MoBill: a framework for billing in next generation networks

Submitted in partial fulfilment
of the requirements of the degree of

Bachelor of Business Science

of Rhodes University

Moses Thizwilondi Nkhumeleni

Grahamstown, South Africa
November 1, 2010

Abstract

Convergence of voice, video, and data has led to the development of more complex services that combine different media. These services are commonly referred to as next generation services. Traditional billing frameworks are inadequate for next generation services. In part this is because an application may be composed of different services. Furthermore, the use of different media adds additional complexities to traditional time-based charging. Hence, next generation services require new innovative billing.

The project aims to investigate billing for next generation services. In this project we identified event, time, volume, and subscription as core charging models. Reward and flexible charging models were extended from the core charging models. Ericsson Diameter API implements the Diameter protocol which is used to perform accounting of resource usage. As proof of concept for these charging models (core and extended charging models), we developed a billing component called MoBill. MoBill was implemented using the Ericsson Diameter API. Mobicents was used as a service development platform. Mobicents examples were used to test MoBill. These examples essentially demonstrated how we can implement different charging models using MoBill.

ACM Computing Classification System Classification

Thesis classification under the ACM Computing Classification System (1998 version, valid through 2010):

C.2.1 [Network Architecture and Design]: Packet-switching networks

D.2.3 [Coding Tools and Techniques]: Object-oriented programming

D.4.7 [Organization and Design]: Real-time systems and embedded systems

Acknowledgements

This research project has been exhaustive and at times required a lot of support. It would be unfair of me not to mention some of the people and sponsors that contributed to the success of this project.

Firstly I would like to thank my supervisors, Professor Alfredo Terzoli and Mr Mosiuoa Tsietsi. Thanking Professor Alfredo for all the structural and high-level support to my research. In addition, I would also like to thank him for assisting me in my career aspirations and motivating me to do Computer Science honours. I have learnt a lot from just knowing Professor Alfredo. Thank you very much for being a supervisor and a mentor. I would also like to thank Mosiuoa for all the assistance he gave me throughout the year. I cannot recall how many times I went to his office saying, "but this thing just does not work". I would like to thank him for his valuable time that he sacrificed to come sit and help me with the technical concepts, proof reading my work and always being the first person I consult with difficulties. Thank you Mosiuoa, you just an inspiration.

I acknowledge the support of the convergence group. One name which sticks out is Mathe Maema for her tough love approach and at times for pushing us hard so that we can achieve our potential. Yes, super woman indeed! Thanking Ray for his system and support he gave me. Shange, Zelalem, Walter, Erasmus, Grace, and Gerhold have been fantastic research group members for brainstorming ideas.

This research would have not been possible without the financial support. The financial assistance from the Allan Gray Rhodes Prestigious Scholarship towards this research is hereby acknowledged. I would also like to acknowledge the financial and technical support of Telkom, Comverse, Tellabs, Stortech, Amatole Telecom Services, Bright Ideas 39 and THRIP through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University.

Last but certainly not least my family, friends and colleagues for all the support given to me throughout the year. Through the hardship, through successes; my family has always been there, at times believing in me more than I do myself. Above all, I thank God for being faithful and carrying me thus far.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Billing is a fundamental function of a telecommunication network provider. The billing operation must be performed accurately and reliably. IMS (IP multimedia subsystem) is an architecture that allows easy deployment of multimedia services in next generation networks [4]. Billing forms will be important with the introduction of IMS, since there will be a number of new types of services presented to the user.

The convergence of voice, video and data allows developers to deliver rich services to the user. Factors such as increased number of services, service composition, and the quality of service offered are driving the search for innovative billing approaches. The shift towards an all IP network allows developers to produce more sophisticated services that require more complex charging models. For instance, if during a voice session the user upgrades to video call with more network resources, charging should adjust accordingly. Another example would be flexible location based charging, where the user might be billed differently based on their location [26].

1.1 Problem Statement

Traditional charging models were primarily time-based. The time-based charging models provided a sufficient solution for voice calls. The complexity of new services means that more innovative billing strategies need to be investigated. More service development due to the convergence of voice, video, and data leads to variety of services available to the user. This variety translates to the requirements of more complex charging models. Moving towards IP networks introduces issues with QOS (Quality of services). Traditional

billing approaches were not concerned with QOS since they operated on circuit switched networks.

1.2 Research Goals and Scope

The Convergence Research Group at Rhodes University has been researching on a number of IMS related services, such as IPTV, video on demand, and location based services. The main objective of this project is to develop a billing system for next generation services.

Goals of the project are:

- Develop a billing system to provide billing for Mobicents services.
  - The Ericsson charging SDK will be used to develop the proposed MoBill component.
  - The Ericsson charging SDK has a number of tools that allow developers to incorporate charging into their systems.
- Test the billing system by using Mobicents applications.
  - In order to test the billing system, we identify a number of different examples to demonstrate some of the concepts.
  - These examples will be extended to incorporate the proposed MoBill Component.
  - The Ericsson Diameter Emulator is used to emulate a prepaid system consisting of user accounts. The emulator handles and responds to charging requests. The emulator extends our test bed for the billing operations.
- In pursuing the two goals above, we have loosely used an "IMS point of view". That is due to the fact that the environments for which we see the applicability of our work are at the moment by and large IMS compliant.

1.3 Methodology

The approach taken in developing the Mobicents based billing system was to firstly develop a billing component (MoBill) that interacts with the Ericsson server. The component contains required parameters to setup a TCP connection to the Ericsson server and hence be able to transmit Diameter messages. Once connections are established, the component contains billing logic that is required to implement event, time, volume, reward and flexible session charging models.

Following development, the second phase of the methodology was to test the charging models by modifying example applications to use MoBill. This involves adding the component to the Mobicents platform and modifying application in order to add billing code required by each application. This involves linking MoBill to the services building blocks (SBBs) used by the Mobicents platform. Due to the given time frame, we focused primarily on charging models instead of the applications used to test these models. This afforded us the opportunity to develop more sophisticated charging approaches without diverting the focus onto developing Mobicents applications.

1.4 Thesis Structure

The remaining chapters are structured as follows:

- Chapter 2 reviews literature around billing and billing implementation. Online and offline charging functions are discussed as part of the charging infrastructure. The Diameter protocol is discussed in detail as the base protocol that is used for implementation.
- Chapter 3 discusses billing within IMS. Factors affecting billing and charging principles are discussed as a guideline to developing billing systems. Also discussed are tools that may be used to develop services and charging systems.
- Chapter 4 discusses the general design of the proposed MoBill Component. System specification, UML diagrams are used to elaborate on the functionality of the MoBill Component.
- Chapter 5 focuses on the implementation and technical details of the system. Then it discusses the cases used to test the system.

- Chapter 6 concludes the research conducted. We revisit the projects objectives and goals. Possible future work and extension are discussed.

Chapter 2

Literature Survey

2.1 Introduction

Billing includes a number of concepts such as: online and offline, prepaid and post paid, and real time charging. Charging models allow operators to generate revenue for services offered. Subscription, time, event, volume, and reward based charging models will be discussed in detail.

Diameter protocol performs Authentication, Authorisation, and Accounting. The diameter protocol is important when transporting messages containing accounting information and authentication information. Diameter will be discussed in detail since the proposed billing system is based on it.

The charging architecture describes a logical layout of charging functions. The charging architecture is composed of online and offline charging functions. We will discuss the online and offline charging functions in detail as they are important when performing both offline and online billing.

2.2 Billing Concepts

This section defines and highlights major concepts in billing. The objective is to introduce terminology that will be used throughout the document.

2.2.1 Offline and Online Charging

Offline charging refers to charging that is conducted after resources are consumed. For example, if a user initiates a call; the user would be billed after the call. This means that the charging information gathered does not affect the actual service.

In offline charging, billing information is sent to an external billing system. The function of this external billing system is not part of the 3rd Generation Partnership Project (3GPP) standard nor is the transferring of data from the network to a billing system [2].

With online charging, the user account is checked to determine if there is enough credit for the user to consume the service. To clarify this point further, online can be thought of as an approach that has real time interaction with the service that is being offered. Online charging is done making use of an online charging system that conforms to the 3GPP standard [2].

Direct debiting and Unit Reservation are two scenarios for online charging. With direct debiting, the account is debited immediately after a charging event occurs. With unit reservation, units are reserved on the account of the user. When the user completes using the resources the remaining credits are returned to the users account.

In both online and offline charging there are two main approaches to charging:

1. Event-based charging
    - With event-based charging, an event is raised with a single transaction performed by a user, such as sending a multimedia message.
    - Event-based charging results in a creation of a single CDR (Charging Data Record) [2]. CDR's are collection of charging information about an event.
2. Session-based charging
    - A characteristic of session-based charging is the establishment of a user session, such as a phone call.
    - Session-based charging results in multiple charging events that could result in multiple CDRs [2].

2.2.2 Account Types

There are two primary types of accounts that are used by users, namely prepaid and post-paid accounts. With a post-paid account, the user receives a bill listing the charges that have been incurred after a certain period. This allows for a simple billing system to be implemented. A contractual agreement is required between the network operator and the customer for a post-paid account. Prepaid account requires no contract; however before using services offered by the network provider, the user must have credit in his/her account.

A trend has shown that there has been an increase in the number of prepaid users [14]. The major reason for this is that pre-paid account requires less administration compared to post-paid account, which usually requires a contract to be signed between the user and the service provider. Network operators offering prepaid to customers have experienced huge growth with the number of customers and the revenue [13]. Prepaid function is based on alarms, whereby the user is able to use services up until a certain threshold [14]. For instance, the number of credits the user has would translate to the number of seconds that can be used. If the numbers of credits run-out; an alarm is raised.

2.2.3 Business Models

A business model is defined as the relationship between a business and the service it provides [5]. There are three main business models in the telecommunications industry, namely content providers, service providers and network operators.

- Content providers focus mainly on delivering downloadable content such as music.
- Service providers focus on the delivery of services to the users.
- Network operators provide access to the network.

Businesses may combine two or more of these business models. Network operators often take the role of service providers or content providers [5]. With the convergence of telephony and IP networks, network operators will be required to move towards service delivery models in order to remain competitive[25].

2.2.4 Different Charges

There are number of different charges that telecommunication primarily focuses on, mainly service charges, tariff charges, and content charges.

- Service charges are for using particular services.
- Tariff charges are for accessing the network.
- Content charges are charges based on the content downloaded by the customers.

2.3 Charging Models

Charging models are used by service providers in order to recover costs and to generate revenue. It is imperative to have a suitable charging model for the service or content provided. In this section we will review some of the charging models we identified.

2.3.1 Subscription-based

With subscription-based charging model, the users pay a fixed amount over a period of time. Subscription-based model encourages more of resource usage since the customer pays a fixed amount which is independent of the amount of resources used. Subscription-based model is most popular in charging for internet access, the users pay a fixed amount based on the access channel they prefer [8]. The main advantage with subscription-based model is that users can predict their charges.

2.3.2 Event-based

Event-based charging is based on chargeable events that occur. A chargeable event is a single transaction, e.g. sending an MMS. Furthermore, event based charging uses a single CDR [2].

2.3.3 Volume-based

In volume-based charging model, the user is billed according to the amount of data transferred. The advantage of this model is that the users are only charged for the amount of content that they have received or sent.

2.3.4 Time-based

Time-base charging is based on the units of time the user consumes. This is a popular traditional billing approach used by telecommunication companies. With the introduction of IMS (IP Multimedia Subsystem) this charging model will be unable to cater for the variety of services that users will be able to access.

2.3.5 Reward-based

Reward-based charging model is based on the users selected criteria of use. This model is useful for creating brand loyalty. For example the more the user consumes a services, the price is reduced as they continue to consume that particular service.

2.4 Pricing Strategies

There are a number of pricing strategies available. Pricing strategies are used by service providers to put a price on their services. Ozianyi *et al.* [23] identifies a number of these pricing strategies namely: Flat-rate pricing, Paris-metro pricing, Responsive pricing, and Priority pricing.

2.4.1 Flat-rate Pricing

Flat-rate pricing strategy is based on a fixed subscription amount the customer pays for a service. Flat-rate allows the user to consume the services or resources over an agreed period of time. In telecommunication networks, flat-rate pricing would allow users to use resources in an unlimited manner; thus, it is the simplest pricing strategy to implement [23].

2.4.2 Responsive Pricing

Responsive pricing is primarily based on the network traffic. If there is high network traffic, the price is adjusted to a higher price and when there is low network traffic the price is reduced. This pricing strategy is effective at controlling the amount of traffic on the network, since the price sensitive users will delay their use to times when the price is low. Responsive pricing strategy requires more network resources due to signalling information [23].

2.4.3 Paris-metro Pricing

In Paris-Metro pricing, the network is partitioned into separate channels each with a different price [21]. Even though there is no bandwidth guarantee and the network still offers best effort delivery, it is expected that the channel with higher price will have less traffic and channels with lower prices will have more traffic. Similar to Responsive pricing there is a trade off between low quality of service and the price that the users pays for the service.

2.4.4 Expected Capacity Pricing

With expected capacity pricing, the prices vary with the amount of capacity that is expected on the network. The user is not limited to consume more resources when the network is congested but does so at higher price [8]. This pricing creates incentives for users to consume resources when the network is not congested by reducing the price. Additionally this pricing scheme directly translates to the cost that the provider incurs.

2.4.5 Priority Pricing

This pricing strategy focuses on the priority that the user receives on the network structure. A higher priority would translate to higher price that the user will be charged. Priority pricing can be used to create QoS profiles since it relates directly to traffic control [23].

Table 2.1: Service Charging Information

Application	QoS	Weight	Price
app1	Q1	W1	P1
app2	Q2	W2	P2
...	...	...	...
appN	Pn	Wn	Pn

2.5 Examples of Implemented Systems

In this section, we discuss two proposed solutions to billing for services in third generation's networks.

2.5.1 Service Definition Approach

In order to overcome the complications associated with service composition as described above, Oumina *et al.* [22] proposes a service definition approach. A table containing QoS, weight, prices of each application is stored in a charging module as shown in Table 2.1. The table is updated in real time when there are changes in the network. When charging for an application, the table is queried in order to give a charging amount.

2.5.2 Call Differentiation Approach

In order to address issues with billing for services with QoS, Barachi *et al.* [4] proposed a system which charges users differently based on the QoS of the call. Furthermore, the user could subscribe to one of the following classes: silver, gold, or platinum. These classes represent the Quality of service that the user has opted, silver represent low while platinum represents high QoS. Higher QoS translates to a higher cost. Furthermore, the system caters for emergency calls. Emergency calls do not have peremption because of the critical nature of the calls [4].

2.6 Diameter Protocol

The Diameter protocol has been created to perform AAA (Authentication, Authorisation and Accounting). Authentication is verifying the claimed identity of a subject, whilst

authorization is focused primarily on access that the subject has. Authentication, Authorisation and Accounting are linked and hence in order to account for the usage of resources all tasks must be performed [7]. Accounting is focused on collecting information with regards to the amount of resources being used. We are particularly interested in accounting since it allows for billing to be conducted on resource usage information.

IETF (Internet Engineering Task Force) proposed Diameter as a replacement of RADIUS protocol. RADIUS runs on unreliable UDP protocol. Diameter runs on TCP which offers reliability and congestion control. There are a number of entities defined by diameter such as Diameter client, Diameter server, realm, etc [7]. Diameter client is a located at the edge of the network and mainly performs access control and Diameter server is primarily used for authentication, authorisation, and accounting [7].

Diameter is a peer to peer protocol; therefore peers can send requests to other peers [7]. Diameter messages contain header information and a number of AVPs (Attribute value pairs). AVPs can be considered as containers of data. AVPs can be used to transport authentication information, in order for the user to be authenticated. AVPs can also be used to transport resource usage information which is required for accounting. The Diameter protocol can also be extended by defining new AVPs values, or by creating new AVPs [6]. Diameter makes use of either request or answer commands. There are a number of request and answer commands used by diameter, e.g. ACR (Accounting Request) and ACA (Accounting answer) [6].

