reply. It is in the form **aaa://IpAddress:PortNumber**.

Realm: diameter makes use of the realm, also loosely referred to as domain, to determine whether messages can be dealt with locally, or whether they must be proxied.
DestinationRealm: the realm of the charging server.

DestinationHost: indicates where the charging server is hosted and tells the charging client where to send the charging request. It is in the format **aaa://IpAddress:PortNumber**.

ServiceName: the name of the service the user requested and stored in the tariff list of the charging server.

ServiceProviderID: the ServiceProviderID of the service the user requested and stored in the tariff list of the charging server

TrafficCase: will be set to 20 for the Originating Service Charging (normal end-user initiated) and 21 for Terminating Service Charging. In this application, it is always set to 20.

RequestedServiceUnit: tells the charging server how many service units the user requested.

UsedServiceUnit: For Session-based charging only! Tells the charging server how many service units are used. The server can then charge the user based on the number of the requested units and the used units.

InterimIntervalInSecond: For Session-based charging only! Determines the time interval between sequential interim requests. Note that the measure is in Second.

CurrencyName: For use case DirectDebit/Refund only! Determines which currency to be used for refunding the user. Fill in with one of the three currency names used in this application: SEK, USD and EUR.

Among the parameters mentioned above, only the following are related to the Charging API to form the charging request message that is sent to the charging server:

- MSISDN
- DestinationRealm

- DestinationHost
- URI
- ServiceProviderID
- TrafficCase
- RequestedServiceUnit
- UsedServiceUnit
- UnitType
- ChargingType
- CurrencyName

The remaining parameters, shown below, are only for maintenance, display and accounting purposes.

- ID
- Description
- ServiceName
- InterimIntervalInSecond

Note that all the child elements of the <useCase> element are **not** case sensitive. In the event a certain element is not used in the test case, just **leave it blank**. For instance, leave the CurrencyName element like this “<CurrencyName></CurrencyName>” when a Direct Refund use case is not going to be created.

5.4.5 Create a new test case

Suppose the application wants to charge John Smith for his requested “CallSweden” service using the example application. This is quite simple. Just add a new use case to the testCase.xml included in the package.

Please note that each use case must have a unique use case ID that can be either a number or characters.

```
<enabler>

<charging>
    <useCase id="" description="">
……….
    </useCase>

   <useCase id="9" description="A call within sweden">
          <MSISDN>0737762157</MSISDN>
          <ChargingType>Session</ChargingType>
          <UnitType>Time</UnitType>
          <URI>aaa://localhost:9998</URI>
          <Realm>realm.com</Realm>
```

```
            <DestinationRealm>test.com</DestinationRealm>
            <DestinationHost>aaa://localhost:1812</DestinationHost>

            <ServiceName>CallSweden</ServiceName>
            <ServiceProviderID>590</ServiceProviderID>
            <TrafficCase>20</TrafficCase>

            <RequestedServiceUnit>5</RequestedServiceUnit>
            <!--UsedServiceUnit and InterimIntervalInSecond are used
ONLY for session based charging !-->
            <UsedServiceUnit>5</UsedServiceUnit>
            <InterimIntervalInSecond>5</InterimIntervalInSecond>
            <!--the CurrencyName is only used for DirectDebit for
REFUND-->
            <CurrencyName></CurrencyName>
    </useCase>
</charging>

</enabler>
```

Attention should be paid to some important elements. First, as mentioned, a phone call is a charging session based on used time. So set the ChargingType element to Session and UnitType to Time. The InterimIntervalInSecond element determines how often your application contacts the charging server. If it's set to 5, then your application contacts the charging server every five seconds. For the phone call, the developer can estimate that each UsedServiceUnit is equal to the InterimIntervalInSecond so that the developer can set them to the same value. But in other cases, such as file downloading, the developer has to develop another function to supervise how many service units have been used before sending out each interim request.

5.4.6 Start the test application

From the command line window, execute the run.bat file

What are going to be displayed are the welcome message, the instructions and the abstract information of the testCase.xml file before the application prompting the user to input the ID of the use case the user wants to test.

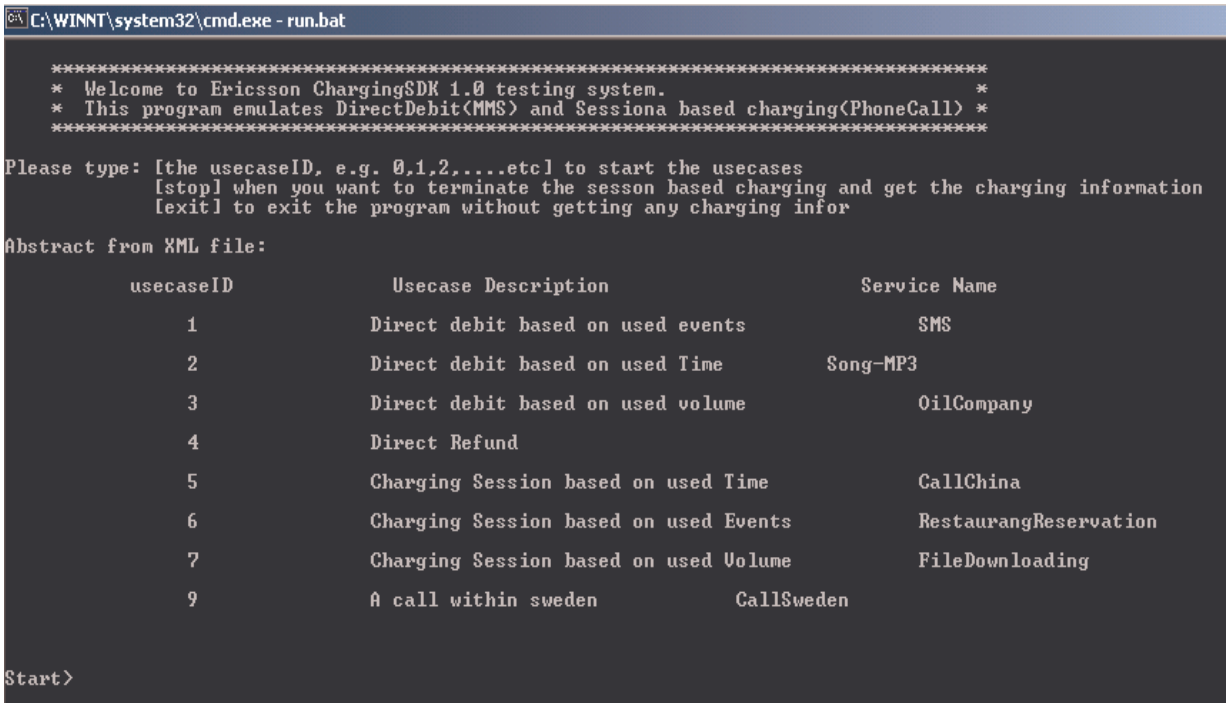

Figure 5-3 Application snapshot

Once the application is started, the abstract of the testCase.xml file will be shown. The user can choose one test case to test based on its description and service name.

5.4.7 Start the selected charging test case

After typing 9, the ID of the test case just created, the charging session will continue.

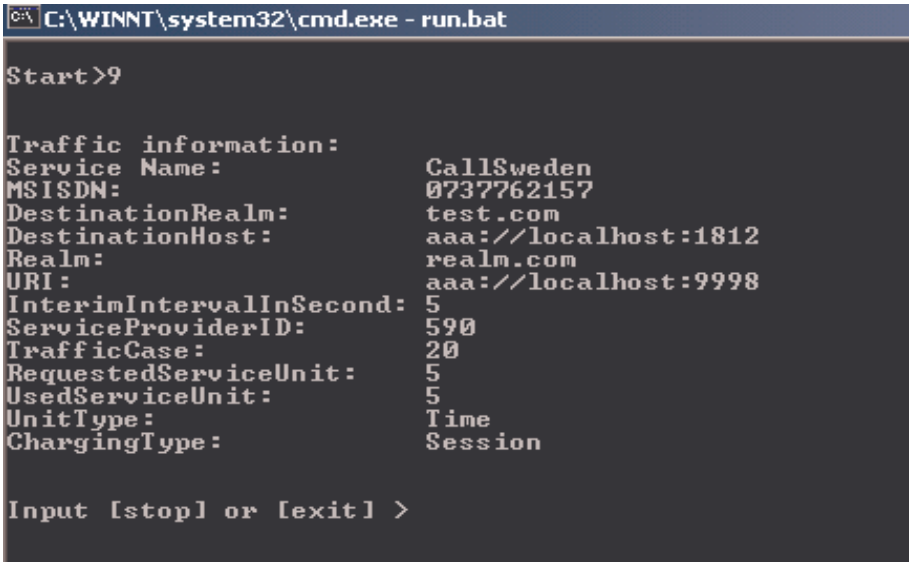

Figure 5-4 Start one usecase

Given the test case ID, the traffic information shown here is read from the testCase.XML file.

5.4.8 Terminate the charging test case process

The user can just input stop when The user want to terminate the process and retrieve accounting information such as the duration of the session and the final cost for the user.

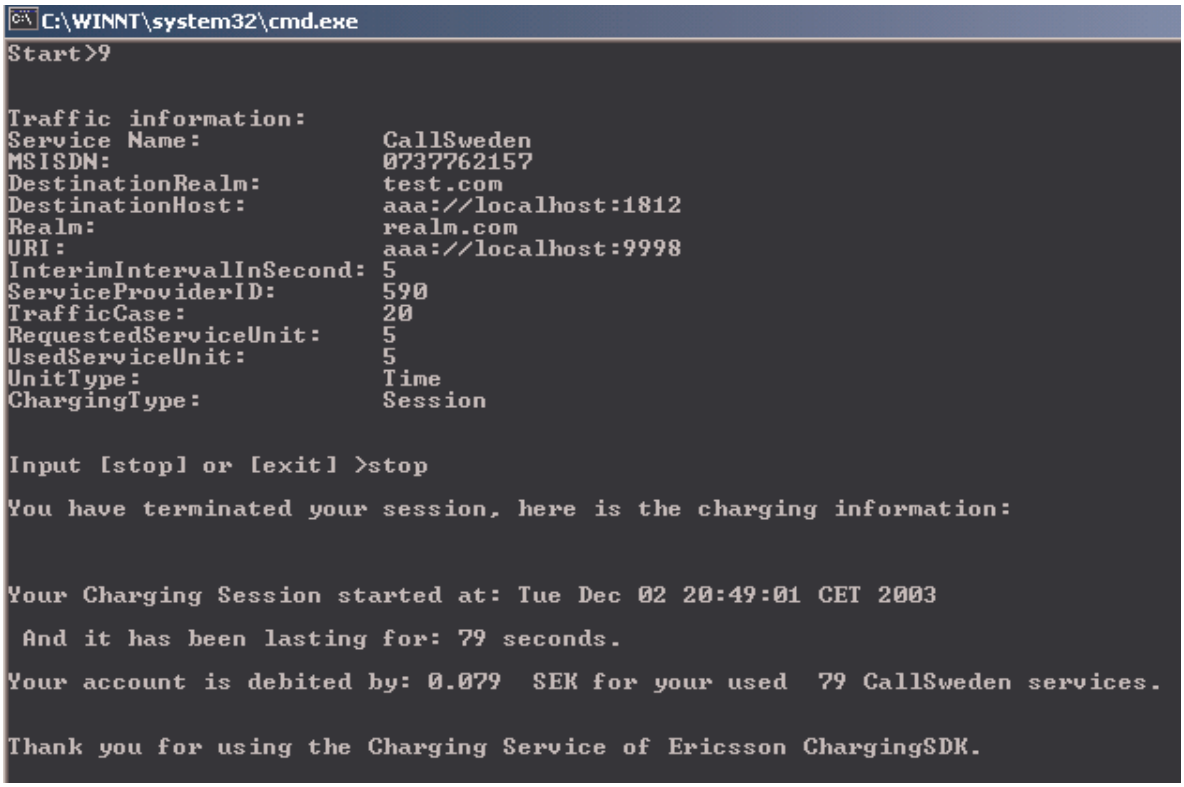

Figure 5-5 Terminate one use case execution

Please note, only the cost for the end-user, which is “0.079” here, and the currency code of “SEK” are received directly from the charging server. In the application, the currency codes are translated into the currency names.

6 Charging High Level API

Charging High Level API is built on and backward compatible to the low level Charging API 1.0 D12E, as shown in Figure 6-1

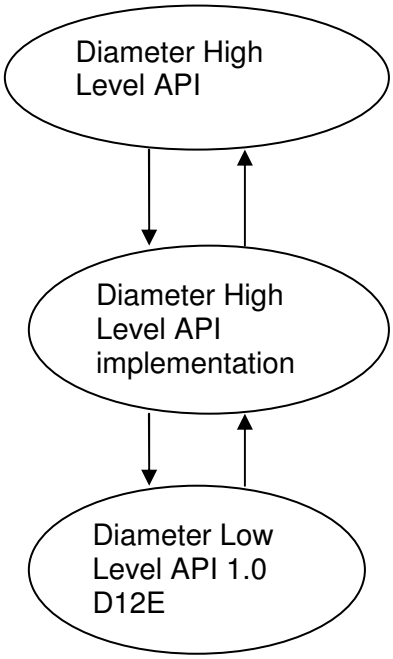

Figure 6-1 High Level API structure overview.

Diameter High Level API provides a simplified, use-case based programming interface structure for Diameter Charging. In the new design, design patterns are applied as an important design principle. Bridge (factory) design pattern is used for creating charging use-case based object and The Singleton Pattern is used for managing Diameter stack and ensures that within one JVM instance there is only one Diameter stack object running.

Diameter High level API is structured in the package:

com.ericsson.pps.diameter.api.scap.usecase

Diameter High Level API implementation is grouped in the package:

com.ericsson.pps.diameter.api.scap.usecase.impl

6.1 Design principles

6.1.1 Object oriented design

In order to take advantage of object oriented features of Java programming language, the design of High Level API follows strictly the OOP principles. Object Inheritance and interfaces abstraction are two main OOP features implemented in the design. Figure 6-2 illustrates the detailed class structure of the High Level API.

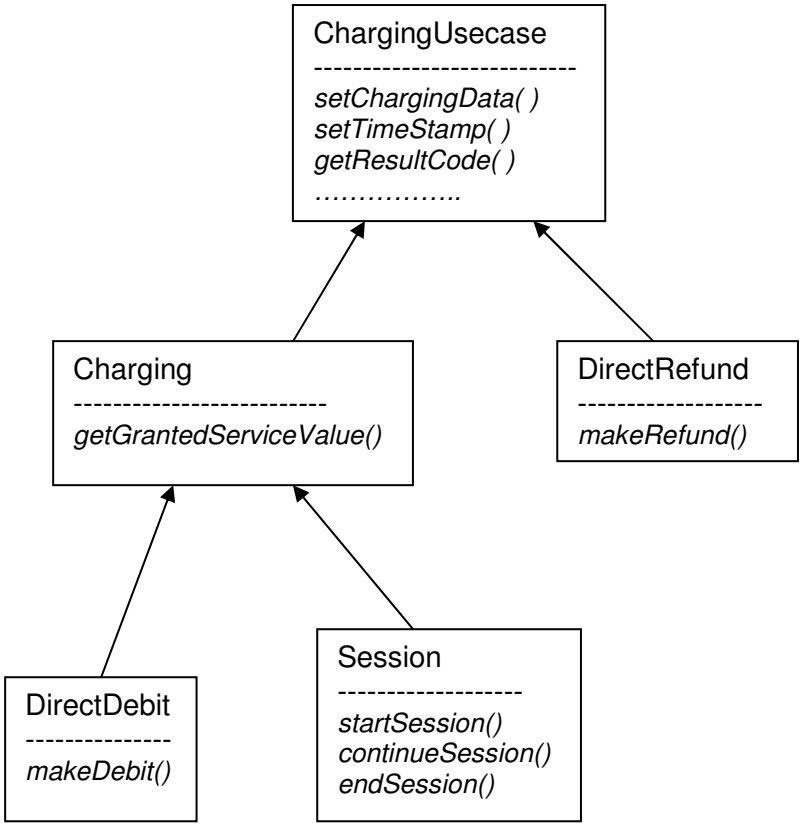

Figure 6-2 Class Inheritance structure of High Level API

6.1.2 Singleton design pattern

Software Design patterns is a design concept which aims to create reusable and well- structured software components for a certain common problems. Singleton Pattern is one of the simple design patterns which ensures only one object instance is created and can be accessed through public method invocation. Figure 6-3 illustrates the main concept of Singleton Pattern.

```
SingletonClass
--------------------
oneInstance
SingletonClass getInstance()
```

```
SingletonClass getInstance()
{
   If (oneInstance == null)
          oneInstance= new
SingletonClass();
   return oneInstance;
}
```

Figure 6-3 Singleton Design Pattern

Singleton Pattern is applied in the ChargingServerConnection class to maintain the initializing and stopping of Diameter stack, and to create or delete static routes to different charging Servers identified by the URI and Realm, as shown in Figure 6-4. This design ensures that only one static route is added for each charging server.

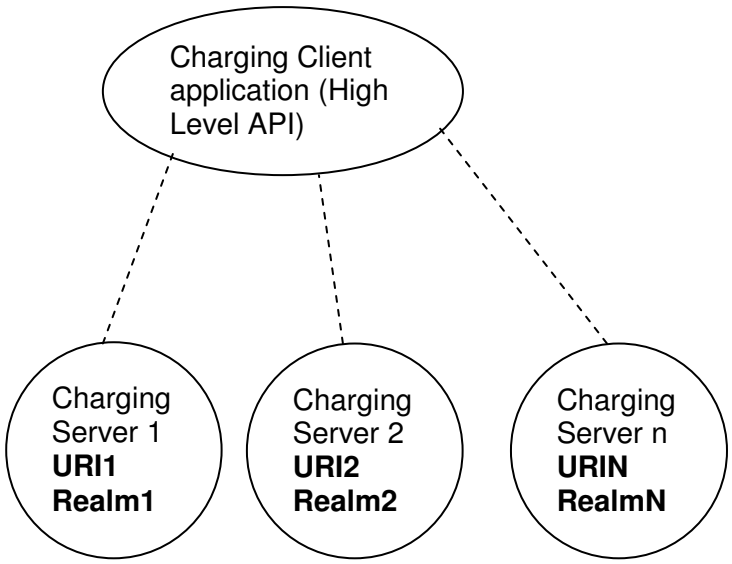

Figure 6 - 4 Static Routes to Charging Servers

The code skeleton of the ChargingServerConnection class is shown as below:

```
package com.ericsson.pps.diameter.api.scap.usecase;

import com.ericsson.pps.diameter.api.scap.usecase.impl.ChargingExceptionImpl;

import
com.ericsson.pps.diameter.api.scap.usecase.impl.ChargingServerConnectionImp
l;

import java.util.Hashtable;

import java.util.Enumeration;

public class ChargingServerConnection

{  private String chargingServerURI = "";

   private String chargingServerRealm = "";

   private String chargingServerKey = "";

   private int applicationID;

   private int vendorID;

   private ChargingServerConnection() {   }
```

```
    public static ChargingServerConnection createConnection( String
chargingServerURI, String chargingServerRealm, int applicationID, int vendorID)

  { chargingServerURI = chargingServerURI.trim();

    chargingServerRealm = chargingServerRealm.trim();

      // Fetch connection. Null is returned if it does not exist.

    ChargingServerConnection connection =
ChargingServerConnectionImpl.getConnection( chargingServerURI,
chargingServerRealm, applicationID, vendorID );

       // Create new connection if it does not exist

    if( connection == null )

    {  connection = new ChargingServerConnection();

      connection.applicationID = applicationID;

      connection.vendorID = vendorID;

      connection.chargingServerRealm = chargingServerRealm;

      connection.chargingServerURI = chargingServerURI;

      try

      {       ChargingServerConnectionImpl.createConnection( connection );

      }
      catch (ChargingExceptionImpl ex)

      {       throw new ChargingException(ex.getMessage());

      }   }

    return connection;

  }  public void closeConnection() {  }

  public void closeAllConnectionToRealm()  {  }

    public static void shutDownStack()

  {    ChargingServerConnectionImpl.shutDownStack();

  }

  public static boolean startStack( String ownURI, String ownRealm, String
productName, String ownIP )

  {    return ChargingServerConnectionImpl.startStack( ownURI, ownRealm,
productName, ownIP );

  }}
```

The **createConnection** method returns a **ChargingServerConnection** instance by either rerurning the existing one or otherwise creating a new one. The bold code paragraph is the main implementation of this method.