2.7 Charging Infrastructure

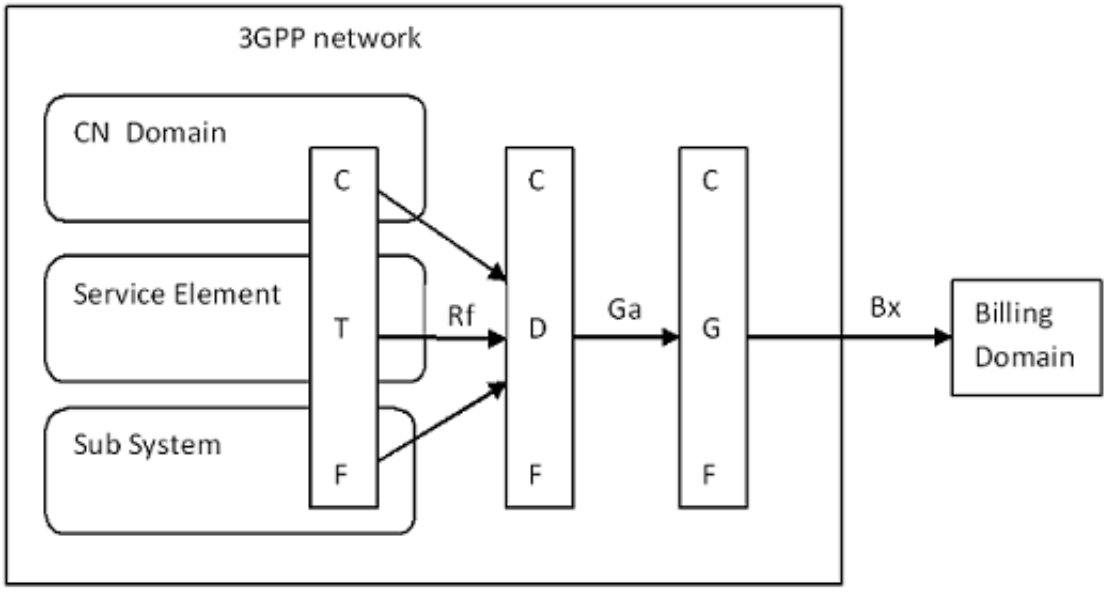
Depending on whether we are implementing an online or offline billing system, the charging architecture will change accordingly. This section describes offline and online charging architectures.

2.7.1 Offline Charging Function

2.7.1.1 Charging Trigger Function

The charging trigger function (CTF) creates charging events based on the network resource usage [2] . The CTF gathers information about charging and packages them into

Figure 2.1: Offline Charging Function



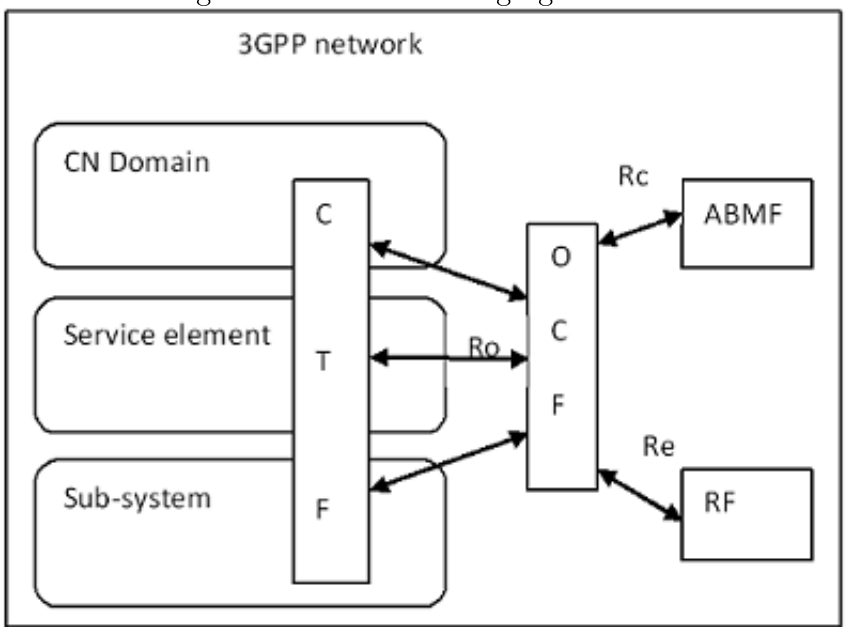
charging events, which are sent to the next function (Charging Data Function). The CTF consists of the following functions:

- Accounting Metrics Collection
  - This function simply monitors call signals, events, or sessions that users establish.
  - This function would produce information that identifies the user and the amount of resources consumed.
- Accounting Data Forwarding
  - This function receives the information and attempts to identify chargeable events from the received information.
  - The events are assembled and forwarded to the Charging Data Function (CDF).

2.7.1.2 Charging Data Function

This function receives charging events from CTF via Rf reference point as shown in the diagram. These events are used to create CDRs (Charging Data Record). CDR contains information such as time of call, duration of call, amount of data transferred, etc.

Figure 2.2: Online Charging Function



2.7.1.3 Charging Gateway Function

The CDRs produced in the Charging Data Function (CDF) are transferred to the Charging Gateway function (CGF). The CGF therefore links the 3GPP network to the Billing Domain; remembering that the Billing Domain is not part of the internal 3GPP network structure. The CGF will receive CDR, validate, format and transfer CDR to the billing domain.

2.7.2 Online Charging Function

2.7.2.1 Charging Trigger Function

This process is similar to CTF defined for offline charging, however information gathered is different. Additionally, since online charging requires permission to be granted to resources before use, the CTF must be able to delay the use of resources until permissions are granted [2]. Another difference with offline charging is that there is bidirectional communication between CTF and OCF. The number of messages sent between the CTF and OCF means there is more overhead as a result of charging [24].

2.7.2.2 Online Charging Function

OCF contains two functions, namely: Session Based Charging Function(SBCF) and Event Based Charging Function (EBCF). SBCF conducts online charging for user sessions such as voice calls and IMS sessions [2]. EBCF is responsible for event-based online charging.

2.7.2.3 Rating Function

Rating Function (RF) is used to determine the value of the resource using the information in the charging event. The RF output is monetary or non monetary units which are sent back to the OCF.

2.7.2.4 Account Balance Management Function

This is a storage area containing the users account balance.

2.8 Summary

In this chapter, we have reviewed literature around billing. Different billing concepts were highlighted. The five identified charging models were: event, time, volume, reward and subscription based charging. Pricing strategies are closely related to charging models; whereas pricing strategies focus primarily on how to price for services, charging models are focused on how to implement charging. The Diameter protocol has been introduced as a replacement for RADIUS protocol. Diameter is responsible for performing AAA (Authentication, Authorization, and Accounting). Charging and billing is primarily focused on the accounting functions of Diameter. The choice between online and offline billing will affect the architecture that is used when considering charging into systems. We have discussed online and offline charging infrastructure in detail and highlighted different functions carried out within the infrastructure.

Chapter 3

IMS Billing and Related Technologies

3.1 Introduction

The concerns of network operators charging users by the amount of traffic on the network impose a challenge when charging various services. IMS (IP Multimedia Subsystem) billing can be used to allow network operators to use more complex billing approaches instead of just billing by the amount of traffic on the network only. Billing for next generation services will be the driving force behind releasing services into the market due to high competition and the variety of services that will be available to the user. Being able to deliver services to the user rapidly will allow service providers to have a competitive edge over others. In order to achieve these requirements, there are a number of tools that aid the development of IMS-compatible services such as the Mobicents platform.

The Diameter protocol discussed in Chapter 2 is used for accounting purposes. There are a number of APIs that implement the Diameter protocol that can be used to develop billing systems. In order to test our billing system, we need a server that responds to the billing requests and performs debits and credits on user accounts. In this case, we use the Ericsson Emulator Server which hosts user accounts and responds to Diameter messages sent from the application.

In this chapter we introduce factors affecting billing for next generation services. Furthermore, we highlight why the current time-based charging model is insufficient to cater for the variety of services available. We continue by discussing the Mobicents platform and the different Diameter APIs.

3.2 Factors Affecting IMS Billing

With the introduction of third generation (3G) networks, the main idea has been to merge cellular networks with IP networks. 3G networks operate using packet switching and therefore allow the users to access the Internet via the IP protocol. However, the IP protocol offers only best-effort delivery. There is no bandwidth guarantee that is offered with data transfer. This means that services are available without guaranteed QoS (Quality of Service). This presents additional concerns when offering real-time multimedia services. IMS (IP Multimedia Subsystem) allows for synchronisation of session management with QoS and hence this allows users to specify desired QoS [7]. In this section, we summarise factors affecting billing and charging in 3G networks.

3.2.1 Variety of Services

IMS promises to deliver a number of different services to its users such as voice, data, video, conferencing and e-commerce services. These newly developed services impose a challenge to the traditional time-based charging model. Variety of services is one of the factors motivating the investigation into new innovative billing strategies.

3.2.2 Quality of Service

Quality of service (QoS) can be defined as the quality of transmission in a network that the user receives based on various factors such as transmission rates and packet loss. When providing services across IMS, the issue of QoS becomes important. The quality of service should therefore be considered in the charging model. Service differentiation is the ability to treat different classes of traffic differently based on their QoS [10].

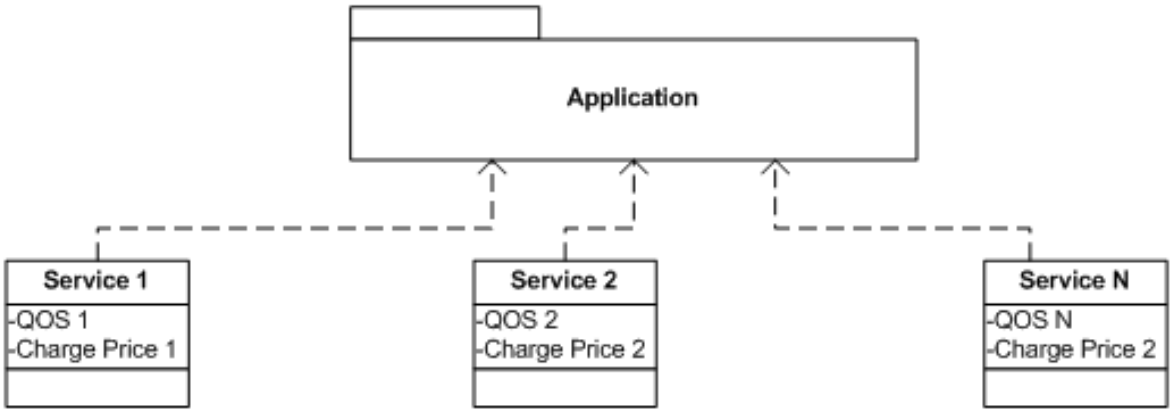
The users should be able to select the quality of service that they prefer. Hence different QoS tiers should apply to different rates to bring about service differentiation.

3.2.3 Service Composition

With the introduction of IMS, service composition has become attractive due to the ability to deliver rich applications that combine different media forms such as voice, data and

video. There are a number of additional complications that are associated with service composition. Namely, an application could make use of different application servers and third parties maybe involved in delivering services to the user [22].

Figure 3.1: Application Composed of Different Services, adapted from [22]



These additional complexities give rise to a more complex charging system. For example, if we have an application that consumes n different services from different service providers as shown in Figure 3.1, each service is associated with a certain quality of service and charge price. The overall quality of service will be dependent on the QoS of different services composed in the bundle. Additional complications can be added to the charging model since all services may not necessarily be used at the same time. During the execution of the application, the service bundles will change and the QoS will vary. Services in an application will have different level of importance and therefore, an application can also execute without some services [22]. If we have a charging approach that depends on QoS, charging will vary with the change in time due to the factors mentioned above.

3.2.4 Different Service Providers

IMS allows developers to combine a number of different services to create a single application. These services may be available from different service providers. For example, an application can use weather services from one service provider and news services from another provider. This poses challenges to the charging model, since the service providers will require their share of the amount that the user is billed for.

3.2.5 Flexible Charging

In order for network providers to remain competitive, their pricing strategies will need to be revised. The charging scheme must be flexible and cater for different QoS tiers. Flexibility means that pricing strategies should be subject to change based on the context of the user. For instance, different locations would result in users being charged with a different price. Quality of service and having flexible charging of services are fundamental with the introduction of IMS [4].

3.2.6 User Preferences

Traditionally, network operators have proposed a billing model whereby a tariff is set and rolled out to all customers. This means that all customers were charged in the same manner. There was no personalised pricing strategy which considered the customers preferences. Future billing models will need to consider users preferences when it comes to billing for different services [3]. So each user should be billed differently based on their individual preferences. This means that network operators will need to reverse the traditional billing approach and therefore, consider user's preferences or different parameters associated with the user. For example a user can have a tailored billing scheme based on their preferences.

3.3 IMS Charging Principles

The complexity involved in IMS charging as discussed above requires following a few guidelines and principles when developing billing services. 3GPP [1] suggests a number of principles to follow when performing billing for next generation services. In this section we highlight some of these suggested charging principles. Additionally, we use the examples reviewed in Chapter 2 to help with the explanation of these principles.

3.3.1 Quality of Service Requirements

Different levels of QoS should be charged differently. Depending on the quality of service that the user receives, they should pay accordingly. For example, if the user is streaming video at high quality, s/he should therefore pay more than another individual who is

streaming at lower quality. This approach becomes important when managing network traffic. The call differentiation example reviewed in Chapter 2 demonstrates how billing a service with QoS can be done [4]. The user is billed differently based on the QoS they receive on the call.

3.3.2 Location Considerations

There should be consideration for flexible location based charging, where the user might be billed differently based on their location [26]. The user can be billed higher when outside their local area. The location requirements mean that there should be different tariffs for use in different locations.

3.3.3 Charging Media and Services Separately

The convergence of voice, video, and data media allows the developers to have services using a combination of different media. As a principle to follow, different media should be charged separately from others. Additionally, different services should be charged separately to cater for composed services. Multi-Definition of Services in IMS example we reviewed in Chapter 2 demonstrates charging an application with composition of services [22].

3.3.4 Upgrading a Session

If the user upgrades a session, charging should change accordingly during the initial session. For example, if the user was using a voice call and decides to upgrade to video call, charging should change accordingly. With the call differentiation example the user can upgrade a call to higher QoS and hence adjust billing accordingly.

3.4 Suggested Charging Approach for Services

In this section, we list different IMS services with a suggested charging approach [1]. These listing are captured in Table 3.1. As it can be seen, for each service, there is a charging approach that is associated with a charging model. These services can be combined to

Table 3.1: Different Services with Respective Charging Models

Service	Charging Approach	Charging Model
Voice	Charge the duration of the session	Time-based charging
Video and Audio Streaming	Charge by the amount of content downloaded. Irrespective of the users usage criteria, charge the user a fixed monthly rate.	Volume-based charging and Subscription-based charging
Multimedia Messaging	Charge per message. Charge by the amount of content sent.	Event-based charging and Volume based charging
Location based Services	Charge user for location information received.	Event-based charging
Downloadable Content	Charge by the amount of data retrieved.	Volume-based charging

produce a more sophisticated application that would result in a more complex charging model.

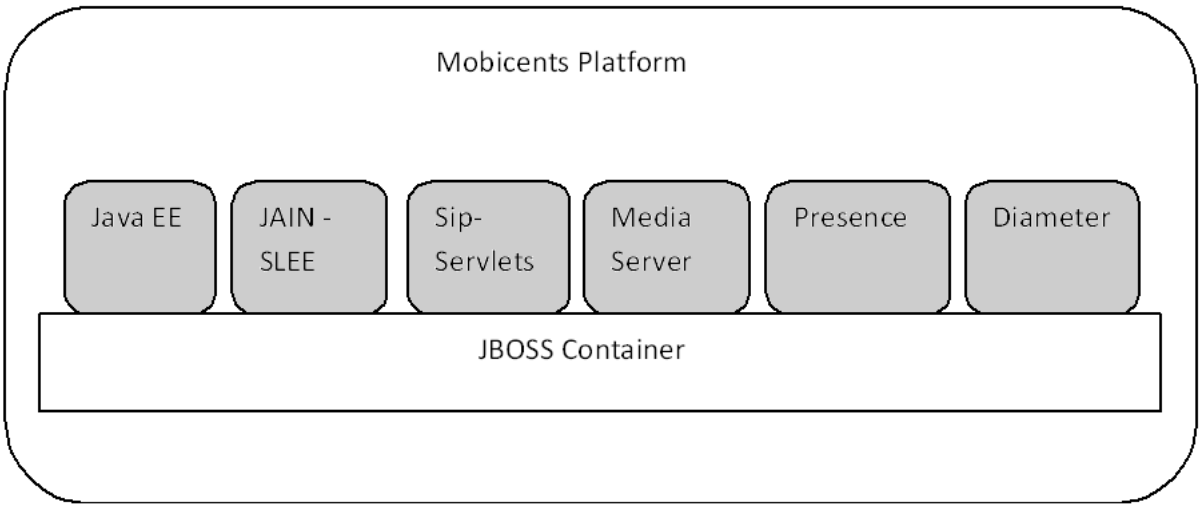
3.5 Related Technologies

In this section, we discuss technologies for developing next generation services. Additionally, we discuss the different implementation APIs of the Diameter Base Protocol. The two APIs we discuss in this section are Open Diameter and Ericsson Diameter Charging API. We also motivate our selection of different technologies that we used in developing the billing system.

3.5.1 Mobicents

Mobicents is an open source service development platform for next generation applications. Mobicents allows developers to create applications that combine voice, video and data. The Mobicents platform is certified as JAIN SLEE compliant. SLEE (Service Logic Execution Environment) is a popular standard in the telecommunications industry. JAIN SLEE is Java standard for SLEE. SLEE is similar to EJB (Enterprise Java Beans) but has different components [9]. Mobicents JAIN SLEE is built on top of the JBOSS application server as seen from Figure 3.2.