6.1.3 Bridge (factory) design pattern

The Factory design pattern consists of one interface used for creating objects. The subclass that implements this interface determines the concrete content of the object. The Figure 6-5 illustrates the class structure of the Factory Pattern.

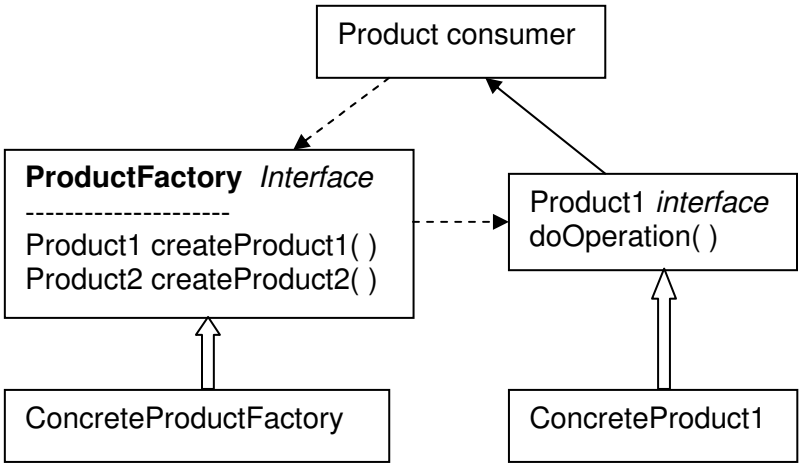

Figure 6-5 Factory Design Pattern

In the design of High Level API, **ChargingUsecaseFactory** is the Factory interface used for creating charging usecase objects. The code skeleton is written as below:

```
package com.ericsson.pps.diameter.api.scap.usecase;
import com.ericsson.pps.diameter.api.scap.usecase.ChargingUsecase;
public interface ChargingUsecaseFactory

{
  public DirectDebit createDirectDebitEvent(String subscriberID);

  public DirectDebit createDirectDebitVolume(String subscriberID);

  public DirectDebit createDirectDebitTime(String subscriberID);

  public DirectRefund createDirectRefund(String subscriberID);

  public Session createSessionEvent(String subscriberID);

  public Session createSessionVolume(String subscriberID);

  public Session createSessionTime(String subscriberID);
}
```

In **com.ericsson.pps.diameter.api.scap.usecase.impl** package, **ChargingUsecaseFactoryImpl** class is the class that implements the **ChargingUsecaseFactory** interface and creates the concrete usecase object. Figure 6-6 illustrates how Factory Pattern was applied from class level.

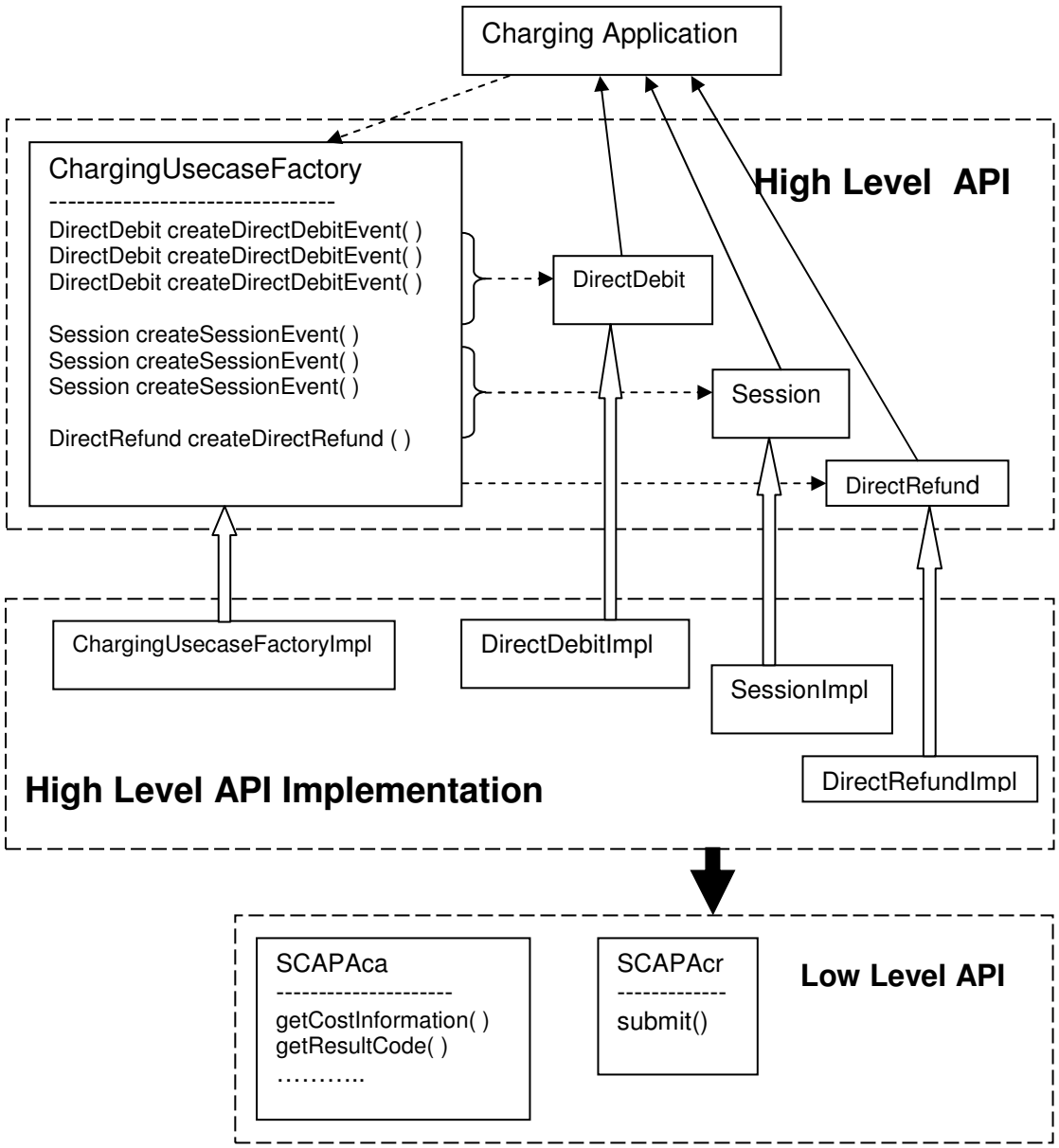

Figure 6-6 Factory Design Pattern applied in High level API

6.1.4 Use case based object structure

The Diameter Charging Low Level API supports the following basic use cases:

- Direct Debit based on used Event / Volume / Time

- Direct Refund
- Charging Session based on used Event / Volume / Time

Direct Debit is a simple event that can be understood as an atomic transaction. Direct Debit is chosen by the application when the full scope of the service to be delivered is known before the delivery and the application will not supervise the delivery, for example, sending an MMS. And Direct Refund is chosen when it's necessary to return a certain amount of money to the user's account.

At a Charging Session, the final cost of the service is not known at the start.
Charging Session is chosen when the full scope of the service to be delivered is not known before the delivery and the application will supervise the delivery. For examples, a phone calls or files downloading service.

When using low level API, there is no corresponding class for each use cases, instead, the developer needs to configure a certain parameters in order to structure one use-case, for example, to create a charging request for **Direct Debit based on used Event** use case, we need the following code:

```
SCAPAcr acr = new SCAPAcr( myMSISDN, SubscriptionId.END_USER_MSISDN
);

//Set the Accounting-Record-Type to EVENT
acr.setAccountingRecordType(SCAPStack.EVENT_RECORD);

//Set the Requested-Action to DIRECT_DEBIT
acr.setRequestedAction(SCAPStack.DIRECT_DEBITING);
```

While using use-case based High Level API, the developer can create Direct Debit based on used Event use case using the ChargingUsecaseFactory interface as simply as below:

```
ChargingUsecaseFactory myFactory = ChargingFactory.createUsecaseFactory();

DirectDebit myDebit = myFactory.createDirectDebitTime(myMSISDN);
```

6.2 Conclusion

Diameter Charging Low level API meets only the basic requirement of building J2SE applications to perform charging operations towards Ericsson PPS. But it brings coding complexity to the developers. Because of its Thread- embedded Diameter Stack, Low Level API could not be used in EJB components in the J2EE framework. A Resource Adaptor is needed for this purpose

Diameter Charging High Level API was created to provide a more simplified use-case based factory structure for creating use case objects. It also applied singleton design pattern in managing Diameter stack and static routes. The API also provides a standard interface for integrating the additional Resource Adapter into J2EE framework.

Diameter Charging SDK 1.0 D13E is design based on the proposed High Level API structure and could be downloaded from Ericsson Mobility World Website.

7 Evaluation of Ericsson NRG SDK

The Ericsson Network Resource Gateway (NRG) is a Service Capability Server, which provides service capabilities to the application in a secure way. The NRG has built-in security that makes sure that access to the telecom network elements is done on business terms and in a secure way **[Ref 8]**. Ericsson NRG SDK provides both CORBA and Java abstract interface for accessing the Ericsson NRG Server capabilities.

NRG API is built on HOSA (High level Open Service Access) interface based on the Parlay/OSA (Open Service Access) standard interface. It allows the developers concentrate on the application logic only because the provided interface is unified and independent of underlying telephony networks, as shown in Figure 7-1 **[Ref 9]**.

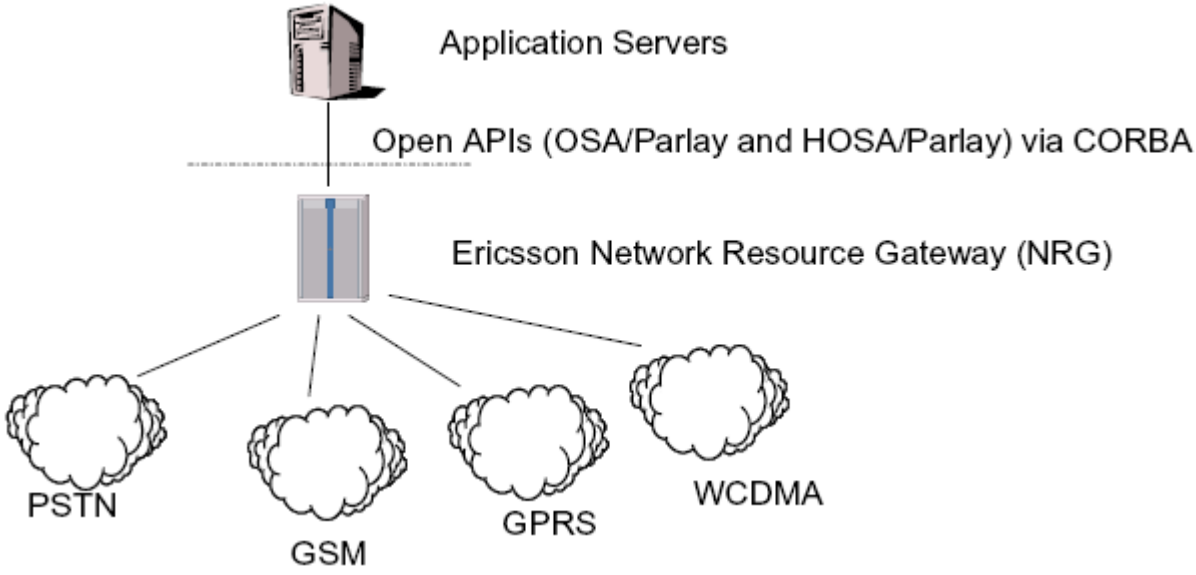

Figure 7-1 Independent underlying networks

7.1 Inside the SDK

Ericsson NRG SDK offerings include:

- HOSA standard Core API, proprietary Utility API, and Test API for developing application simulating NRG.
- NRG simulator
- NRG predefined sample client automated test tool
- User's guide and API javadoc documentation

The evaluation work is based on these items listed above.

7.1.1 Protocol and standards

Parlay/OSA is an open API (Application Programming Interface) for current and next generation communication networks. It is a joint effort of 65 companies in the IT and telecom industries. Parlay/OSA is based on the most common open standards such as JAVA, CORBA, IDL, UML and Web Services. Parlay/OSA plays an import role in the leveraging IT services with telecom services by integrating telecom network capabilities with IT applications via a reliable, secure and billable interface.

Parlay/OSA is designed to be network and technology independent so it can be used for the development of current mobile network services, current fixed telecom network services and next generation IP-based network services. The creation of Parlay/OSA enhances the capabilities of the current mobile data services by enabling the development of new value added services with the technologies that are widely used in the IT developer community, for example, Open APIs, J2SE/J2EE technology, web services and XML. Operators, service providers and content providers will gain long-term revenue for the introduction and wide adoption of Parlay/OSA all around the world.

HOSA is the Ericsson superset of Parlay/OSA and is the pre-standard extensions to Parlay/OSA with additional services and functionalities. HOSA is the interface to access the Ericsson Network Resource Gateway (NRG), which acts as a Service Capability Server by providing service capabilities to the HOSA applications. HOSA also provides flexible alternatives for developers to either program with Java language through the NRG JAVA API or with other language choices through CORBA to communicate with the NRG directly, as shown in Figure 7-2 [Ref 10].

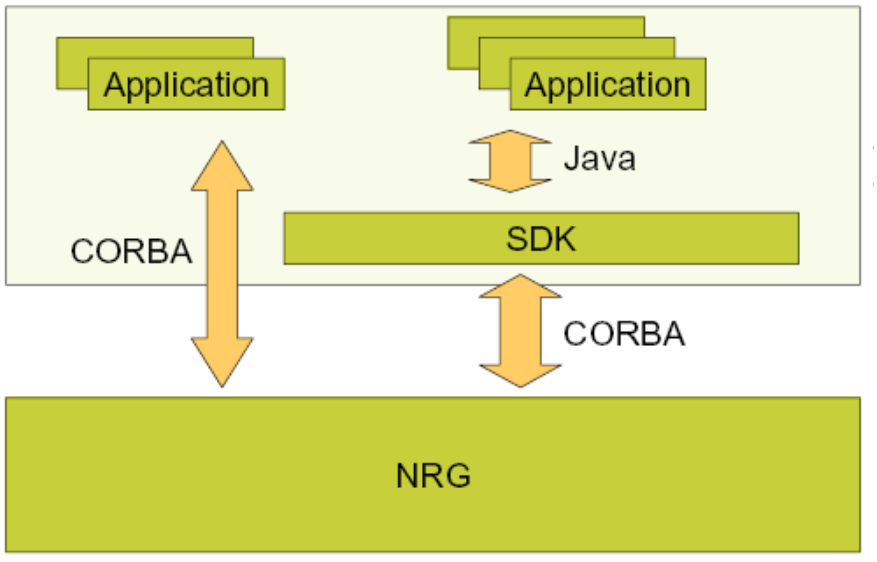

Figure 7-2, Building applications based on CORBA or NRG API.

7.2 Evaluation Overview

In this section, the author will give the overview of the evaluation results of the NRG SDK in Section 2.1 and detailed evaluation results of the separate part of the whole NRG offering will be covered in the later sections.

7.2.1 Pros

Innovative technical concept

Parlay API is an open standard API and technology independent. It makes it possible to combine the successful IT application development model with the telecom network capacities. This makes Parlay attractive and beneficial to a wide range of market players, which includes operators, service providers and application developers. Parlay is also a future-proof technology because it can function across multiple networks and supports both current and the next generation 3G networks with the same API.

Simplified Java API abstraction

NRG SDK provides the developer with the high level abstraction of the underlying CORBA protocol. It hides much of the telecom complexity from the developers so that lowers the knowledge curve for the developer to start developing the NRG applications.

7.2.2 Cons

Rearrangement of the package structure is recommended.

The current Ericsson NRG SDK contains the following four packages for the developers to download separately:

- Documentation
- JAVA software Libraries
- NRG Simulator
- Automated Test Tool and Example Applications

Merge the Libraries and the Example applications into one package

It is recommended that JAVA software libraries and the Automated Test Tool and Example Applications be integrated into one package. There are two reasons for doing this. First, currently, after downloading the two separate packages, the SDK users have to integrate them into one package anyway since the example applications and automated test tool have to include the SDK libraries into the code. Second, the merging of the two packages will reduce the effort of the developers for downloading and installing the SDKs. Therefore it will make the product more robust and easy to use.

Move the Programmer's guide from the documentation package to the nrgsdk package

Programmer's Guide is the most important documentation for the developer to get started. So it's better to include the Programmer's Guide into the nrgsdk package.

Optional: Move the documentation and emulator package into the nrgsdk

Just like other Ericsson Service Enabler SDKs, the suggested package structure is depicted as following:

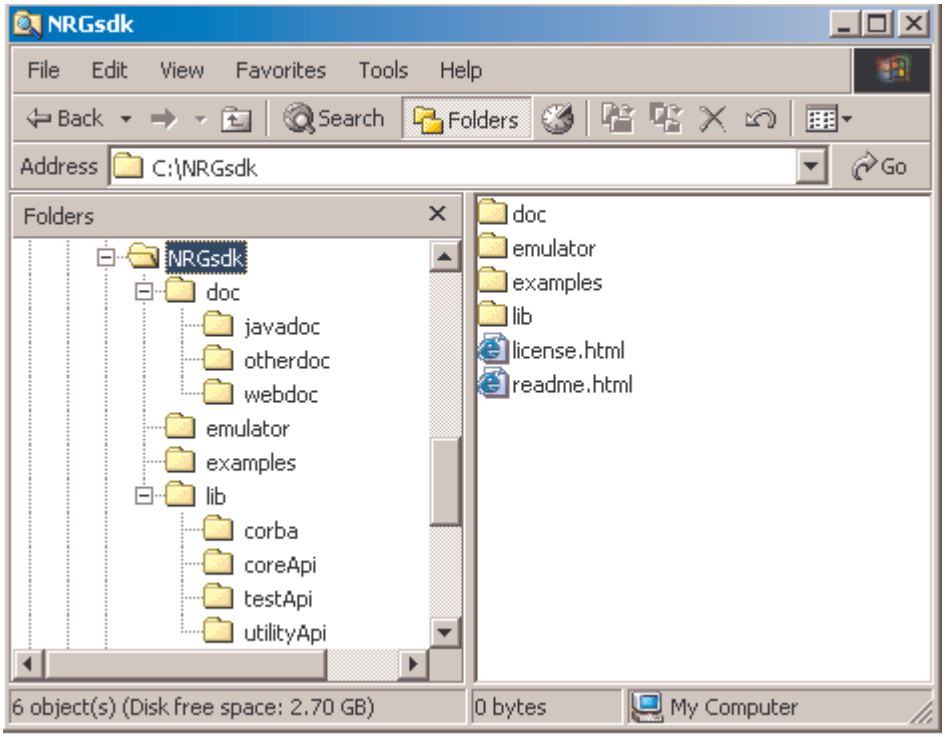

Figure 7-3 Suggested package structure

Integrated Look& Feel of the whole SDK documentation as other Ericsson Service Enabler SDKs is recommended

Despite of the convenient and powerful programming libraries and emulator, the packaging and documentation of the SDK package are not organized. A well-structured and organized product package as well as documentation will give the end user a good and professional impression of the product.