Figure 3.2: Mobicents Plaform, adapted from [9]



Mobicents JAIN SLEE compliments J2EE (Java enterprise) container features [9]. Mobicents also facilitates the composition of SBBs (Services Building Blocks). The different SBBs available are: billing, call control, user-provisioning, administration and presence. The use of resource adapters allows developers not to be concerned with the underlying network technologies. Resource adapters act as wrappers that allow SLEE applications to communicate with external resources using standard internet protocols [18]. Of particular interest to billing is the Mobicents Diameter Resource Adapter. The Diameter Resource Adapter provides a development API for applications that implement billing. The Diameter Resource Adapter is based on the Diameter protocol, which is described in the next section.

3.5.2 Diameter APIs

In this section we discuss some of the implementations of the Diameter protocol. This section discusses the different APIs that are used to implement the Diameter protocol. We motivate our preferred API used to develop MoBill by highlighting the advantages and disadvantages of each API.

3.5.2.1 Open Diameter

Open Diameter is an open source implementation of the Diameter protocol. The Open Diameter is implemented using C++. Open Diameter provides classes to perform AAA

(Authentication, Authorisation, Accounting). The classes are grouped into two categories, client and server classes [12]. Client and server classes are further subdivided to authentication, authorization, and accounting classes.

Open diameter has little documentation. However, the advantage with Open Diameter is that source code is available. Open Diameter is not used for implementation, since preference was given to a Java implementation of Diameter.

3.5.2.2 Ericsson Diameter API

The Ericsson SDK provides a high level implementation of the Diameter API with a few layers of abstraction from the base API [20, 11]. The Ericsson Diameter API is implemented in Java which complements the Mobicents platform that we are currently working on. The Ericsson SDK provides a number of examples that demonstrate how to develop billing applications. The Ericsson Charging API also provides an abstracted view of base API with methods that allow developers to send and receive diameter messages. The developer is not concerned with the underlying transport protocols when working with the API. The disadvantage is that the entire source code for the Charging API is not available to the user, since the API works with pre-compiled binary jar files.

3.5.3 Testing Emulator

The Ericsson Diameter Emulator is a server application which acts as a potential prepaid system. It responds to requests from the client. The developers can use the emulator to test their systems. The emulator contains a database with all the accounts, currency, and tariff records [**?**]. The option menu allows users to set log files and additional connections options. The users can create and edit account information such as balance, currency and the account holder's name. Tariffs can be defined that affect how transactions will be charged. Counters such as received and sent Diameter requests are shown in the emulator. The emulator shows denied and faulty Diameter requests. Furthermore, the emulator also allows us to view Diameter messages and help in debugging applications. Since we are using the Ericsson Diameter API to develop MoBill, we have chosen to use the Ericsson Diameter Emulator to avoid potential interoperability issues.

3.6 Summary

In this chapter, we have discussed the factors that affect billing. We have explained the complexities associated with billing for next generation services. We discussed some of the principles to apply when implementing a billing system. Furthermore, we gave practical examples to demonstrate core principles. Examples of services were given with related charging models and description of how they can be charged. We indicated that an application composed of different services will lead to a more complex charging model.

Mobicents was discussed as the platform that can be used to develop next generation services. The Mobicents platform is certified as JAIN SLEE compliant. We discussed different APIs that implement the Diameter protocol. We motivated our choice of the Ericsson Diameter API by highlighting the advantages and disadvantages. The Ericsson Diameter Emulator was discussed as an application that can be used to test MoBill, since it receives and replies to Diameter requests.

Chapter 4

Designing MoBill

4.1 Introduction

This chapter discusses system design of the MoBill and examples that we used to test it. Mobicents examples were used; since they provide an opportunity to test various charging models.

Firstly we will discuss the considered approach for developing the MoBill. We will continue the discussion by stating the system specifications and requirements. Secondly we discuss the general system architecture and describe how the proposed component (MoBill) will be integrated into Mobicents. In the description of the system architecture we include the class diagram describing the associated between classes.

4.2 Design Approach

The three approaches considered for developing MoBill are monolithic design, using service SBBs (Service Building Blocks), and component design. The three approaches offer both advantages and disadvantages. We highlight the three approaches in this section and give reasons for our preferred approach.

4.2.1 Monolithic Design

Next generation services will be billed differently based on their requirements or the billing strategies. One approach is to incorporate billing logic into a service. With reference to mobicents, billing logic is added to the SBB that contains the code for the service. Hence, we would have a monolithic application where the service and billing logic are not separated. This gives programmers the ability to tailor a billing approach to a specific application. The disadvantage with this approach is that it is not modular and billing logic or processes are not separated from service logic. This adds additional difficulties such as code maintenance and separation of code. Furthermore, if billing strategies change the service needs to be edited. This is a problem since many services will use a combination of time, event, volume, and subscription based billing. This means that using this approach will replicate code with the obvious complications for each update. Finally, this approach contradicts the component based approach of the Mobicents platform, where the design approach is that services are developed by a composition of different service building blocks.

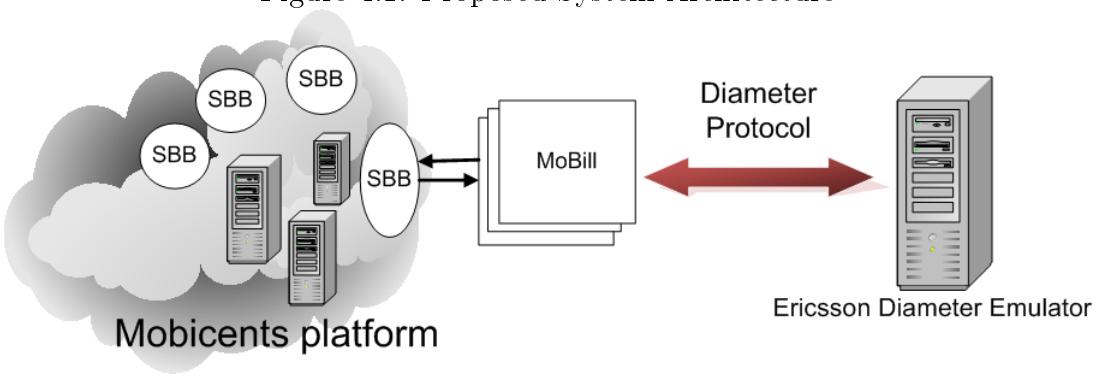
4.2.2 Service Building Block Approach

Another approach is to incorporate billing logic into specialised SBBs. The Mobicents platform provides a "plug-and-play" architecture through the use of SBBs. SBBs are service objects which can send and receive SLEE events; SBBs can therefore perform logic based on the events they receive. SBBs cater for reuse of code and additionally, SBBs can be organised into a graph to generate more complex services. The approach of creating SBBs allows other developers who require billing in their system to use these specialised SBB in order to perform billing. The use of SBBs allows for ease of extension of the billing system since the developers can incorporate additional billing strategies and make changes to configurations without modifying the services.

4.2.3 Component Design

The proposed approach is to develop an object in Java called MoBill that contains all the billing logic required to implement different charging models. The developer can simply incorporate MoBill into their Mobicents application. The developer is provided with a simple interface to MoBill and makes use of methods to add billing into their services.

Figure 4.1: Proposed System Architecture



The component design offers similar advantages as the SBB approach with an additional benefit of simplicity. Implementing billing engine via Mobicents SBBs means that the developed billing approaches are limited to Mobicents applications, whereas building a Java object means that we can plug in the billing logic to different platforms other than the Mobicents platform.

4.3 System Specification

MoBill has been proposed to demonstrate different billing strategies that can be used by developers in their services. The system should be able to cater for a number of different charging models, namely event, volume, time, and reward based billing. The system, through the use of modular programming, should be easily extensible. The developers should be able to plug in the component into their application and be able to access methods to commence on billing for their services. The system should provide a consistent interface between the service and MoBill such that changes to the underlying charging implementation should not require changes to the service. The system should allow developers with limited understanding of IMS billing and Diameter protocol to use intuitive methods to perform billing.

4.4 System Architecture

In this section we describe the general system layout including the Mobicents platform, MoBill and the Ericsson Diameter Emulator.

4.4.1 Mobicents

As it can be seen in Figure 4.1, from the left we have Mobicents platform. Mobicents provides a platform where developers can create services that allow for the convergence of voice, video, and data. Furthermore, the Mobicents platform supports IMS. The Mobicents server will contain a number of services that are currently deployed. These services are developed using a number of different SBBs (service building blocks). The SBBs are Java files containing code that is used to create a service. This allows us to add additional code to perform billing or to import objects such as MoBill that can be used to perform billing.

As we move to the right we have MoBill and Ericsson Diameter Emulator. These will be discussed briefly in the next subsections.

4.4.2 MoBill

MoBill contains methods to implement different charging models. MoBill has public methods that allow communication between the SBBs (service building blocks) in Mobicents application. MoBill will be instantiated in a SBB that requires billing. Based on the required charging model, different methods will be called to perform billing functions. For example, if we are conducting time-base charging model, the two methods that would be used are charge_startSessionBilling(), and charge_stopSessionBilling().

4.4.3 Ericsson Diameter Emulator

As shown in Figure 4.1 we see that MoBill is connected to the Ericsson Diameter Emulator. The communication between MoBill and the Ericsson Diameter Emulator is based on the Diameter protocol. MoBill is developed in Java and uses the Ericsson Charging API in order to send and receive Diameter messages. MoBill creates CCR (credit control request) messages containing a number of AVPs (attribute value pairs). Furthermore, MoBill also listens to the response messages from the Ericsson Diameter Emulator. The response messages state the success of the requests that have been received.

4.5 Selection of Charging Models

Charging models as described in the literature survey provide mechanisms for accounting resource usage to enable providers to recover costs. Here we discuss the implemented charging models and the reasons behind their selection.

4.5.1 Core Charging Models

The core charging models are event, time, subscription and volume based charging. Sophisticated charging models such as reward, and flexible session based charging are extended from core charging models. We chose to implement core models since they can also be combined to produce more sophisticated charging models. For instance, we were able to combine event-based charging and time-based charging to come up with flexible session based charging model. We have not bothered to implement subscription-based charging since the user pays a fixed amount irrespective of the usage. This is a trivial charging model to implement, hence it is omitted here.

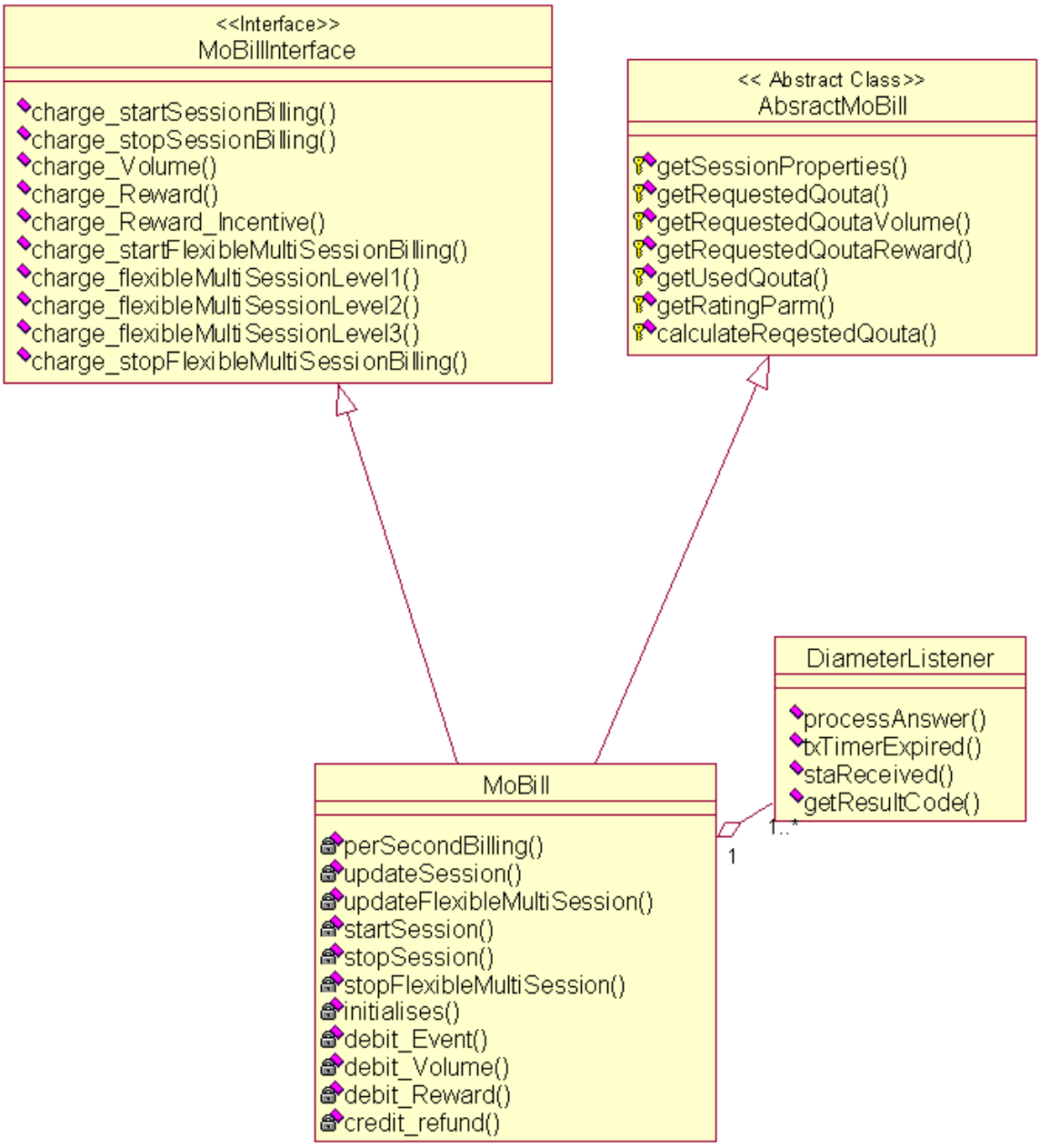
4.5.2 Extended Charging Models

The extended charging models constitute of reward-based charging and flexible session based charging model. As already suggested, extended charging models make use of the core charging models to produce more sophisticated charging models. Furthermore, they focus on providing billing where there is more interaction with the user. For instance, with reward based charging the user can be billed differently based on their selected criteria of use. Similarly if the user upgrades a session, e.g. requests higher QoS during a session, we can adjust the charge price accordingly to accommodate the user's changes.

4.6 MoBill Class Diagram

Figure 4.2 shows the design class diagram, we have omitted parameters from the class diagram. Parameters and additional method descriptions can be obtained in appendix A.

Figure 4.2: MoBill Class Diagram



4.6.1 AbstractMoBill Class

The AbstractMoBill class contains a number of protected methods that are mainly for returning various Objects such as UsedQuota, RequestedQuota, SessionProperties object. Below is a brief explanation of these methods.

1. getSessionProperties – returns a SCAPSessionProperties object with connection information (realm and user account ID).
2. getRequestedQuota – returns a RequestedQuota object with money units used, and currency.
3. getRequestedQuotaVolume – similar to 2 but used for volume based charging.
4. getRequestedQuotaReward – similar to 2 but used for reward based charging.
5. getUsedQuota - returns a UsedQuota object with money units and currency used.
6. getRatingParm – returns a RatingParameters object with information specific to service such as time zone.

4.6.2 MoBill Interface

The MoBill interface contains all the public methods required to implement different charging models. These methods will be discussed in more detail in section 4.6.5 . The interface allows us to define public methods which the MoBill classt must implement.

4.6.3 MoBill Class

MoBill class implements the MoBillInterface and extends the AbstractMoBill class. MoBill is the main class that contains most of the logic to implement different charging models. MoBill contains a number of private methods to support the methods described in the MoBillInterface. We will highlight some of the important methods here but for full description of other methods please refer to the appendix A.2.3.

1. perSecondBilling method– contains timers to perform per second billing on a session.

2. updateSession method – depending on the criteria specified for billing, a session needs to be updated when certain time lapses. For example, with per second billing; every second a session is updated to debit the account.

3. initialises method – Sets up connection information to the Ericsson server.

4. debit_Event, debit_Volume, debit_Reward methods performs debits on respective charging models.

4.6.4 DiameterListener Class

This is a listener class that is used to listen to responses from the Ericsson Diameter Emulator. The class contains the following methods.

1. processAnswer – executes when the Credit Control Answer (CCA) message is received. We simply display the answer.

2. txTimerExpired- used to perform tasks in case the specified waiting time for the response from Ericsson Diameter Emulator expires. An appropriate message is displayed.

3. staReceived – used to perform a task when the STA (session Termination Answer) message has been received. This occurs when a session has been terminated. We display an appropriate message.

4.6.5 MoBill's Public Methods

The MoBill Interface contains a number of public methods. These public methods implement various charging models. The methods are described in Table 4.1, which shows the methods and how they are grouped according to the charging model.

4.7 Mobicents Examples

To test the MoBill, we need a number of services to show different charging models. The emphasis is not on the complexity of the services but rather on demonstrating various charging models as proof of concept.

Table 4.1: MoBill's Public Methods

Method	Description	Charging Model
charge_Event()	When there is a single action performed, charge_Event() method is called to bill that event.	Event-based charging
charge_startSessionBilling()	Used to start session billing and to start the timer.	Time-based charging
charge_stopSessionBilling()	Used to stop session billing and to stop the timer.	Time-based charging
charge_Volume()	Used to initiate volume based charging. The method calculates the charge amount based on the number of kilobytes specified.	Volume-based charging
charge_Reward()	Initiates reward based charging and debits the account.	Reward-based charging
charge_Reward_Incentive()	This is used to indicate the usage criteria. Every time when this method is called, the number of incentives performed is incremented. Depending on the threshold, the user can receive a discount; hence we perform a refund on the users account.	Reward-based charging
charge_startFlexibleMultiSession Billing()	Starts session billing and the required timer.	Flexible session Billing
charge_stopFlexibleMultiSession Billing()	Stops session billing and the timer.	Flexible session Billing
charge_flexibleMultiSessionLevel1()	Sets the session level to one, this is done in case of downgrading from other level. Each level has a different charge amount.	Flexible session Billing
charge_flexibleMultiSessionLevel2()	Similar to the point above upgrades or downgrades session to level two.	Flexible session Billing
charge_flexibleMultiSessionLevel3()	Upgrades session to level three.	Flexible session Billing

4.7.1 Google Talk Bot Example

The Google Talk Bot example is a chat example that connects to the Google Talk service; it is available as one of the Mobicents examples [16]. The Google Talk Bot example provides a service that can be used to demonstrate event-based charging model. Originally, the example appeared as regular Google talk user that interacts with Mobicents service and responds with a message length of the characters the service received. However, we have slightly modified the example to respond with a text message, such that it acts like a Chat Bot. The user is charged per transaction performed. This provides a suitable example to demonstrate event-based charging.

4.7.2 SIPB2B UA Example

The SIP back-to-back user agent (SIPB2BUA) example is a call controlling service[15]. This example is also available with the Mobicents package. The SIPB2BUA service handles SIP [27] messages between the sip user agents. The example separates the SIP communication into two legs, allowing developers to add additional code to perform accounting of resources and prepaid billing.

4.7.3 Video on Demand Example

To illustrate reward based-billing, we made use of Video on Demand Service developed by Ray Musvibe, Rhodes University [19]. The System shows how we can use On-Demand advertising. On-Demand advertising allows the user to chose when they want to watch adverts. In order to encourage users to watch advertisements, an incentive is given to the user by giving the user a discount if they watch x amount of adverts.

4.8 Summary

In this chapter we discussed the different design approach for our system; highlighting the advantages and disadvantages of each method. Furthermore, we motivated the reasons why we chose to use the component design. Specification and requirements of the system were discussed. We emphasised the need to have a billing component (MoBill) that contains different charging models to cater for various billing requirements. The system

architecture was discussed and we showed how MoBill interacts with both the Ericsson Emulator and the Mobicents server.

We motivated the reasons why we chose various charging models to implement and showed how we can derive extended models from the core charging models. We discussed the class diagram and the interaction between different classes. The chapter was concluded by discussing the testing examples used for testing MoBill.

Chapter 5

Implementation, Dynamic Analysis and Testing of MoBill

5.1 Introduction

MoBill is the proposed solution for combining different charging models into one system. MoBill is developed in Java and focuses on demonstrating event, time, reward, volume, and flexible-session charging models.

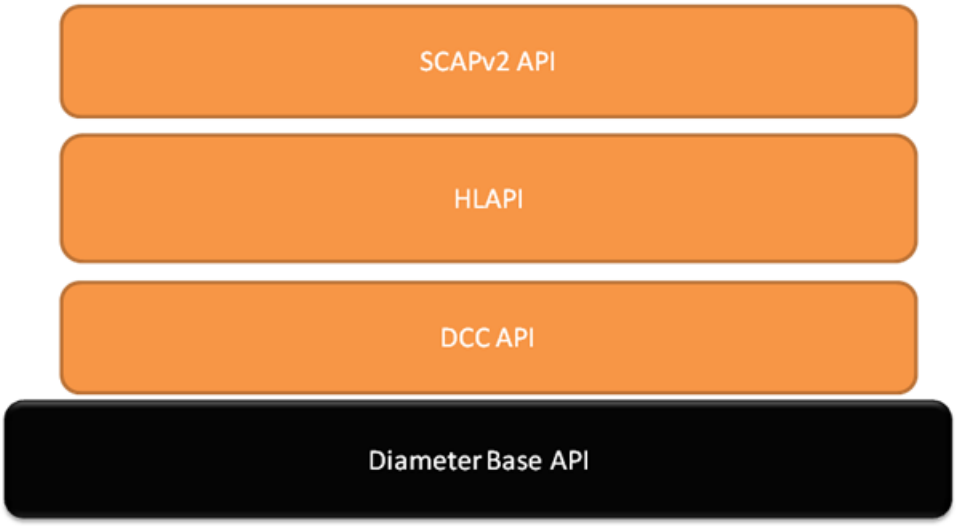
In this chapter we discuss how MoBill is constructed. We show more details to how each charging model is implemented. After the implementation details are discussed, we explore the different examples used for testing different charging models.

5.2 Using Ericsson Diameter API to Implement MoBill

MoBill was developed in Java and made use of the Ericsson Diameter API [11]. The Ericsson Diameter API provides a high level abstraction from the Diameter Base protocol. The Ericsson Charging SDK contains a number of jar files (based on Diameter) that can be used to implement charging applications. These files include:

- DiameterBaseApi.jar – This implements the Diameter Base protocol [6].

Figure 5.1: Ericsson Diameter API



- DccApi.jar – It is based on Diameter Credit Control Application protocol [13] that provides guideline to real time and credit control. DCC Api is placed on top of Diameter Base Api.
- HighLevelApi.jar – It is a additional layer of abstraction that provides easier interface to the Diameter Base Api. The High Level Api is placed on top of the DccApi.
- Scapv2Api.jar – This limits High Level Api to SCAP (service context for the application) and adds additional methods and parameters relating to SCAP.

Figure 5.1 shows the dependencies and the additional layers placed on top of the Diameter Base API.

The jar files contain a number of classes that can be used to develop a charging application. Please refer to Appendix F.3 for a list of classes and description of classes available. In Table 5.1 we give a brief description of the classes we have used to implement MoBill.

The first step (before performing charging) is to initialise the SCAPStack. We define a new SCAPStack with the originating URI, the original realm and a product ID. Once these parameters are defined, we add a static route to the Ericsson Diameter Emulator. For this we need the destination realm and the destination URI. Once attained, we start the SCAPStack.

The next step is to create and define a listener class that receives an answer once a credit control request (CCR) message is sent. This class will capture the different types of results

Table 5.1: Classes Used for MoBill Implementation

Class Name	Description
MoneyUnit	Used to describe unit type Money. Money is described in terms of value, exponent and currency code.
RequestedQuota	We use RequestedQuota to represent the requested quota that is sent with a CCR (credit control request) message.
UsedQuota	Represents the used quota that is placed in CCR message.
SCAPEvent	Contains doDirectDebit and doRefund which allows us to perform a transaction based on a single event.
SCAPSession	Used for session based charging. Contains start(), stop() , terminate(), update() to perform related functions.
SCAPChargingFactory	Allows the creation of SCAPEvent and SCAPSession.
SCAPSessionProperties	Represents services specific parameters as AVP(attribute value pairs)
SCAPRatingParameters	Used to provide SCAP settings such as the destination realm.
SCAPStack	A wrapper for the Diameter stack which provides SCAP settings.
TimeZoneAvp	AVP that holds time zone. We use this to define rating parameters.

we receive. Hence, we need implement the processAnswer() method of the listener class that captures response messages. The result code is stored.

Once we have initialised and started the stack, we can perform a debit or credit on user accounts defined on the Ericsson Emulator. The Ericsson Charging API can provide charging based on either a session or a single event. Below we describe the process followed for each approach used.

5.2.1 Charging Based on a Single Event

For single event charging we need to define SCAPevent class. The two main methods we are interested in are doDirectDebit() and doDirectRefund(), which allow us to perform a debit and credit respectively. When any one of these methods is used, a CCR message is created and sent. The CCR message (Diameter message) will contain a number of AVPs (attribute value pairs). AVPs are tuples containing an attribute name and associated value. Below we have identified two AVPs:

- CC-Request-Type AVP = EVENT_REQUEST
- Requested-Action AVP = [DIRECT_DEBITING | REFUND_ACCOUNT]

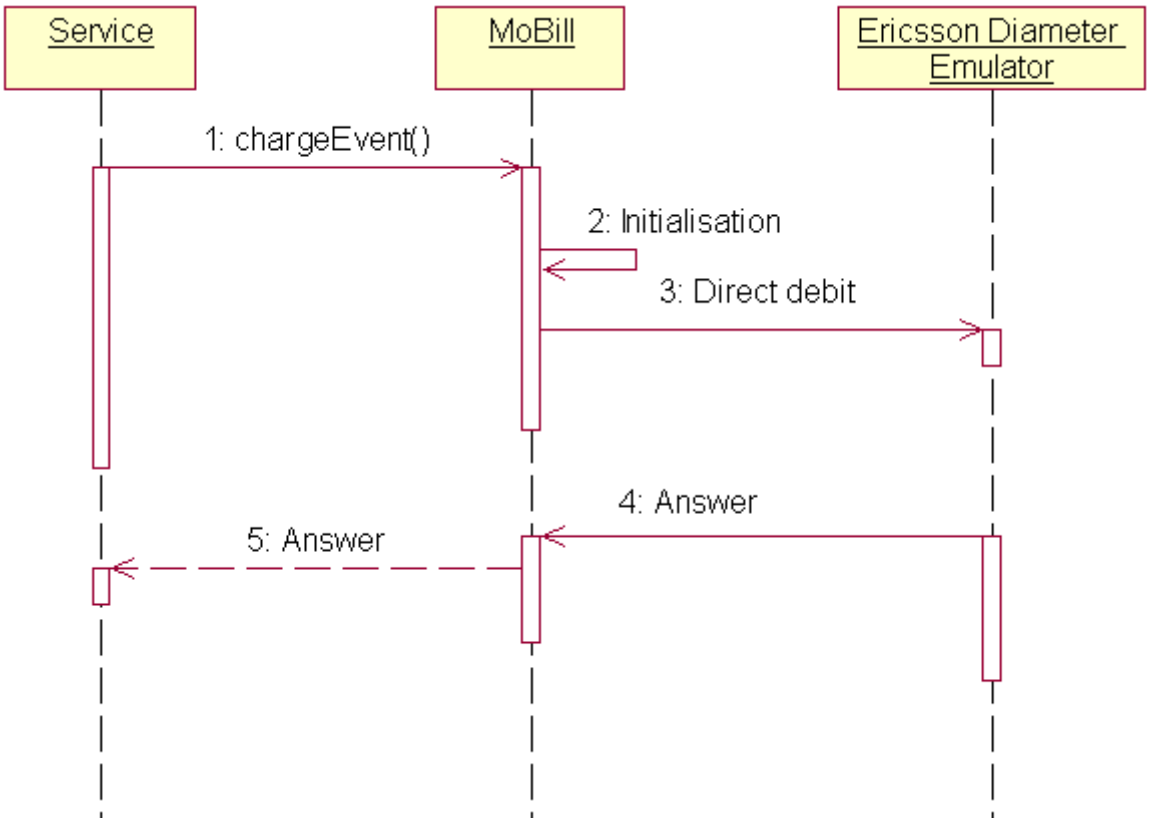
5.2.2 Charging Based on a Session

Session charging involves a number of additional steps namely start, stop, update and terminate a session.

For session charging we define a SCAPsession class. Firstly we start as session and after certain duration we update a session. When the session has ended we stop the session. Below we give more details to each method.

- The start() method will create and send a CCR message with CC-Request-Type AVP set to INITIAL_REQUEST.
- The update() method will create and send a CCR message with CC-Request-Type AVP set to UPDATE_REQUEST.
- The stop() method will create and send a CCR message with CC-Request-Type AVP set to TERMINATE_REQUEST.

Figure 5.2: Event-based Charging Model Sequence Diagram



5.3 MoBill's Charging Models

In the previous section we provided a brief overview of the Ericsson Diameter API. We discussed different classes that we used in the proposed component (MoBill). We gave a brief description of how we perform event and session charging using classes defined in the API. Different AVPs were shown to aid the discussion of CCR messages.

In this section we will discuss how we extended these two approaches to implement different charging models. We chose to illustrate the different charging models using sequence diagrams since they provide an easier representation of the algorithm than code. Appendix B provides a description of classes and methods in MoBill.

5.3.1 Event-based Charging

Event-based charging occurs when a user performs a single atomic transaction that raises an event.

Figure 5.2 shows the detailed description of how event-based charging is performed.

1. When the user service wants to perform charging, it calls the charge_Event() method in MoBill class.
2. MoBill performs initialisations. This includes setting up connection parameters, and starting the Diameter stack.
3. MoBill creates and sends CCR (credit control request) message to the Ericsson Emulator in order to perform a direct debit. The CCR message will contain CC-Request-Type AVP set to EVENT_REQUEST and Requested-Action AVP set to DIRECT_DEBITING.
4. The Ericsson Emulator server responds with an answer, indicating whether or not the transaction was successful.
5. The answer is returned to the service.

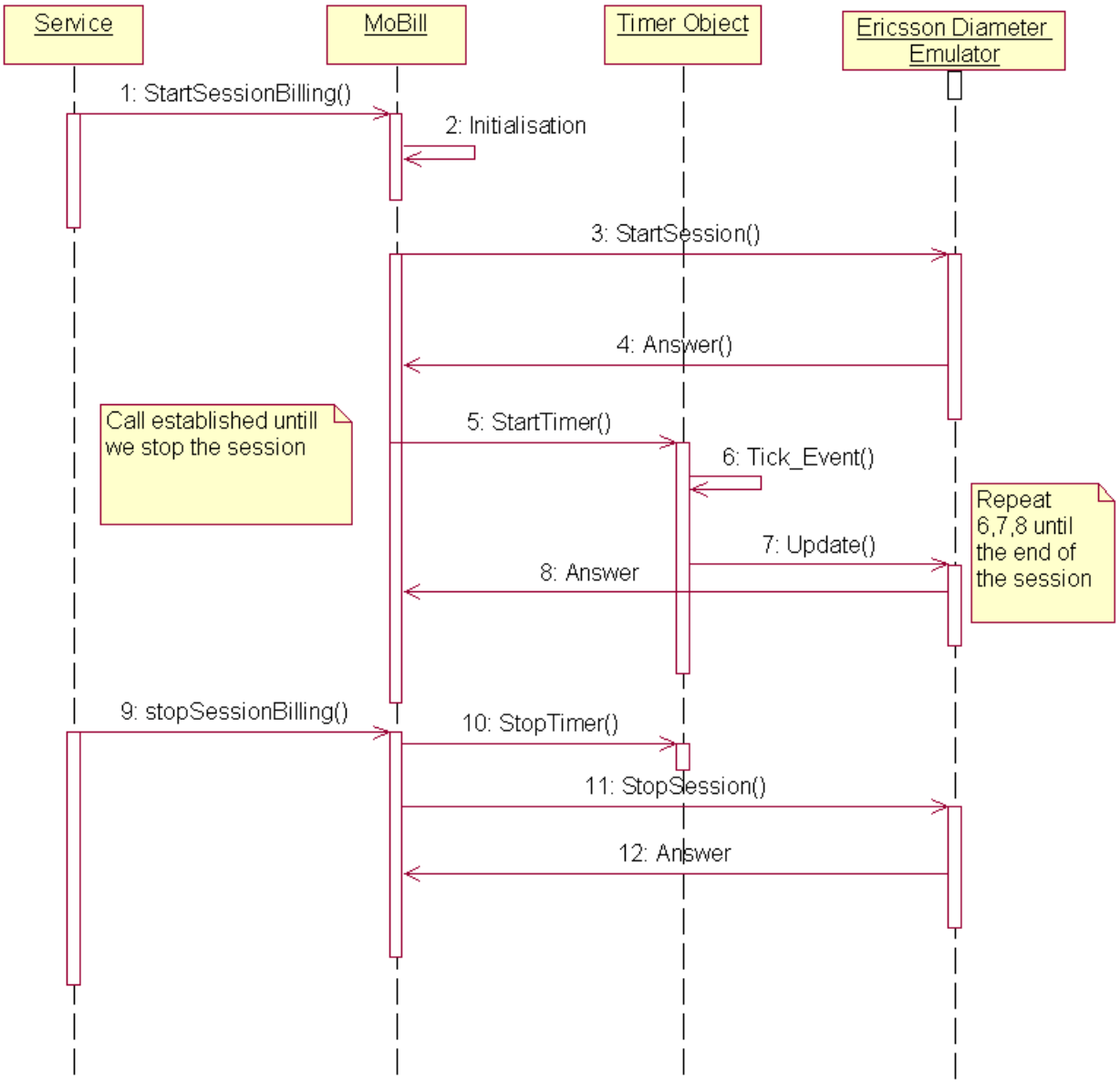
5.3.2 Time-based Charging

Time-based charging is billing for the duration of a session. To demonstrate time-based charging, we have implemented per second billing.