There is a good example out there for SDK documentation. All the Ericsson Service Enabler SDKs have a common and integrated Look& Feel, the example showed below is the picture taken from the Ericsson Diameter Charging SDK web documentation.

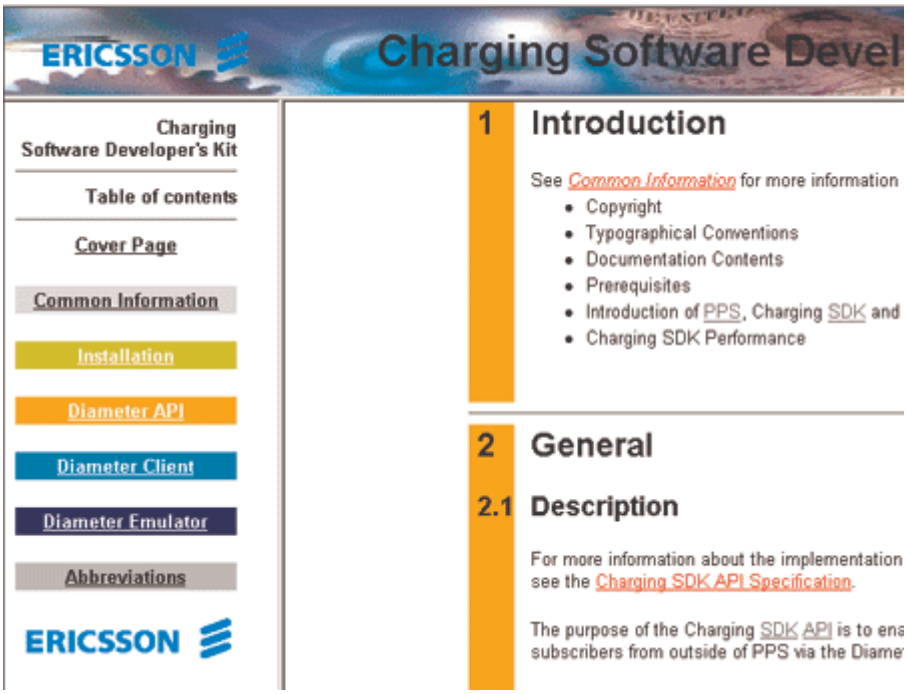

Figure 7-4 Ericsson Diameter Charging SDK documentation Look& Feel.

So it is suggested that a common web documentation interface with navigation frame is used to link all the fragmental documentations contained in the NRG SDK package. This will save the developer from having to click into different “doc “directories to look for information.

Typographical and grammatical errors are discovered and corrected

There are several obvious grammatical errors found in some documentations contained in NRG SDK. Though this should concern as an important part of the evaluation, the author still suggests all the documentation should be viewed by a professional English editor to ensure a good quality of the whole SDK offering. Some errors the author found is documented separately from this report and will be sent to the responsible designer.

7.2.3 Time needed for studying and putting the SDK in use

In this section the approximate time and effort consumed in studying and using NRG SDK is stated. This is only for reference purpose and the time might vary from person to person.

Preparation

This step includes downloading all the needed NRG SDK packages from Ericsson Mobility World and the installation of all the libraries and tools. Besides, surfing the net for some common information about Parlay is also carried out during the preparation phase. The author visited the www.parlay.org quite often and subscribed the newsletter to get the up-to-date information about Parlay.

The time needed for this step is one working day, that is, 8 hours.

Programmer's Guide and example applications

The programmer's guide for NRG SDK is just like a book so that more effort is required to fully understand the context. From the Programmer's Guide, the developer can not only learn how to use the APIs but also can learn some very good design concept for Java programming such as the synchronous/asynchronous scheme and the interface adapter.

The time needed for studying the Programmer's Guide is 5 working days, that is, 40 hours.

Since part of the design principle of the example applications is covered in the Programmer's guide, so it takes not long time to get familiar with the example applications. But it might take longer time to implement a test tool, which simulates a single NRG node, using the testapi included in the NRG package.

Time needed for example applications is three working days, that is, 24 hours.

Simulator

It's an enjoyable experience to test the applications with the simulator and the included pre-defined applications.

This step takes approximately two working days, that is, 16 hours.

In total, it will take a moderately experienced Java developer 16 working days from scratch until being able to develop business applications using NRG SDK tools. And also, the developers assumed to have the common knowledge in Mobile communication field beforehand. That means that for a fresh developer that is new to some common concepts in mobile communication field, the pre-study step will take longer.

7.3 NRG libraries

The NRG libraries package contains four main library packages: corba, coresdk, utilitysdk, and testsdk. These are the most important parts of the NRG SDK product. So they should be well delivered and documented.

7.3.1 Pros

Abstraction from the underlying protocols

- **Abstraction of Core API from CORBA**

  The Core API translates all Java data types and method invocations into CORBA data types and method invocations. This makes the complicated underlying protocol such as ORB (Object Request Broker) and IIOP (Internet Inter-ORB Protocol) transparent to the client application. So the developer with little knowledge in CORBA can still develop innovative applications using the Core API.

- **Abstraction of Utility API from Core API**

  The Utility API simplifies the application development by offering abstraction from the Core API, which provides a pure HOSA interface towards client applications. An invocation of one method through the Utility API has the same result as invocation of a fair amount of methods through the Core API. This makes the development of NRG application more efficient.

Flexibility of coding with CORBA directly

This feature gives the advanced developers more control and flexibility. For example, programming languages other than Java could be used to access the services provided by NRG. This proves the Parlay/OSA a real technology and protocol independent interface.

Very useful adapters for callback interfaces

The adapter provides default implementation of all the callback methods defined in one interface. By using this adapter, the developer just need to overload the method that is needed instead of having to implement all the methods contained in that interface. This clever design saves time and coding effort of the developer.

7.3.2 Cons

Change the sub-package name from XXXsdk to XXXapi

As in each sub-package, only the corresponding API and the java documentation are contained. So it is more appropriate to name the sub-package name XXXapi than XXXsdk.

More elaborate class description in the Javadoc

Although in the Programmer's guide, some important classes and methods are explained shortly, it is still very helpful to give each class a detailed description and a short code fragment showing how the class could be used. The description could include what the class is used for (Not only explanation with flat technical words, but also the connection with the real business case), how to use it by giving a short code stem, and the classes that used in conjunction.

Naming convention of the class names is introduced in Programmer's Guide

The class names of the APIs sometimes are hard to understand. So it's very necessary that in the Programmer's guide, a chapter describes the naming convention of the all the APIs. For examples, what does "Ip" mean in the class name IpAppCallAdapter? What does "Tp" mean in the class name TpAddress?

Using SetParam(paramType paramValue) or similar to avoid some methods having too many arguments

The method that has too many arguments is not convenient for the user, For example:

```
public int hosaSendMessageReq(
IpAppHosaUIManager appUIManager,
TpAddress originatingAddress,
TpHosaTerminatingAddressList terminatingAddressList,
String subject,
TpHosaMessage message,
TpHosaUIMessageDeliveryType deliveryType,
String billingID,
int responseRequested,
boolean deliveryNotificationRequested,
TpHosaDeliveryTime deliveryTime,
String validityTime)
```

This method has 11 arguments. A suggested improvement could be: Leave the arguments that don't have a default value or could not be Null in the method argument list. For the rest of the arguments that are optional, could be null or have a default value, a SetParam(paramType paramValue) model could be used, for example.

Design Patterns are applied in the design of APIs

Implementing design patterns in the APIs will give an elegant and reusable solution. Some commonly used design patterns are: Bridge pattern for separating an object's interface from its implementation, Factory pattern for object creation, Singleton pattern for ensuring only one object exist in one JVM and so on.

Instruction on the integration with J2EE architecture

At the moment, CORBA plays an important role in distributed application development. At the same time J2EE technology becomes more and more dominant in the market. So some questions are raised:

Can NRG APIs be used within the J2EE components, such as EJBs?

To the extent of the author's knowledge on J2EE, the current release of NRG APIs and implementation can not be used by EJB because of the incompliance with the SUN EJB specification. The incompliance lies mainly in the fact that the NRG API involves threads that are unacceptable for EJB container.

One of the possible solutions of this problem is the introduction of the Resource Adapter, which is a J2EE component that implements the J2EE Connector architecture for a specific EIS. So it is strongly recommende that a Resourece Adapter for NRG is provided in the future release of the SDK. Instructions on how to integrate the NRG sdk with Enterprise applications are also expected.

7.4 Programmer's guide

7.4.1 Pros

A very useful handbook for NRG application developing

The Programmer's Guide contains enough information and is the best place for the developer to start the NRG exploration. It uses the use case based example applications to illustrate the usage of the SDK. The code example and sequence diagram give the developer better understanding of the design.

The documentation is structured in a way from ease to difficulty gradually.

Very pedagogical chapter: "Design and Implementation Considerations"

The 7th chapter is an extra harvest for developers. The designer shares his precious experience and consideration on the common design concepts such as threads, parallelism and adapter for interfaces. This will definitely help the developer to develop stable and robust applications.

7.4.2 Cons

More issues about charging to be addressed

Charging is an important part for Mobile Internet Applications. There are only two pages in the 6th chapter (Multi Party Call Control and User Interaction Services) that covers charging issue. There the author cannot find answers for some questions stated as following:

Does only the Multi Party Call Control use case concern Charging? If not, why not put the Charging into a separate chapter to serve all the use cases. If yes, how about the charging schemes of the rest of use cases? Are all the Charging schemes for them are set to “Application cannot influence charging”, which is one of the charging capabilities specified in the Programmer’s guide?

As stated in the Programmer’s Guide, when using transparent charging, the application determines the string to be sent in the charging information message. So what does the charging information message look like and how to compose it using the NRG SDK?

So it is recommended that the answers to these questions could be included in the Programmer’s Guide.