The sequence diagram, shown in Figure 5.3, shows detailed description of time-based charging operation. The details are outlined briefly below:

1. When a session is established within the user service, the charge_startSessionBilling() method in MoBill is called.
2. Similar to event-based charging, MoBill performs initialisations.
3. MoBill creates and sends CCR message to the Ericsson Emulator in order to start a session. The CCR message will contain the CC-Request-Type AVP that is set to INITIAL_REQUEST.
4. The Ericsson Diameter Emulator responds with an answer message.
5. MoBill calls the Start() method of the Timer object.
6. When a second lapses a tick event is generated by the Timer object.

Figure 5.3: Time-based Charging Model Sequence Diagram



7. MoBill sends a CCR message to the Ericsson Diameter Emulator in order to perform an update. The CC-Request-Type AVP is set to UPDATE_REQUEST.

8. The Ericsson Diameter Emulator will respond with an answer message indicating that the transaction was successful or not. Steps 6 – 8 get repeated until the end of the session.

9. When the session is complete, the service will call the charge_stopSessionBilling() method in MoBill.

10. MoBill calls the stopTimer() method.

11. The StopSession() method is called. This action will send a CCR message with the CC-Request_Type AVP set to TERMINATE_REQUEST.

12. The Ericsson Diameter Emulator responds with an answer.

5.3.3 Volume-based Charging

Volume-based refers to charging that uses amount of data transferred. The implementation for volume-based charging is similar to event based charging. However, with each event created, different rates apply based on transferred data.

The sequence diagram in Figure 5.4 shows the detailed description of volume-based charging. Below is a brief explanation for each step.

1. The service calls the charge_Volume(Kilobytes) method in MoBill, sending the number of bytes to MoBill.

2. The calculateRequestedQuota(Kilobytes) is initiated in MoBill. This method calculates the charge price based on the amount of kilobytes that are sent.

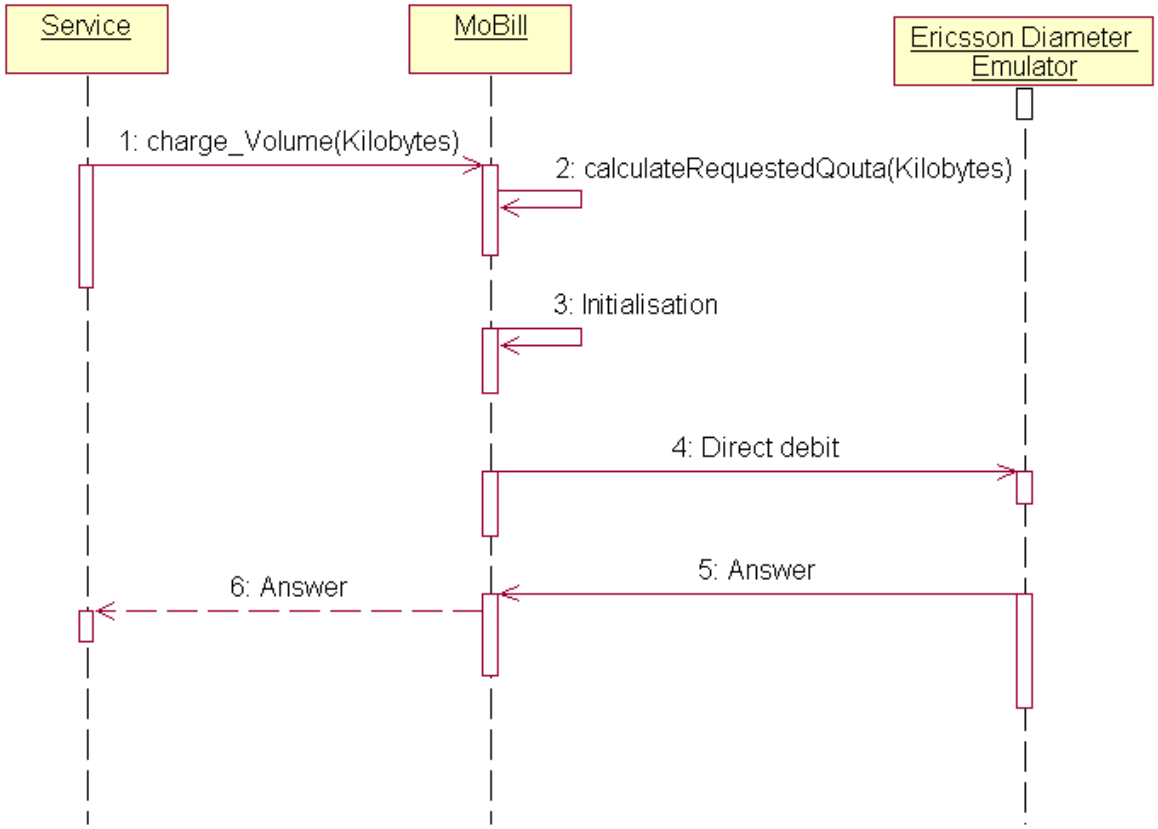
3. MoBill performs initialisations. These include setting up connection parameters, and starting the Diameter stack.

4. Similar to event based charging, MoBill creates and sends CCR message to the Ericsson Diameter Emulator in order to perform a direct debit.

5. The Ericsson Diameter Emulator responds with an answer, indicating whether or not the transaction was successful.

6. An answer is returned to the service.

Figure 5.4: Volume-based Charging



5.3.4 Reward-based Charging

With reward-based charging, billing is performed with the consideration of the user's selected criteria of use. For instance, the more the user uses a service; s/he may receive a discount. We have implemented reward-based charging by combining both incentive-based billing functions and event-based charging. For example, when a transaction is performed, e.g. downloading a video; we raise the first event to charge for the video. Similarly when the user accomplishes a task that affords them a reward, we use event-based charging to perform a refund on the user account.

Our implementation of reward-based charging counts the number of times the user performs a function that affords them a reward. The Video on Demand example, explained in Chapter 4, provides a suitable example to demonstrate this charging model.

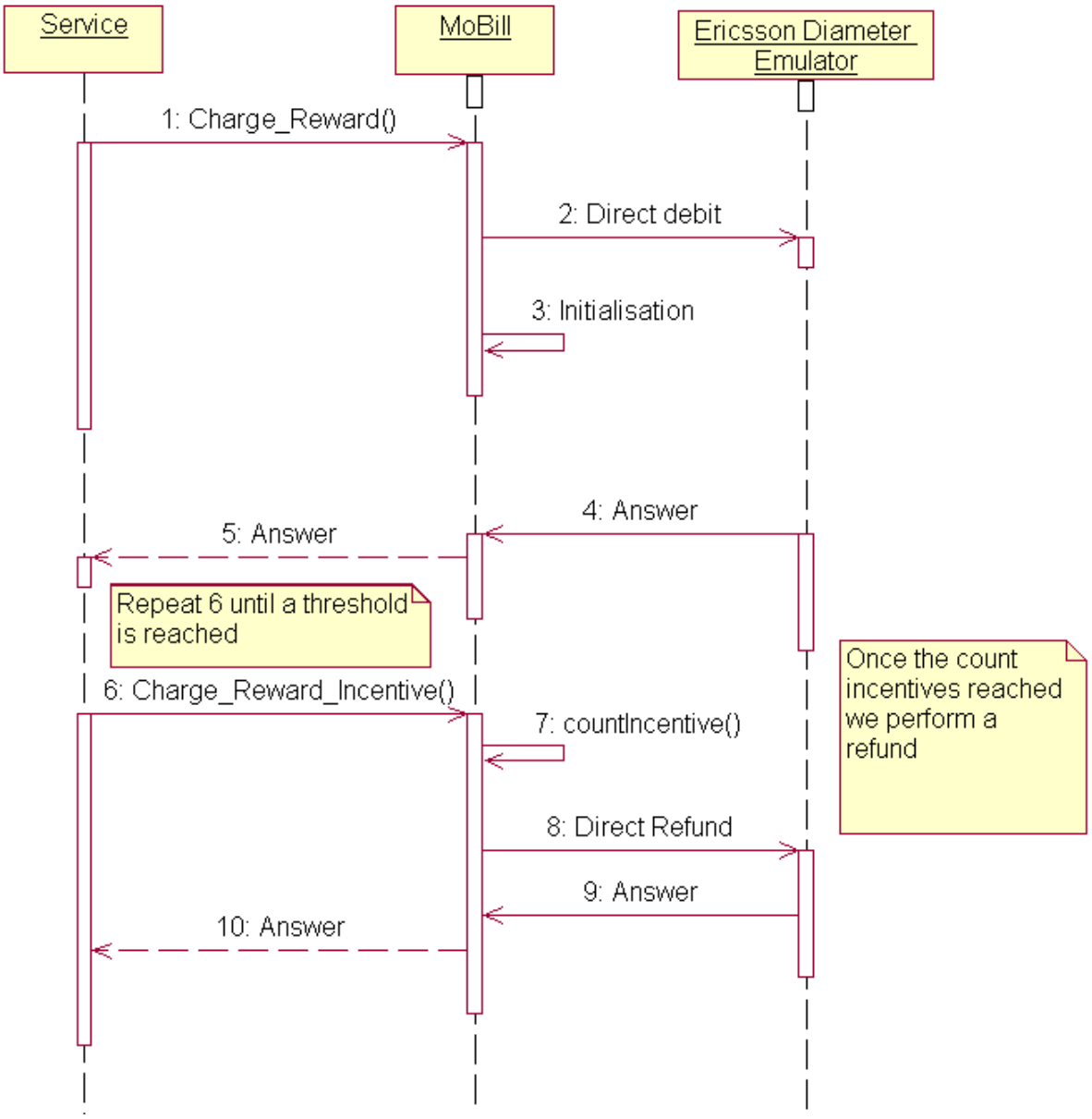
The sequence diagram in Figure 5.5 shows the detailed description of the reward-based charging. Below we provide a brief description for the steps shown in the diagram.

1. Steps 1 to 5 are similar to event-based charging, previously discussed.
2. In 6, the service will call the charge_Reward_Incentive() method when the user performs a task that affords him/her the ability to receive an incentive.
3. In 7, we count the number of incentives.
4. If the number of incentives equals the threshold, we send a CCR to perform direct refund (8). The CC-Request-Type AVP is set to REFUND_ACCOUNT.
5. In 9, the Ericsson Diameter Emulator responds with an answer. 10 An answer is returned to the service.

5.3.5 Flexible Session Charging

Flexible session charging model extends the time-based charging model. However with flexible session charging, the rates paid can change within a session. For instance, if a user upgrades a session their rate could be increased. Alternatively, it could be decreased if the user downgrades a session. The flexible session charging implements per second billing and allows for a convenient way to change the rate depending on the events raised by the user.

Figure 5.5: Reward-based Charging



Listing 1 Installation Command

```
mvn install:install-file -Dfile=charging.jar -DgroupId=com.ericsson -DartifactId=charg14
-Dversion=10.2.0 -Dpackaging=jarv
```

Sequence diagram shown in Figure 5.6 provides a detailed description of this charging model. The steps are briefly summarised below:

1. The first eight steps are similar to time-based charging explained earlier; except we need to set the initial charging level (3). This will set the pricing level. There are three levels prescribed, level 1, 2, 3 with each level having a different charge price associated with it.

2. In 10, the user raises an event to change the level by calling the method charge_ FlexibleMultiSessionLevel_N, where N is the level number.

3. In 11, the quota is changed based on the level specified. We change the price per second that the user is billed on the next second.

4. The remaining steps are similar to time-based charging.

5.4 Adding MoBill to a Mobicents Application

To deploy Mobicents examples we make use of Maven [17]. Maven uses xml files (POM files) that contain configurations about the project. The jar file containing MoBill needs to be installed on the local repository in order for the example to compile. Listing 1 shows the installation command. This adds the Charging.jar (contains MoBill) to the local repository. Different parameter such as artifact ID, version, and group ID are also defined for the jar file.

After this installation we need to add the dependency in an xml POM file in order for Maven to include this file when deploying the examples [29]. Figure 5.7 contains the dependency added.

The jar file needs to be placed in the library directory on Mobicents server in order for it accessed during run time. View Appendix C.2 for the installation instructions.

Figure 5.6: Flexible Multi-session Charging Sequence Diagram

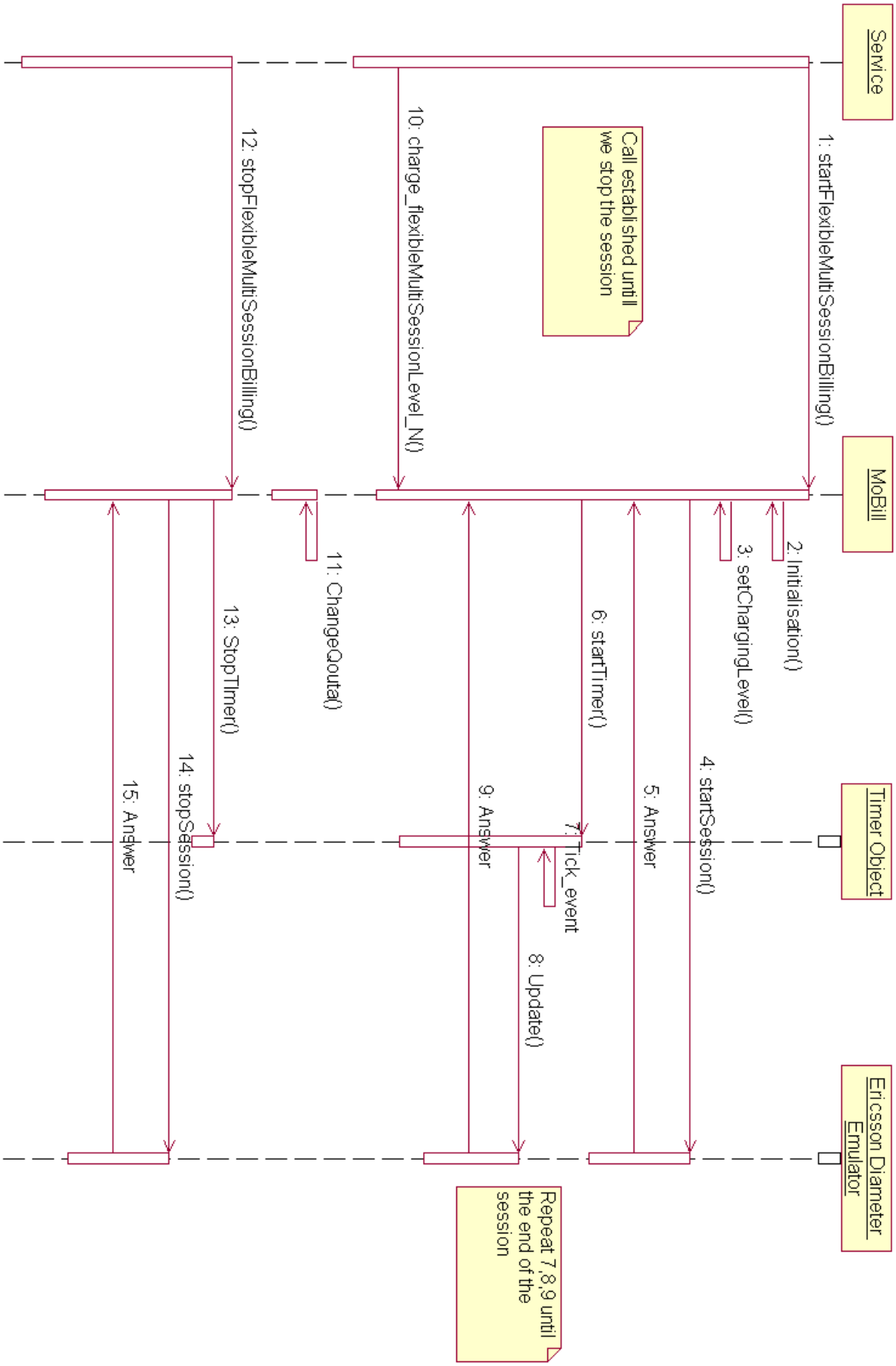


Figure 5.7: Adding a Dependency

```
<dependency>
        <groupId>com.ericsson</groupId>
        <artifactId>charg14</artifactId>
        <version>10.2.0</version>
        <scope>compile</scope>
</dependency>
```

5.5 Testing Examples

As proof of concept and to test various charging models, we have a number of testing examples that make use of the Billing Component to charge users for services rendered. The main emphasis was not on the development of new services or making use of sophisticated services, but rather it was on demonstrating different concepts.

Mobicents is a service development platform that we use for the project. Before we can deploy the examples, Mobicents Server needs to be installed. Appendix C discusses the required installations and configurations. Ericsson Diameter Emulator acts as a Prepaid System that responds to charging requests from a client application. Setup procedure for the Ericsson Diameter Emulator is discussed in appendix E.

We have extended two Mobicents examples, namely: the Google Talk Bot and SIPB2BUA. We have made use of Video on Demand service developed at Rhodes University [19] to demonstrate reward-based charging model. For all these, we add additional billing logic that uses the Billing Component.

5.5.1 Google Talk Bot Example

This example uses XMPP (Extensible Messaging and Presence Protocol) resource adapter. XMPP is used for real-time communication and supports applications such as instant messaging, presence, voice and video calls [28].

Figure 5.8 shows details about the example. A message is sent from the user to the Google Talk Bot service. The Google Talk Bot service performs billing before sending a reply message. In our example, we send a message back if the transaction was successful. Alternatively, if the transaction was unsuccessful, i.e. no credit, an error message is sent back.

Figure 5.8: Google Talk Bot Sequence Diagram

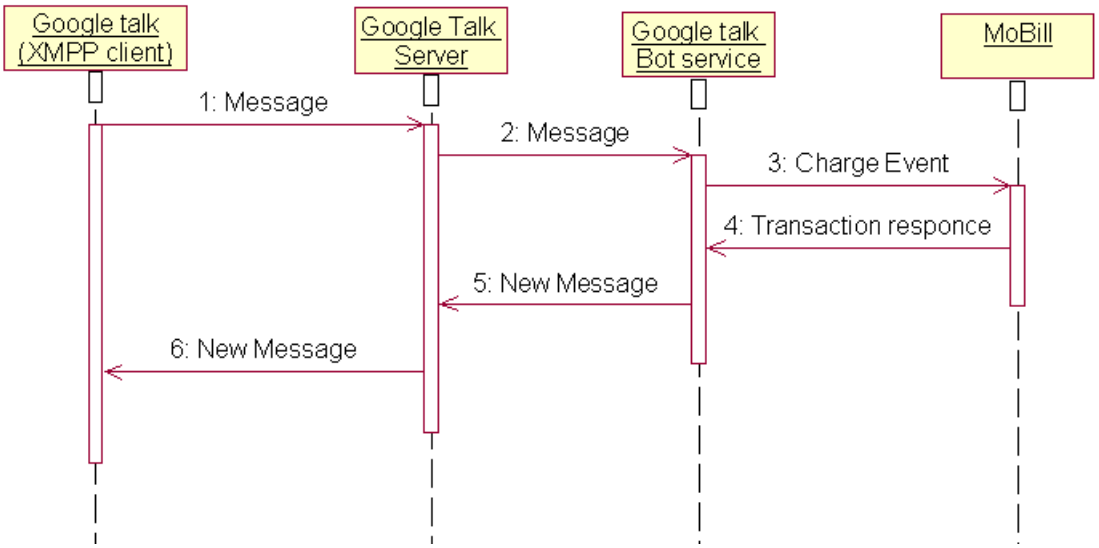


Figure 5.9: Google Talk Bot Example

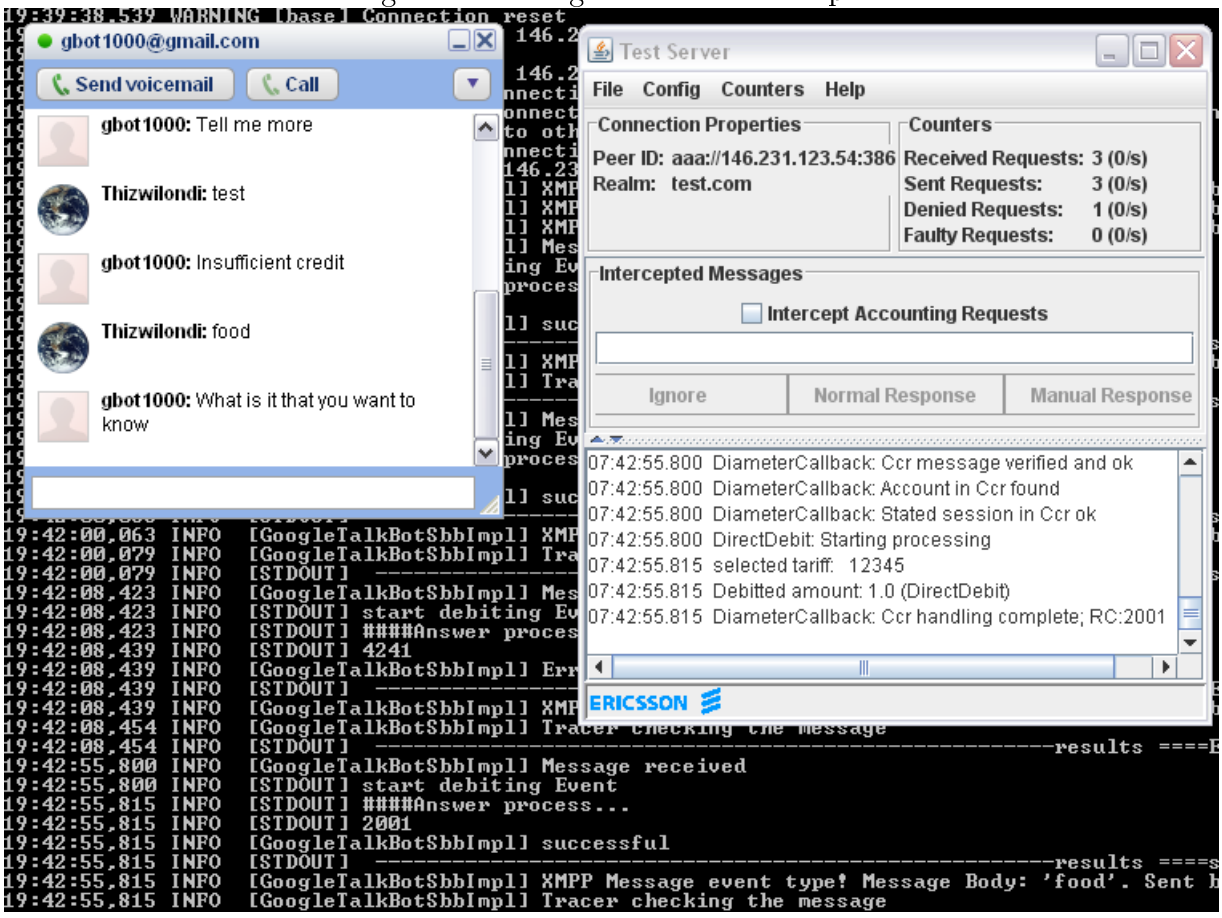


Figure 5.10: SIPB2BUA Example

Appendix D.1 discusses the installation of Google Talk Bot example. Figure 5.9 shows a screenshot of the Google Talk client, Ericsson Diameter Emulator, and Mobicents server in the background. Initially the user's account credit is set to zero. The user attempts to send a message. The Ericsson Diameter Emulator shows a denied request received. We proceed by giving the user credit. When the user sends a message we can see that the request is successful (sent and received request displayed on the Ericsson Diameter Emulator). This example demonstrates how we can use Google Talk Bot example to implement event-based charging model.

5.5.2 SIPB2BUA Example

The sip back-to-back user agent (SIPB2BUA) example is a call controlling service. The example separates communication session into two call legs. This allows developers to add additional code to perform accounting of resources and prepaid billing.

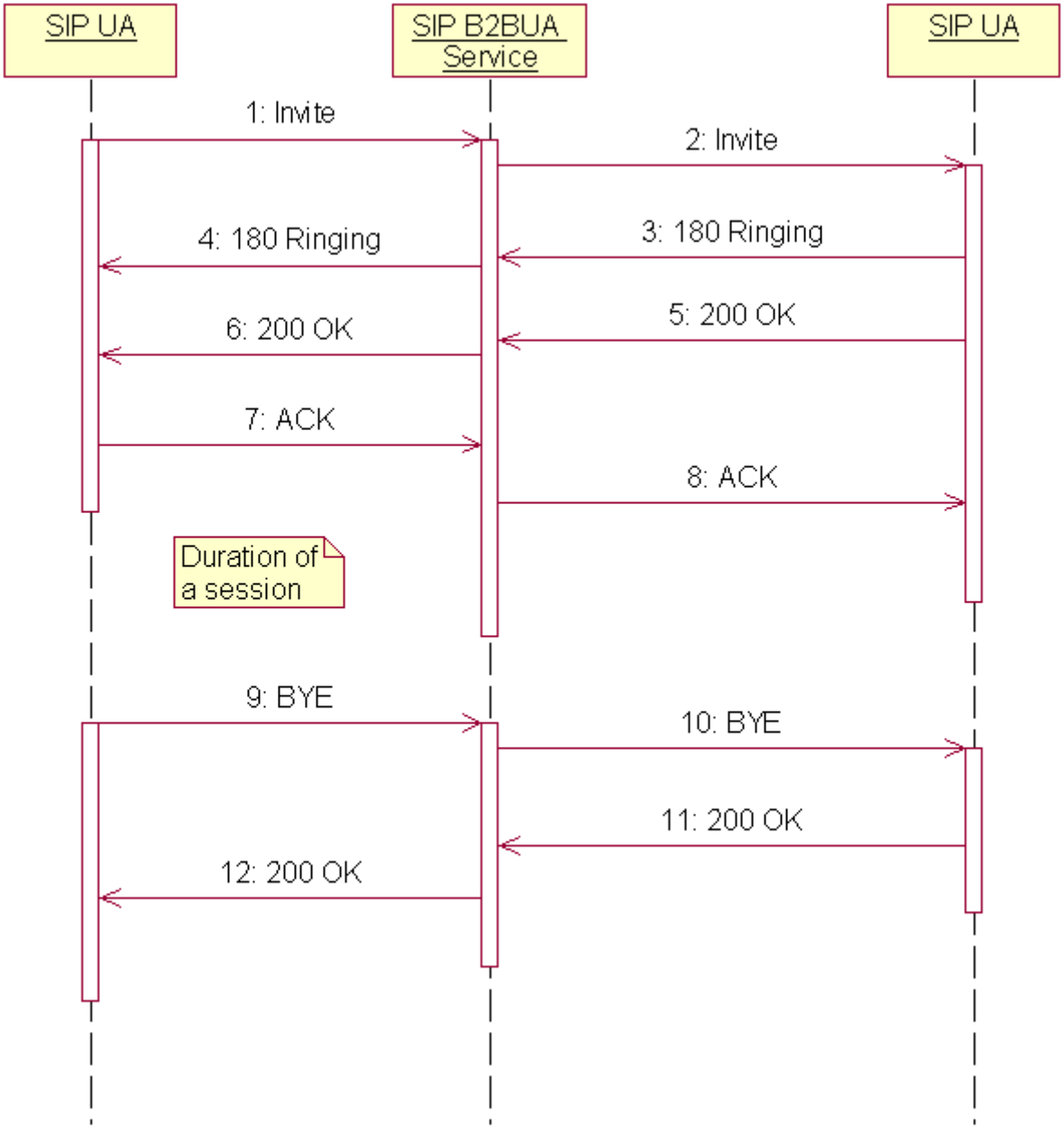
The general architecture of the example has three main components:

1. SIP UA: The originating user agent (UAS).
2. SIPB2BUA service: The service manages the interaction between the user agents.
3. SIP UA: The answering user agent (UAC).

As it can be seen from Figure 5.10, the B2BUA service handles SIP [27] messages between user agents. Therefore the B2BUA acts as a mediator between communicating SIP user agents. This example provides us with the ability to add billing logic in the SIPB2BUA service.

Figure 5.11 shows how a SIP session would be established between the two user agents. The SIP session is established as follows:

Figure 5.11: Sequence Diagram with SIPB2BUA Service



1. To initiate a session, the originating user agent sends an invite message.
2. The answering user agent responds with a message 180 ringing.
3. When the second user picks up, the answering user agent; a 200 OK message is sent to the originating user agent.
4. The originating user agent receives the 200 0K message and responds with an ACK message. At this point a session is established, this is where accounting for the use of resources is initiated.
5. Either the originating user agent or the answering user agent hangs up; a SIP BYE message is sent to the other user agent. The receiving user agent will respond with a 200 OK message. At this point, we stop accounting on resource usage.

To add MoBill to the SBB (Service Building Block) used for the B2BUA service, we need to import the jar files containing MoBill. The process was previously explained in section 5.4. At class level, we define a MoBill object. Additionally we define a Boolean parameter to indicate if billing has already been stopped or not.

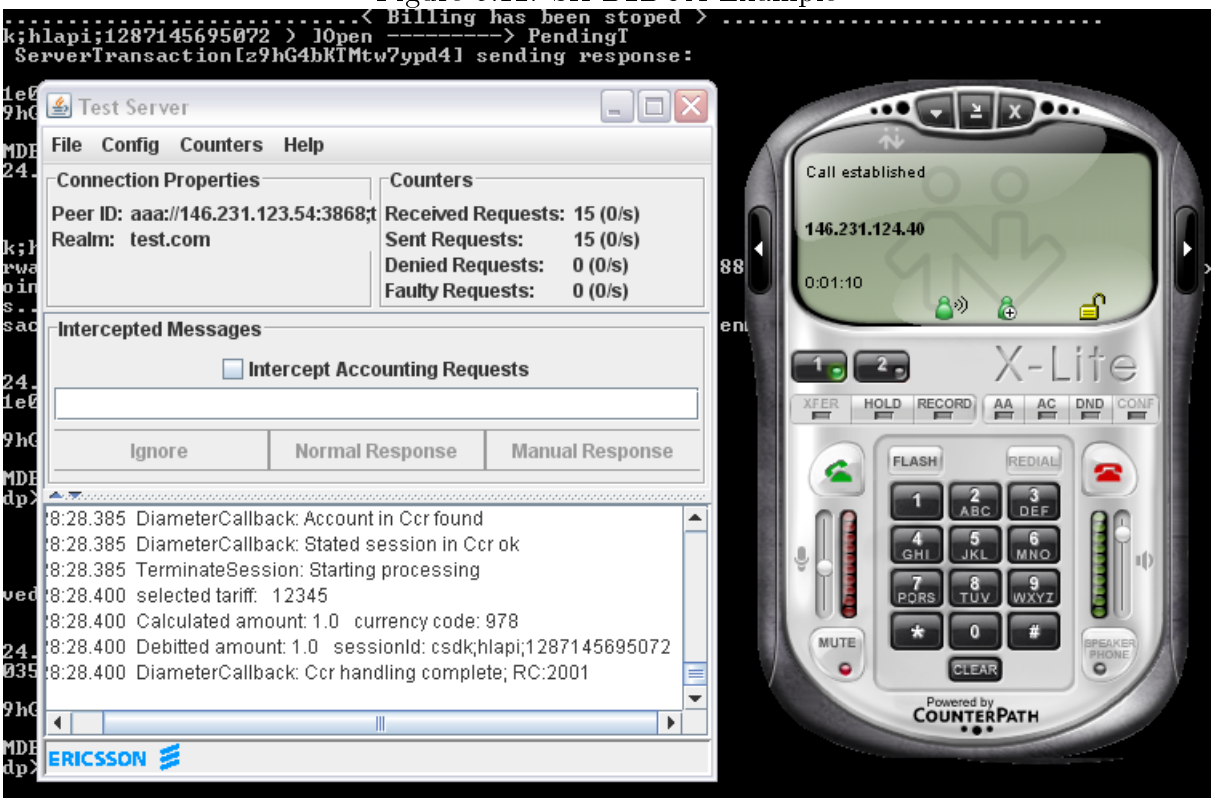
The SIPB2BUA SBB contains a number of event handlers that handle a number of SIP events. We list and explain event handlers that need to be modified below.

- onInviteEvent() – When an Invite is received, the initial method that is called. Initialisation code is placed here.
- on2xxResponse() – Handles responses starting with 2 such as 200 OK. We start billing once we have received 200K.
- onBye() – Raised when a BYE message is received. This indicates the end of the session. At this point we added code to stop billing. Since either of the user agents can terminate the session we needed to check that we have not already stopped billing. The onBye() event handler handles BYE messages from both the originating user agent and answering user agent.

After these changes, we can deployed SIPB2BUA example. Appendix D.2 shows how to deploy the example.

To execute the SIPB2BUA example, we start the Mobicents server. In order to test SIPB2BUA service, we shall make use of two SIP phones. The originating client needs

Figure 5.12: SIPB2BUA Example



to be configured in order to connect to the Mobicents server. View appendix D.2.4 for these configurations. When a call is made, diameter messages are sent to the Ericsson Diameter Emulator. The Ericsson Diameter Emulator responds with an answer for each request message sent. In Figure 5.12, we can see the received and sent requests on the Ericsson Diameter Emulators window.