An error in section “2.3 Getting Access to a Service” to be corrected

In Example1 “Obtaining a service”, the obtainSCF() only has one parameter, but it seems like that the obtainSCF() method should have two parameters from the explanation of the example. So the recommended modification of this example is:

```
itsHosaUSManager = (IpHosaUserStatus) itsFramework.

obtainSCF(aResultClass,"SP_HOSA_USER_STATUS");
```

7.5 About NRG Examples

7.5.1 Pros

Well documented and commented code

All the examples are written and organized in a very good manner. The Javadoc of each example has an IDE look& feel, which is very clear and straightforward. The comments of the code help to make the program well understood.

Covers the main use cases

There are 9 example applications included in this package. These examples cover almost all the possible services that the Parlay/HOSA applications can get from the NRG. This helps the developers get a very good profile of the SDK capacities and therefore will facilitate the development of their own business cases based on the example use cases.

Convenient Ant scripts for building the example applications

Using the Apache Ant building tool, it’s convenient to set the classpath centrally and also easy to compile and run the application.

7.5.2 Suggested improvements

Add Charging capabilities to the examples

Charging capability should be implemented and depicted further in the examples.

Recover some broken links in the documentation

A subtle flaw is found that, when browsing index.html in the java doc directory of one example, the page shown in the right frame reads as: “The page cannot be displayed”. This happens to four examples and is depicted below:

- MMSContentRetrieval
- SMSSender
- SMStoEmail
- SMStoMMS

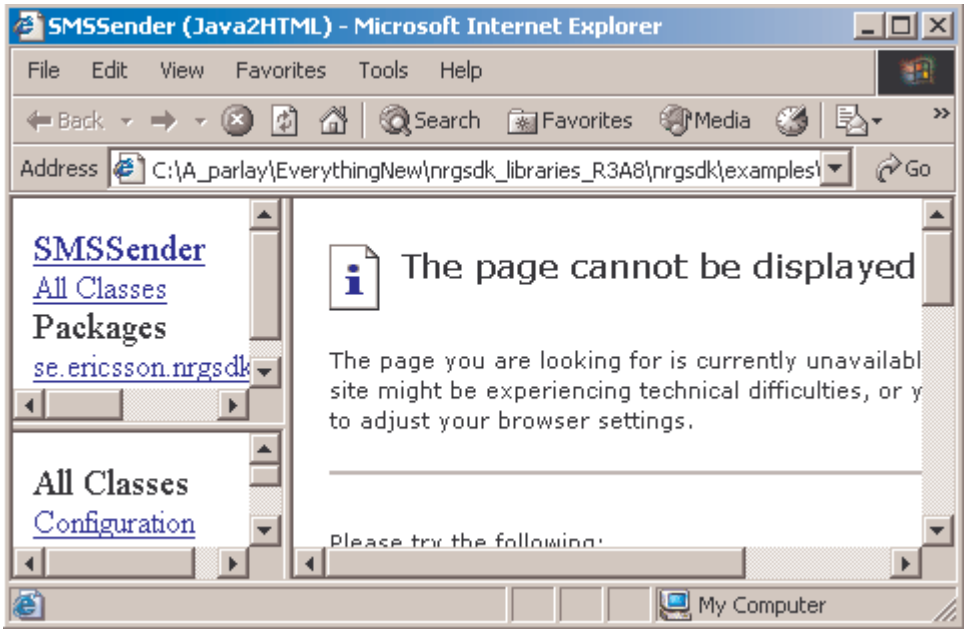

Figure 7 - 5 Broken link illustration

It is recommended that these broken links should be recovered in the future release.

Exception caught in SMStoMMS example should be solved.

The example could not be properly executed because the occurrence of the exception as shown in Figure 7-6

Figure 7-6 Exception of the SMStoMMS example

Script files for Linux and Unix users are added

In the current release of the example applications, there is only .bat file for the Windows users to run the example with or without the automated test tool. In order to make the example package 100% platform independent, it is recommended that the .sh script file for executing the examples within the Unix/Linux platform is added to the package.

7.6 NRG simulator

NGR simulator is a powerful tool for testing the NRG applications without having to set up and invest a real lab environment. The simulator demonstrates very innovative and interesting real life business cases that built on the use cases provided by NRG SDK.

7.6.1 Pros

Very friendly and lively user interfaces and virtual phone

The simulator has a very friendly user interface that makes the tool very easy to use. The virtual phone functionality, which adds great value to this tool, support the simulation of the newest version of Sony Ericsson Mobile phone P900.

Innovative and interesting example applications

The example applications included in the Simulator package demonstrate some of the key capabilities of the Parlay/OSA. Unlike the example applications included in the example package that only illustrate the basic use cases supported by the SDK, the example applications in this package combine several use cases together and illustrate the real world business cases. These examples together with the simulator tool give the developer the first hand experience of the NRG so to grasp the NRG concept deeply.

Convenient map window

The friendFinder application together with the map tool demonstrates the combination of the H-OSA User Location service and H-OSA Message service.

Very convenient Run Window, especially the drag&drop function

The run window gives a central control and management of the example applications. The convenient drag& drop function is very useful for the developer to make their own application added to and tested against the emulator.

7.6.2 Cons

Future compatibility to J2SE 1.4 or above

The emulator must be run against J2SE1.3. It is not reasonable since lots of users only installed the latest version of JDK (1.4 or above). This will force the users to install a lower version of JDK, which is a waste of resources. So the java version should be flexible while not being limited only to 1.3.1_<x>.

It would be also very helpful that during the installation of Emulator, a message about the JDK could be given, for example

“The installation of the JDK1.3 won’t affect other programs to use the JDK with version above 1.3.”

MMS composition function added to the virtual phone

Currently there are three simulated functionalities of the virtual phone: dialup, SMS and Email. Since the MMS service is becoming more and more popular today, it’s better to let the virtual phone have the functionalities of MMS composition, sending and receiving. Then the developers can test their own MMS related application against the simulator.

MMS related example applications to be added

If the virtual phone has the functionality of MMS service, then it’s recommended that MMS related example applications are added to the example directory.

Typical positioning service example to be added

A typical positioning service could be:

The user sends an SMS to service number “007”. And then the “007” service returns the positioning data to the user via SMS or a map via MMS. This example is very similar to the “Guru” contained in the examples.

Charging and cost control function to be illustrated

The charging function applies mainly to the Pre-paid subscriber. So it would be great that the simulator can also simulate the real time charging to provide the cost control for both the NRG application and the end user.

7.7 Additional suggestions

7.7.1 Enhancement to the NRG simulator

It is suggested that later on more and more functionalities be added to the emulator such as the real time charging simulation.

7.7.2 Real world success case be published

If possible, a successful and complete real world business case is shown to the developer community. This success show could include part of the source code of the application, the market feedback and so on. This will help the developer to catch up with the fast growth in both the technology and business world and give them more confidence and passion to learn the SDK.

8 Evaluation of MPS SDK

MPS (Mobile Positioning service) is a new field in telecommunication world and has potential in both mass market and vertical market. Innovative positioning related Mobile Internet applications have great possibilities to be profitable for operators and service providers. Ericsson MPS SDK provides interfaces for creating positioning applications to access Ericsson Mobile Positioning services.

The Ericsson MPS offering is deigned to make the application independent of the location technology. Using the generic Interface, the positioning application can migrate across different type of positioning systems without modification. JML (Java Mobile Location) API supports the positioning services in both GSM and UMTS network.

MPS SDK is built on the most commonly adopted Java related technologies, such as HTTP/POST, XML and TCP/IP. All the positioning requests are sent out in form of XML file through HTTP protocol. XML together with TCP/IP protocol is used for the MPS Emulator to simulate the phone call for the Call related MT-LR (Mobile Terminating Location Request) use case.

8.1 Inside the SDK

Ericsson MPS SDK **6.0.1** contains Java libraries and simulated tools for Java based positioning application development. JML API is the common interface for LBS (Location Based Service) application to interact with Ericsson's Mobile Positioning System, which consists of GMPC and SMPC, as shown in Figure 8-1 [Ref 11]. The JML API hides the complexity of the underlying protocols from the developers. So the programmer with only basic J2EE and JSP/Servlet knowledge can use MPS SDK to produce positioning applications.

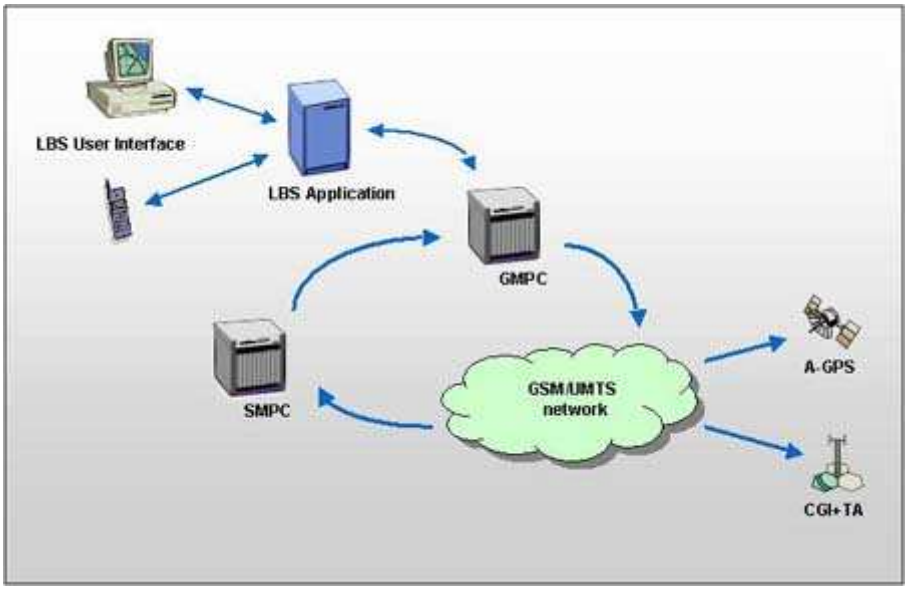

Figure 8-1 Mobile Positioning system architecture

8.1.1 Protocol

Ericsson MPS is built and based on the latest standard interfaces and protocols stipulated by those standardization organizations such as ETSI, 3GPP and OMA. The newest 6.0 release of MPS supports both the standardized MLP (Mobile Location Protocol) and the Ericsson proprietary MPP (Mobile Positioning Protocol).