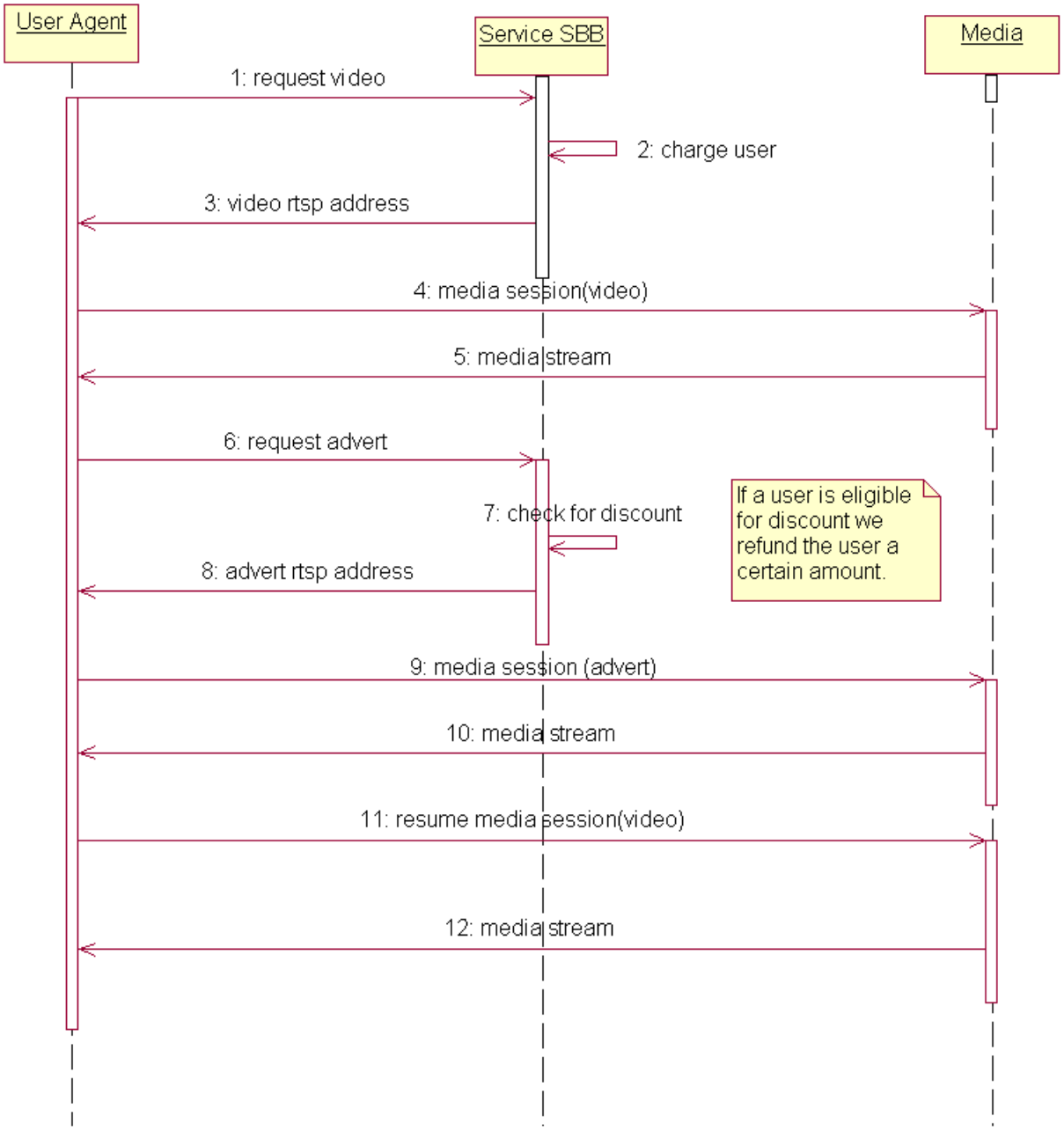
To provide a summary in this example we demonstrated how we can use the SIPB2BUA Mobicents example to add accounting of resource usage. We have shown how we can implement the time-based charging model.

5.5.3 Video on Demand Example

The VoD (video on demand) example allows a user to stream videos. A user may optionally stream through advertisements during the session. The user pays for each video requested. In order to create an incentive for users to watch advertisements, a discount is given to the user.

Figure 5.13 shows the details of the video on demand example.

Figure 5.13: Video on Demand Sequence Diagram



1. The user agent requests a video.
2. The SBBs containing service logic perform billing. We use reward-based charging model to perform a debit on user account.
3. The video rtsp address containing video address is returned to the user agent.
4. The user agent initiates a session with the media streaming server.
5. The server responds with media stream for video.
6. The user agent requests an advertisement.
7. Based on the policy specified, we check if the user deserves a discount. If the user meets the criteria then we perform credit on his/her account.
8. The video rtsp address containing advertisement address is returned to the user agent.
9. The user agent pauses the current media session and initiates another media session.
10. The server responds with media stream for video.
11. At the end of the advertisement, the previous media session (for the requested video) is resumed.
12. The server responds with media stream for video.

Figure 5.14 shows the VoD client with a video. The Ericsson Diameter Emulator shows the received requests from the client.

5.6 Summary

The first part of the chapter focused on the implementation details of MoBill. We used UML diagrams to show how we implemented event, time, volume, reward, and flexible charging model. We have shown with sequence diagrams how we extended the core charging models to produce extended charging models.

The second part was focused on testing MoBill by using it on different examples. The Google Chat Bot was used to demonstrate event-based charging. The SIPB2BUA example

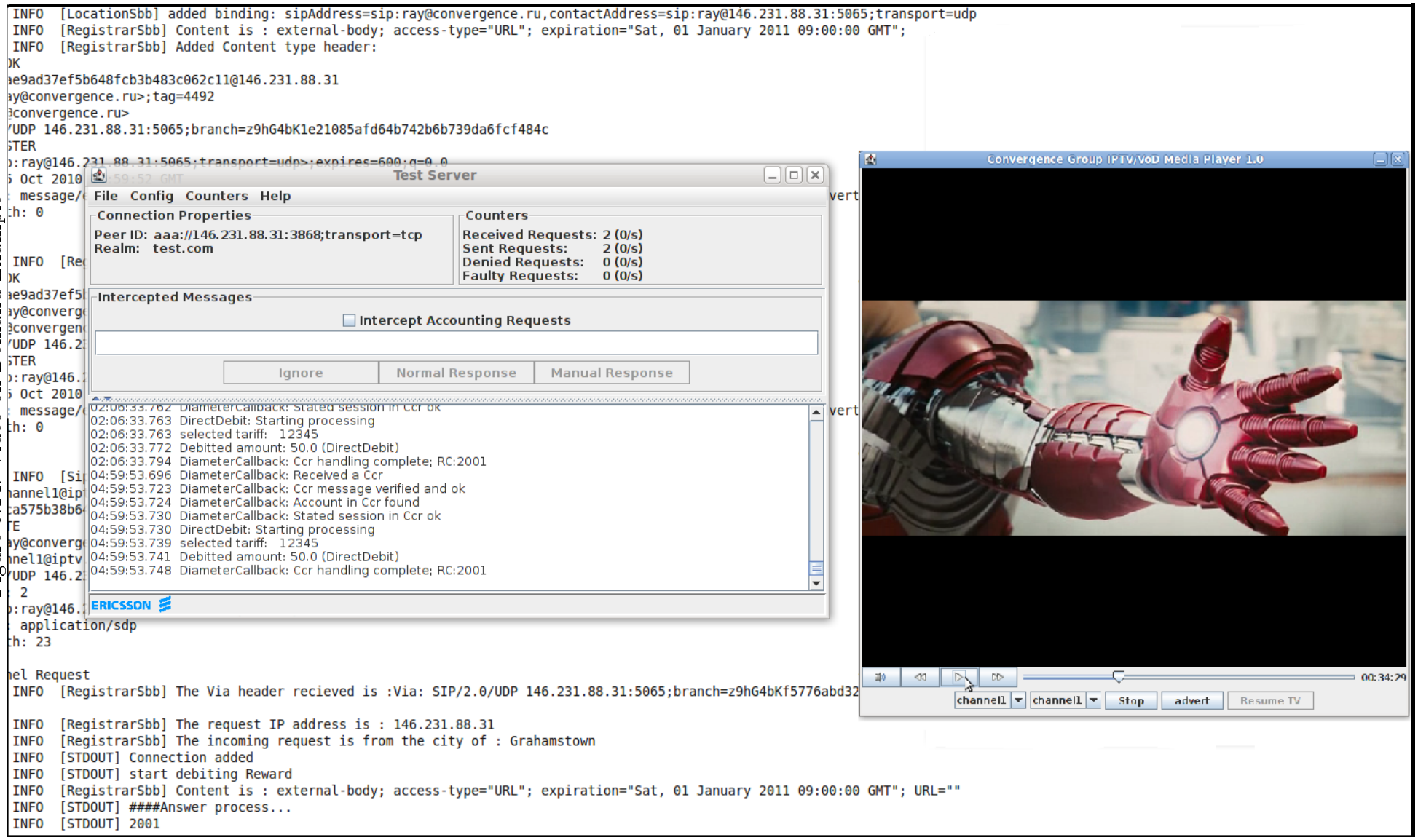


Figure 5.14: Video on Demand Example

was used to demonstrate time-based charging model. We used the video on demand example to demonstrate reward-based charging example. We demonstrated how MoBill can be used to perform different charging models required by different examples.

Chapter 6

Conclusion

6.1 Overview

Factors such as composition of services, variety of services, different service providers, and QoS (quality of service) adds complexity to billing for next generation services. When developing billing system, it is important to apply recommended principles. There are a number of charging models available for services. Using the right charging model is important for the success of the service.

Mobicents provides an open source service development that allows developers to combine voice, video and data. The Mobicents platform contains the IMS (IP Multimedia Subsystem) core. The Ericsson Diameter Charging SDK contains a number of tools that allow developers to develop charging applications. The Diameter API provides high-level implementation of Diameter. The Ericsson Diameter Emulator acts as a potential prepaid system that responds to client requests.

We have developed a centralised billing component (MoBill) that can easily be added to services requiring billing. MoBill implements core and extended charging models. Core charging models consist of: event, time and volume. By combining the core charging models, we produced more sophisticated charging models which we refer to as extended charging models. Reward and flexible session models are the two examples of extended charging model. As proof of concept we have enhanced Mobicents examples to demonstrate how we can use MoBill to perform various charging models.

6.2 Revisiting Projects Objectives

The main objective of the research project was to develop a billing system for Mobicents applications. This was achieved by developing a high abstract billing component called MoBill that enables developers with less understanding of IMS billing and the Diameter protocol to use within their service.

The second objective was to test the billing system. Mobicents examples were used as test services. We have extended the Google Chat Bot to demonstrate event-based charging model. Similarly we extended the Mobicents SIPB2BUA example to demonstrate time-based charging. To demonstrate reward-based charging we used a video on demand example. We used the Ericsson Diameter Emulator as a prepaid system that responds to our accounting requests. Based on different charging models, the Ericsson Diameter Emulator performed debits and credits on user accounts.

6.3 Possible Future Work

The system focuses primarily on implementing core charging models. We have demonstrated through examples of reward-based and flexible session billing how we can combine the core charging models to produce more sophisticated charging models. One possible extension to the project would be produces additional innovative charging models based on the core ones.

Diameter protocol performs AAA (Authentication, Authorisation, and Accounting). The research project primarily focused on Accounting of resources without considerations for security. An extension to the project would be to investigate security threats with regards to billing next generation services.

In the design approach we have chosen to develop a java object (billing component) that can be instantiated in mobicents SBBs (service building blocks) in order to implement various charging model. A minor extension would be to provide a wrapper for the object such that it responds to SLEE events. In this case we incorporate the MoBill into a SBB.

References

[1] 3GPP. *Charging and billing.* TS 22.115, 3rd Generation Partnership Project (3GPP), 2006.

[2] 3GPP. *Charging Managment, Charging Architecture and Principles.* TS 29.278, 3rd Generation Partnership Project (3GPP), 2009.

[3] A. Alonistioti, N. Houssos, S. Panagiotakis, M. Koutsopoulou, and V. Gazis. Intelligent architectures enabling flexible service provision and adaptability. In *Wireless Design Conference (WDC 2002)*, 2002.

[4] M.E. Barachi, R. Glitho, and R. Dssouli. Charging for Multi-grade Services in the IP Multimedia Subsystem. In *Next Generation Mobile Applications, Services and Technologies, 2008. NGMAST'08. The Second International Conference on*, pages 10–17, 2008.

[5] FC Bormann, S. Flake, and J. Tacken. Business models for local mobile services enabled by convergent online charging. *Mobile and Wireless Communications Summit, 2007. 16th IST*, 16:pp. 1–5, 2007.

[6] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko. Diameter Base Protocol. RFC 3588, Internet Engineering Task Force, September 2003.

[7] Gonzalo Camarillo and Miguel García-Martín. *The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds.* John Wiley & Sons, Ltd, 3rd edition, 2008.

[8] D.D. Clark. A model for cost allocation and pricing in the Internet. In *MIT Workshop on Internet Economics*, volume 49, page 13, 1995.

[9] Jean Deruelle. Mobicents communications platform, 2008. Available from http://people.redhat.com/vralev/blog/JavaOne 2008 Presentation - Jean Deruelle.pdf; accessed 01 June 2010.

[10] M. El Barachi, R. Glitho, and R. Dssouli. Context-aware signaling for call differentiation in IMS-based 3G networks. In *12th IEEE Symposium on Computers and Communications, 2007. ISCC 2007*, pages 789–796, 2007.

[11] Ericsson. Ericsson developer tools, 2010. Available from http://devtools.ericsson.com/charging/-sdk/overview; accessed 20 October 2010.

[12] Victor I. Fajardo. Open diameter c++ api, January 2005. Available from http://diameter.sourceforge.net/diameter-api/index.html; accessed 20 October 2010.

[13] H. Hakala, L. Mattila, J. Koskinen, M. Stura, and J. Loughney. Diameter Credit-Control Application. RFC 4006, Internet Engineering Task Force, August 2005.

[14] P. Kurtansky, B. Stiller, and TIK Series. Prepaid charging for QoS-enabled IP services based on time intervals. In *First International Workshop, WINE*, pages 15–17, 2005.

[15] Eduardo Martins and Bartosz Baranowski. Mobicents JAIN SLEE SIP B2BUA Example User Guide, 2010. Available from http://www.mobicents.org/slee/docs/examples/sip-b2bua/2.2.0.FINAL/en-US /pdf/Mobicents_SLEE_Example_SIP_B2BUA _User _Guide.pdf; accessed 10 May 2010.

[16] Eduardo Martins and Ivelin Ivanov. Mobicents jain slee google talk bot example user guide, 2010. Available from http://www.mobicents.org/slee/docs/examples/google-talk-bot/2.2.0.FINAL/en-US/pdf/Mobicents_SLEE_Example_Google_Talk_Bot_User_Guide.pdf; accessed 10 May 2010.

[17] Maven. Maven in 5 minutes, 2010. Available from http://maven.apache.org/guides/gettingstarted/maveninfiveminutes.html; accessed 20 October 2010.

[18] Alexandre Mendonça. Mobicents JAIN SLEE Diameter Base Resource Adaptor User Guide, 2010. Available from http://www.mobicents.org/slee/docs/resources/diameter-ro/2.0.0.GA/en-US/pdf/Mobicents _SLEE _RA _DIAMETER_RO _User _Guide.pdf; accessed 10 May 2010.

[19] Ray Musvibe and Alfredo Terzoli. Towards a customised video on demand service for the mobicents platform. In *Southern Africa Telecommunication Networks and*

Applications Conference (SATNAC 2010), 2010. Spier Estate, Stellenbosch, South Africa.

[20] Magnus Nilsson. Diameter Charging SDK Practicalities, 2008. Available from http://a178.g.akamai.net/7/178/8211/0001/ ericsson-com.download.akamai.com/8211 /high1/mainframe _low.htm; accessed 10 May 2010.

[21] Andrew Odlyzko. Paris metro pricing for the internet. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 140–147, 1998.

[22] H. Oumina and D. Ranc. Towards a real time charging framework for complex applications in 3GPP IP Multimedia System (IMS) environment. In *Next Generation Mobile Applications, Services and Technologies, 2007. NGMAST'07. The 2007 International Conference on*, pages 145–150, 2007.

[23] V.G. Ozianyi, N. Ventura, and E. Golovins. A novel pricing approach to support QoS in 3G networks. *Computer Networks*, 52(7):1433–1450, 2008.

[24] Vitalis G. Ozianyi and Neco Ventura. Service outsourcing and billing in inter-domain ims scenarios. In *Southern African Telecommunication Networks and Applications Conference (SATNAC 2009)*, August 2009.

[25] S. Panagiotakis, M. Koutsopoulou, and A. Alonistioti. Business models and revenue streams in 3g market. In *14th IST Mobile and Wireless Communication Summit*, 2005.

[26] S. Panagiotakis, M. Koutsopoulou, A. Alonistioti, and A. Kaloxylos. Generic framework for the provision of efficient location-based charging over future mobile communication networks. 2:755–759, 2002.

[27] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. Session Initiation Protocol. RFC 3261, Internet Engineering Task Force, December 2002.