8.2 Overview

The evaluation result of MPS SDK includes only one evaluation report. Please refer to **Appendix 3: Evaluation report of Ericsson MPS SDK 6.0.1**

In this report, the following problems were addressed:

- Usability and functionality of the MPS SDK
- Pedagogical perspective of the examples and documentation of the SDK
- Time needed for studying different parts of the SDK

8.2.1 Pros

Very good quality of the documentation

All the documentation included in the MPS SDK is written in a very detailed and pedagogical way. The documentation covers detailed user's guide of every SDK component, annotated java doc and very good Tips& Tricks.

8.2.2 Cons

Integrated Look& Feel of the whole SDK documentation as other Ericsson Service Enabler SDKs is recommended.

All the documentations contained in the SDK package are very well written and explains the SDK package clearly, especially all the User's Guide documents which give detailed instruction on how to use the corresponding SDK component.

All the documents are hyperlinked through the index.html page that is put in the root directory of the SDK. As there is a common web documentation template for all Ericsson Service Enabler SDKs, so it's recommended the documentation is structured the same way as the Ericsson Charging SDK and Ericsson MMS SDK. The example showed below is the picture taken from the Ericsson Diameter Charging SDK web documentation.

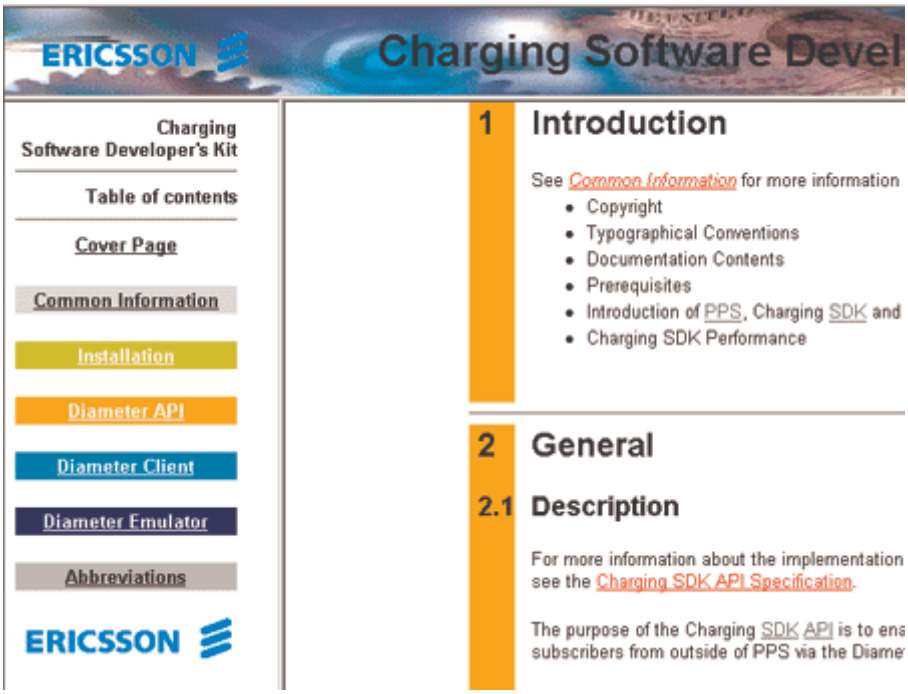

Figure 8-2 Ericsson Diameter Charging SDK documentation Look& Feel

Integration with the J2EE platform

Though the Positioning application built on the current release of MPS SDK could be deployed in the Tomcat web container, the structure that all logic is placed in the presentation layer is not recommended since it will degrade the performance with high overload. So it's necessary to discuss further the possibility to integrate the JML into the J2EE structure, which provides a highly stable and manageable performance and business logic control. In order for the API to interact with EJB, a resource adapter is needed since the MPS API can't be invoked directly from the EJB container. A Resource Adapter that conforms to SUN JCA specification is highly demanded in order for the application to be able to work on variants of J2EE servers in the market such as JBOSS, Weblogic and SUN ONE.

So the designer of the SDK should have rich knowledge in J2EE and take this into consideration when designing the API since nowadays there is a huge J2EE developer community in the market. An Ericsson proprietary resource adapter for MPS is highly recommended to be included in the SDK package on delivery. In the documentation contained in the SDK, the addressing of J2EE with MPS should be elaborated further.

Suggestion for developers on watching MPS web cast

Though the web cast is not included in the SDK package, it is an important and valuable offer associated with the MPS SDK. It abstracts the whole SDK package and presents a big picture of the SDK to the end user from both the business and technical point of view. This educational web cast is very useful for the developer to gain background information before looking into the SDK.

Time needed for studying and putting the SDK in use

Pre-study phase:

Web cast study (including video and ppt documents): 4 hours

Positioning basics: 1 hour

MPS SDK introduction: 1 hour

MPS Emulator together with Example Application: 8 hours

JML API and its Programmer's Guide: 8 hours

Other related documents and specification: 8 hours

Coding and development phase

Approximately 40 hours for intermediately experienced developer (It may vary because of the level of the developer and the complexity of the programmer)

8.3 The JML API

8.3.1 Pros

Adoption of design pattern

The JML API uses the Factory design pattern to handle creation of new objects in a controlled way. For example the creation of the connection to the MPS:

```
connection = (Connection)
connectionFactory.getConnection(username, password);
..... ..... .....

LocationResult result = connection.getLocation(target);
..... ..... .....

connection.close();
```

Managing the connection using the Factory design pattern is more OOP way of design. The connection object can also facilitate the development of the Resource Adapter for managing the connection pool.

For the object distribution over I/O, e.g. RMI (Remote Method Invocation), there are eight classes that implement Serializable interface. These classes contain the content that will be used as arguments by the very important getLocation method in the Connection interface and LocationResult interface. This is also an advantage of this API.

Well annotated and depicted Javadoc

The Javadoc of JML API is amazingly well written and created. For some most commonly used classes, the designer gives the explanation of how to use and code with the class or interfaces by adding text, code segment or figures to the description section of the javadoc of each class. Examples of these classes and interfaces could be: Connection, LocationException and LocationReportServlet.

8.3.2 Cons

Resource adapter for JML is recommended be included in the SDK package

As mentioned in the section 2.1.2 about “Integration with J2EE platform”, a Resource Adapter for MPS is recommended for the SDK package. The Resource Adapter should conform to SUN JCA specification1.5 and could be deployed in all J2EE-compatible application servers.

8.4 The Programmer’s Guide for JML API

The whole document explains thoroughly how to use the JML API. All the instructions given in the context are easy to understand.

8.4.1 Pros

API description chapter gives rich information

This chapter gives illustrated solutions from the system infrastructure point of view. From here, the user can know that the API can’t be used directly from an end user application or a java based mobile terminal application. The JML API can only used either within a LBS (Location Based Service) server trusted by operator or through a middleware server.

This chapter also describes the mapping between the underlying protocol and the JML API. It will help the developer that will use the JML API to have a better understanding the API and the underlying protocol, which is in XML form and easy to read. It will also give a hint to those advanced developers that want to develop the MPS application from scratch instead of utilizing the JML API.

8.4.2 Cons

Annotated example for each use case

The Chapter 6 only annotates a simple MT-LR use case example, which can be compiled and run in J2SE environment.

The exceptions and error code/message are elaborated

Error handling is also an important part of the application development. So it will be valuable if the Programmer's guide could prepare a section that explains about the cause of the exception and how to get the error code/ message as well as how to solve the problem.

8.5 The Emulator

8.5.1 Pros

Well-designed and friendly user interface

Besides the possibility to edit the LBS data XML file by hand, the web based user interface of MPS emulator is another option for the user to edit the LBS data and the Mobile station data. The web interface is very easy to use.

8.5.2 Cons

Can this Emulator be deployed in a J2EE environment?

In the mps_sdk6.0.1\emulator\webapps directory, there is a ROOT. war file included. The author of this evaluation document tried to deploy this .war file in SUN J2EE application server. Unfortunately it didn't work. So it is recommended that in the User's guide, the possibility of integrating the Emulator with the standard J2EE server without modifying the ROOT.war package be introduced.

.txt file be placed by .XML file

Routfile.txt and Celldata.txt are used for the emulator to simulate the routing of base stations and the moving mobile stations. The author of this documentation thinks that presenting these data with XML file will give more scalability and flexibility since Java and XML technology are so tightly combined nowadays. If the Map tool is used, then it is also suggested that the map tool is developed with Java and can generate the data file in XML format.

8.6 The User's Guide for Emulator

8.6.1 Pros

Useful chapter "Trouble shooting"

The trouble-shooting chapter lists all the possible errors that may occur when using the emulator and the corresponding solutions. This is very useful for the users and also gives general pedagogical information on how to do trouble shooting.

8.7 About the example applications

8.7.1 Pros

Well designed and use case based examples

The four examples cover all the four use cases supported by the JML API. The examples are designed in a pedagogical way to show the developers how to design MPS application combining MPS SDK with the advanced technologies. The example application, together with the Emulator can definitely demonstrate the powerful capacities of the Ericsson MPS system.

8.8 The programmer's guide for Example application

8.8.1 Pros

Well organized and structured handbook for the examples

The programmer's guide for example application gives enough information on how to install and run the example applications. Following the instructions, the user could definitely deploy and try the examples successfully. The detailed explanation on even every single fields and button of the user interface eliminate the ambiguity and misunderstanding to a great extent.

Good discussion about the application architecture

Instead of just explaining the plain API example, the addressing of the design issue is very useful and valuable for the users. "This architecture is not recommended for applications with high load, since all logics are placed in the presentation layer" gives a frank statement about the example application. Then the example designer recommended a more correct J2EE application structure, which is more suitable for the application to be deployed in, though the figure about the J2EE structure needs some modification, which will be discussed in the later chapters.

8.8.2 Cons

Correction on Figure 2 regarding J2EE architecture

In the section 3.2, a recommended application structure, which is the combination of the JML API Phone Exchanger and J2EE environment, is depicted, as shown below.

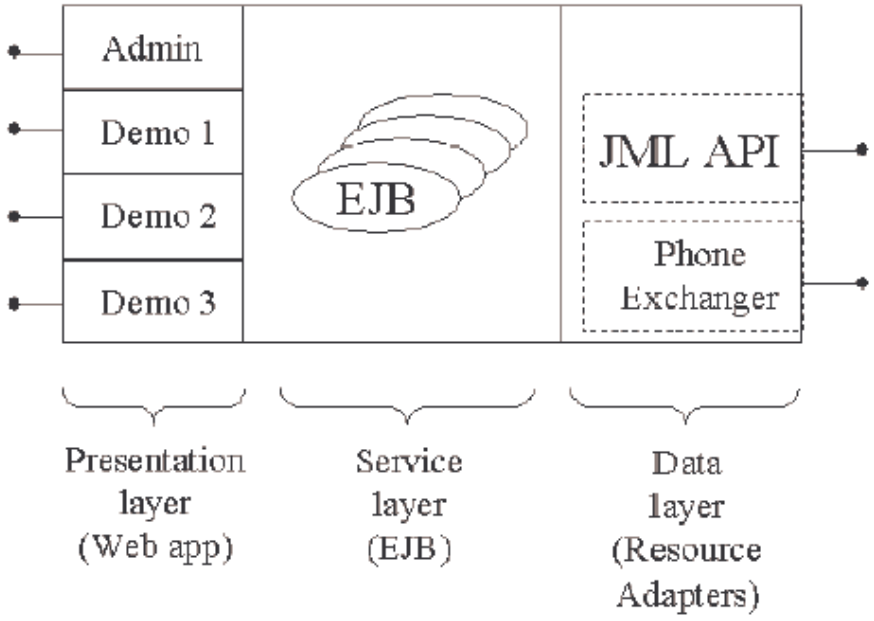

Figure 8-3 J2EE architecture included in the Programmer's Guide

It's not correct to call the JML API and Phone Exchanger as Data layer (Resource Adapters). As defined by Sun, a resource adapter is a J2EE component that implements the J2EE Connector architecture for a specific EIS (Enterprise Information System). The purpose of a resource adapter is to provide a standard API through which an application can access the resources that reside outside the J2EE server. And sometimes, the resource adapter is used for the application to interact with the software components that don't conform to the EJB restrictions so that couldn't be used within the EJB container. The structure of J2EE together with Resource Adapter is shown in Figure 2-4 [*ref 3*].

For MPS, a preliminary structure of integration with J2EE could look like Figure 8-4 as shown below.

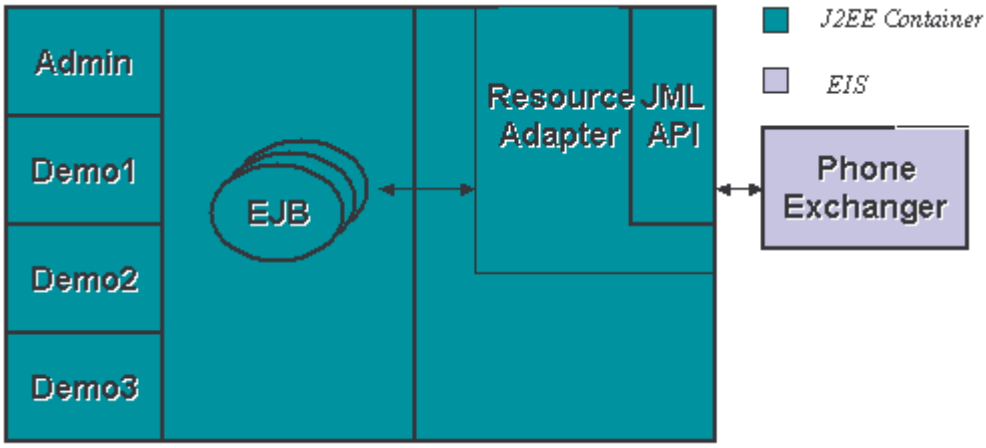

Figure 8-4 proposed integration of JML and J2EE

For MT-LR example, location of a range of MSISDNs could be retrieved

The current example is only applicable to individually added MSISDN. While JML API supports the retrieving of location data of a range of MSISDNs, so it would be perfect if this capacity is added to this example.

Enhancing readability of the examples by Design Diagram

The Programmer's Guide only shows the user how to use the example application while didn't explain the designed detail from code level. So it is recommended that the code design is explained through the sequence diagram, which is a easier way for developers to understand the interaction between the user interface and the API in a good way.

It is also recommended that some important classes or files could be highlighted and explained in the documentation.

Comments added to the source code

Unfortunately, when reading through the whole source code package of the example code, very few comments could be found. A well commented and annotated program will give the user a better understanding of the program and also will be useful for the designer to modify the code later on.

So it is strongly recommended that all the source code to be commented with easily understandable comments.

9 Conclusions and future work

Through the evaluation process of Ericsson SDKs, I have grasped a wide range of know ledges, which include OOP/Java programming skills, Mobile Internet application design, various Internet standards and protocols, and Internet& telecom network architecture.

The imperfection of this thesis work is that I haven't got enough time to design one mobile internet application which integrates the capacities of all of the three SDKs to demonstrate a total solution applicable to current mobile internet technologies.

9.1 Summary

From the evaluation results of the three SDKs, we can see that all the SDKs lack the possibilities to build applications that can be deployed into J2EE framework, which plays very important roles in the design and deployment of reusable, scalable and maintainable enterprise software. So the design organizations should consider seriously about providing Resource Adapter along with the SDKs for enterprise application development.

9.2 Major results

This thesis project has achieved the following results and achievement:

Three evaluation reports which emphasizes the flaws of the SDKs and how they could be improved:

- Evaluation Report of Charging SDK 1.0 for PPS 3.6
- Evaluation report of Parlay SDK R3A8 for Ericsson Network Resource Gateway
- Evaluation report of Ericsson MPS SDK 6.0.1

Two accepted and published improvements to Diameter Charging SDK

- **CDK (Complement Development Kit V1.0**) for Charging SDK 1.0 D12E
- **High Level API** structure and implementation applied in Charging SDK 1.0 D13E

9.3 Future work

Mobile Internet market is growing at a considerable rate because of the vast popularity of mobile phones and high speed internet. It is an inevitable trend for the combination of Internet and telephony services, for example, the growing market of VoIP (Voice over IP) or IMS (IP Multimedia Sub-system) applications. So it's important to provide SDKs which are ease of use and also applicable to enterprise level application development.

As mentioned at the beginning of this chapter, this thesis work could be followed by developing one integrate application which combines the major capacities of all these SDKs. It would be very valuable product demonstration if the application could work not only against the Emulators, but also the real product nodes.

10 Abbreviations

API
Application Programming Interface

AVP
Attribute Value Pair

CORBA
Common Object Request Broker Architecture

EIS
Enterprise Information System

EJB
Enterprise JavaBeans

GMPC
Gateway Mobile Positioning Center

IDL
Interface Definition Language

J2SE
Java 2 Platform, Standard Edition

J2EE
Java 2 Platform, Enterprise Edition

JAXB
Java Architecture for XML Binding

JCA
Java Connector Architecture

JDK
Java Development Kit

JML
Java Mobile Location

JVM
Java Virtual Machine

LBS
Location Based Service

MMS
Multi Media Service

MO-LR
Mobile Originated Location Request

MPS
Mobile Positioning Service

MSISDN
Mobile Station Integrated Services Digital Network

MT-LR
Mobile Terminated Location Request

NRG
Network Resource Gateway

OMA
Open Mobile Alliance

OOP
Object Oriented Programming

OSA/HOSA
Open Service Access/ High level Open Service Access

PPS
Pre-Paid system

SDK
Software Development Kit

SMS
Short Message Service

SMPC
Serving Mobile Positioning Centre

SOAP
Simple Object Access Protocol

RMI
Remote Method Invacation

XML
Extensible Markup Language

11 References

[Ref 1]: SUN J2EE Tutorial, Chapter 1: Overview, Figure 1-1 Multi-tiered Applications

[Ref 2]: SUN J2EE Tutorial, Chapter 1: Overview, Figure 1-5 J2EE Server and Containers

[Ref 3]: SUN J2EE Tutorial, Appendix D, Figure D-1 Resource Adapter Contracts

[Ref 4]: SUN Java Web Services Tutorial, Chapter 1: Binding XML Schema to Java Classes with JAXB, Figure 1-2 Steps in the JAXB Binding Process

[Ref 5]: Xiaoying Wang, Evaluation Report of Charging SDK 1.0 for PPS 3.6, Chapter 2: Overview.

[Ref 6]: Ericsson, User's Guide - PPS Diameter API, Chapter 2.1: Description, Figure 1. Network principle

[Ref 7]: Diameter Base Protocol, draft-ietf-aaa-diameter-17.txt, Abstract Chapter.

[Ref 8]: Ericsson Parlay SDK R3A8, Programmer's Guide, Chapter 1.4

[Ref 9]: Ericsson Parlay SDK R3A8, Programmer's Guide, Chapter 1.4, Figure 4: NRG providing a set of APIs independent from underlying networks.

[Ref 10]: Ericsson Parlay SDK R3A8, Programmer's Guide, Chapter 1.5, Figure 3: SDK Overview.

[Ref 11]: Ericsson MPS API Programmer's Guide, Section "What is MPS SDK?".

[Ref 12]: *Design Patterns,* Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides (Addison-Wesley Publishing Co., 1994; ISBN: 0201633612)

Important websites for downloading Ericsson SDKs:

Diameter Charging SDK 1.0 D12E

Diameter Charging SDK (High level API)

MPS SDK

Ericsson NRG SDK

Important websites for related technologies:

XML technology

Parlay

Sun J2EE tutorial