[28] Ed. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920, Internet Engineering Task Force, October 2004.

[29] Sonatype. Project dependencies, 2010. Available from http://www.sonatype.com/books/mvnref-book/reference/pom-relationships-sect-project-dependencies.html; accessed 20 October 2010.

Appendix A

System Class Diagram

A.1 Detailed Class Diagram

Figure A.1 shows the class diagram of the billing component with parameters and additional methods. Additional Methods are described in the next section.

A.2 Additional Methods Description

A.2.1 DiameterListener

- getResultCode() - This method returns the result code from the Ericsson Diameter Emulator.

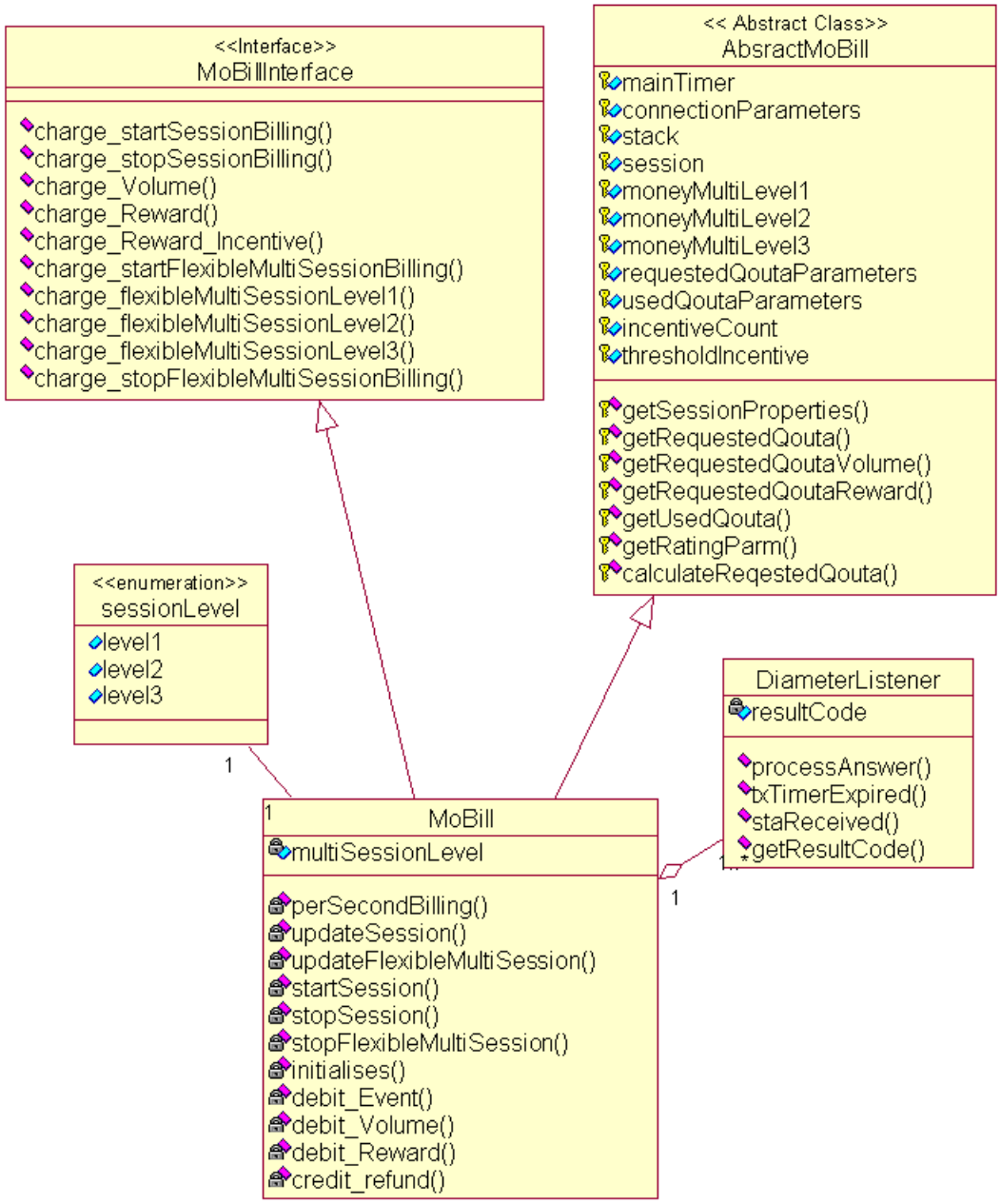
A.2.2 AbstractBillingComponent

- calculateRequestedQuota() - calculates the requested quota based on the amount of Kilobytes downloaded. Used for volume based charging.

A.2.3 BillingComponent

- startSession() - start session billing.

Figure A.1: Billing Component Detailed Class Diagram



- stopSession() - stop session billing.
- credit_refund() - performs a credit on user account used for reward based charging.

A.3 SessionLevel enumeration

- Used for stating the session level (level1, level2, level3).

Appendix B

MoBill JavaDocs

We named the billing component MoBill because the naming happened later in the history of this project, portions of the code still use 'billing component'. So, while reading through the code and documentation, keep in mind that MoBill is referred to as 'Billing Component'.

B.1 MoBillInterface

B.2 AbstractMoBill

\

B.3 MoBill

Figure B.1: MoBillInterface (BillingComponentInterface)

comp

Interface BillingComponentInterface

All Known Implementing Classes:

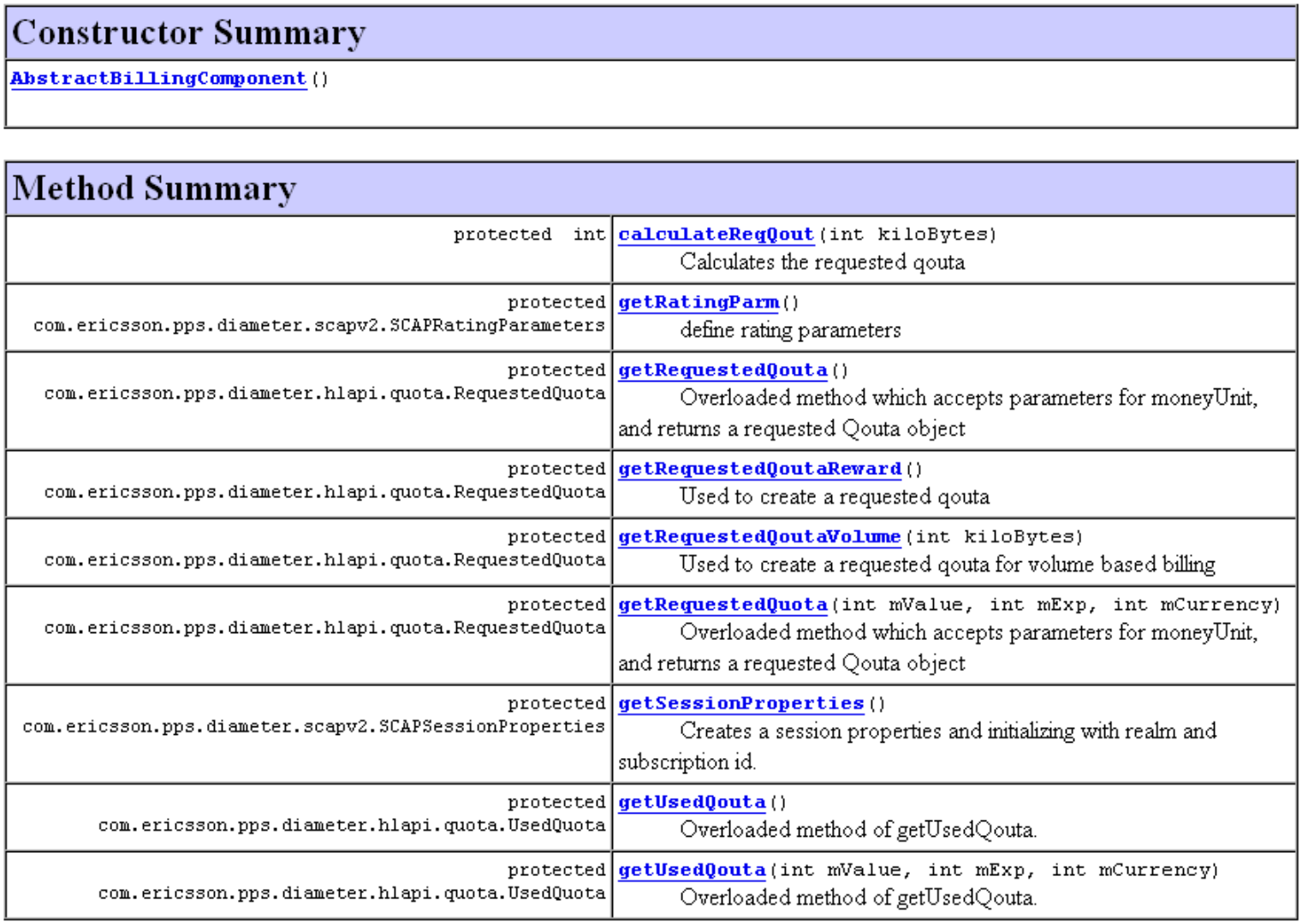
BillingComponent

`public interface` **`BillingComponentInterface`**

Billing Component Interface that is used to define public methods that must be implemented by the Billing Component

Method Summary

`java.lang.String`	**`charge Event`**`()` Event based charging model
`void`	**`charge flexibleMultiSessionLevel1`**`()` flexibleMultiSessionBilling charging model (Set the session level to one)
`void`	**`charge flexibleMultiSessionLevel2`**`()` flexibleMultiSessionBilling charging model (Set the session level to two)
`void`	**`charge flexibleMultiSessionLevel3`**`()` flexibleMultiSessionBilling charging model (Set the session level to three)
`java.lang.String`	**`charge Reward Incentive`**`()` reward based charging model.
`java.lang.String`	**`charge Reward`**`()` reward based charging model
`void`	**`charge startFlexibleMultiSessionBilling`**`()` flexibleMultiSessionBilling charging model (Start session billing)
`void`	**`charge startSessionBilling`**`()` time based charging model.
`void`	**`charge stopFlexibleMultiSessionBilling`**`()` flexibleMultiSessionBilling charging model (Stop session billing)
`void`	**`charge stopSessionBilling`**`()` time based charging model.
`java.lang.String`	**`charge Volume`**`(int kiloByte)` volume based charging model



Constructor Summary

AbstractBillingComponent()

Method Summary

protected int	**calculateReqQout**(int kiloBytes) Calculates the requested qouta
protected com.ericsson.pps.diameter.scapv2.SCAPRatingParameters	**getRatingParm**() define rating parameters
protected com.ericsson.pps.diameter.hlapi.quota.RequestedQuota	**getRequestedQouta**() Overloaded method which accepts parameters for moneyUnit, and returns a requested Qouta object
protected com.ericsson.pps.diameter.hlapi.quota.RequestedQuota	**getRequestedQoutaReward**() Used to create a requested qouta
protected com.ericsson.pps.diameter.hlapi.quota.RequestedQuota	**getRequestedQoutaVolume**(int kiloBytes) Used to create a requested qouta for volume based billing
protected com.ericsson.pps.diameter.hlapi.quota.RequestedQuota	**getRequestedQuota**(int mValue, int mExp, int mCurrency) Overloaded method which accepts parameters for moneyUnit, and returns a requested Qouta object
protected com.ericsson.pps.diameter.scapv2.SCAPSessionProperties	**getSessionProperties**() Creates a session properties and initializing with realm and subscription id.
protected com.ericsson.pps.diameter.hlapi.quota.UsedQuota	**getUsedQouta**() Overloaded method of getUsedQouta.
protected com.ericsson.pps.diameter.hlapi.quota.UsedQuota	**getUsedQouta**(int mValue, int mExp, int mCurrency) Overloaded method of getUsedQouta.

Figure B.2: AbstractMoBill (AbstractBillingComponent)

Figure B.3: MoBill (BillingComponent)

Constructor Summary

BillingComponent()	Constructor

Method Summary

java.lang.String	**charge Event**() This method initiates the event-based charging.
void	**charge flexibleMultiSessionLevel1**() Used to set the sessionLevel to one when using Flexible Session Charging Model
void	**charge flexibleMultiSessionLevel2**() Used to set the sessionLevel to two when using Flexible Session Charging Model
void	**charge flexibleMultiSessionLevel3**() Used to set the sessionLevel to three when using Flexible Session Charging Model
void	**charge MultiSessionService1**() Used to initiate multisession billing
java.lang.String	**charge Reward Incentive**() Everytime the user select an incentive, we keep count, when the count has been achieved we give a discount.
java.lang.String	**charge Reward**() This method is used to initiate Reward based billing.
void	**charge startFlexibleMultiSessionBilling**() Starts flexible session charging.
void	**charge startSessionBilling**() Starts Time based billing
void	**charge stopFlexibleMultiSessionBilling**() Stops flexible session charging.
void	**charge stopSessionBilling**() Stops the session billing.
java.lang.String	**charge Volume**(int kiloByte) Initiate volume based charging

Appendix C

Installations and Configurations

This appendix discusses the installation of the environment and the tools required.

C.1 Configuring the Mobicents server

Mobicents and associated tools installation process is discussed in this section.

C.1.1 Downloading Mobicents

Mobicents JAIN SLEE can be downloaded from:

http://sourceforge.net/projects/mobicents/files/Mobicents%20JAIN%20SLEE%20Server/

Mobicents JAIN SLEE Server version 2.1.1 GA was used in this research. Select the the version and download to a directory on local computer. Uzip the file to your selected folder, e.g. "c:\server\".

C.1.2 Configuring JBOSS

The JBOSS variable needs to be set by editing the environmental variable. The JBOSS_HOME variable contains the BIN directory. In our example we have configured the variable to "C:\servers\mobicents-jainslee-2.1.1.GA-jboss-5.1.0.GA\jboss-5.1.0.GA". Additionally add the BIN directory into the PATH i.e. "C:\servers\mobicents-jainslee-2.1.1.GA-jboss-5.1.0.GA\jboss-5.1.0.GA\bin".

C.1.3 Configuring JAVA

The java SDK version 1.6 is installed and downloaded from ... The JAVA_HOME is set to "C:\Program Files\Java\jdk1.6.0_18". Verify that the BIN directory is set in the PATH. To test that Java is installed properly you can test by java –version in command prompt.

C.1.4 Configuring ANT

Apache ant is used to deploy various mobicents project. Similar to the setup above we configure ANT_HOME. In our installation ant is extracted to "c:\ant". So we configure ANT_HOME to "c:\ant" and we add the BIN directory to the path i.e "c:\ant\bin".

C.1.5 Configuring Maven

In order to build from source Maven needs to be installed and configured. M2_Maven variable is set to "C:\servers\apache-maven-2.2.1" and additionally we add the bin directory to "C:\servers\apache-maven-2.2.1\bin".

C.2 Adding MoBill's Jar File

Navigate to the %JBOSS_HOME/server/default/lib. JBOSS_HOME is defined in Appendix C.1.2 . Place the jar file in this directory.

Appendix D

Mobicents examples

We discuss the installation of the different examples.

D.1 Deploying Google Talk Bot Example

In this section we explain how to deploy Google Talk Bot Example.

D.1.1 Prerequisites

The GoogleTalkBotSbb will require a user account and password to an existing Gtalk account.

The Mobicents XMPP resource adapter is used by Google Talk Bot example. XMPP resource adapter needs to be deployed.

D.1.2 Installing Google Talk Bot Example

To install the Google Talk Bot service, navigate to the installation directory of the system. Use the following command to load the example into the mobicents server. “mvn clean install –o”.

D.1.3 Executing the Example

To start the mobicents server, navigate to the bin directory or type "cd %JBOSS_HOME%/bin". To start the server use the following command "run.bat –b <IP address>".

D.1.4 Google Talk Client

Download the Google Talk client from: http://www.google.com/talk/

The user will be required to have an account.

D.2 Deploying the SIPB2BUA Example

In this section we explain the deployment process for the SIPB2BUA example.

D.2.1 Prerequisites

The example makes use of a JAIN SIP resource adapter. Therefore the SIP11 resource adapter needs to be deployed. To deploy the SIPP11 resource adapter, navigate to the SIP11 RA directory. Deploy the resource adapter by with ant by using the following command "ant deploy". This will load the sipp11 RA to mobicents server.

D.2.2 Installing B2BUA Service

Similar to appendix D.1.2.

D.2.3 Executing the Example

Similar to appendix D.1.3.

D.2.4 Configuring SIP Phones

Two SIP phones are used; desktop soft phone(X-lite) and hard phone.

D.2.4.1 Counter Path X-lite

The phone can be downloaded from: http://www.counterpath.com/

The domain field needs to be set to the IP address that the mobicents server is running on. Additionally the Mode field in the Presence tab is set to Presence Agent.

D.2.4.2 SIP phone

A physical SIP phone is configured with an IP address.

Appendix E

Configuring Ericsson Diameter Emulator

Download the Charging SDK from: http://devtools.ericsson.com/node/18

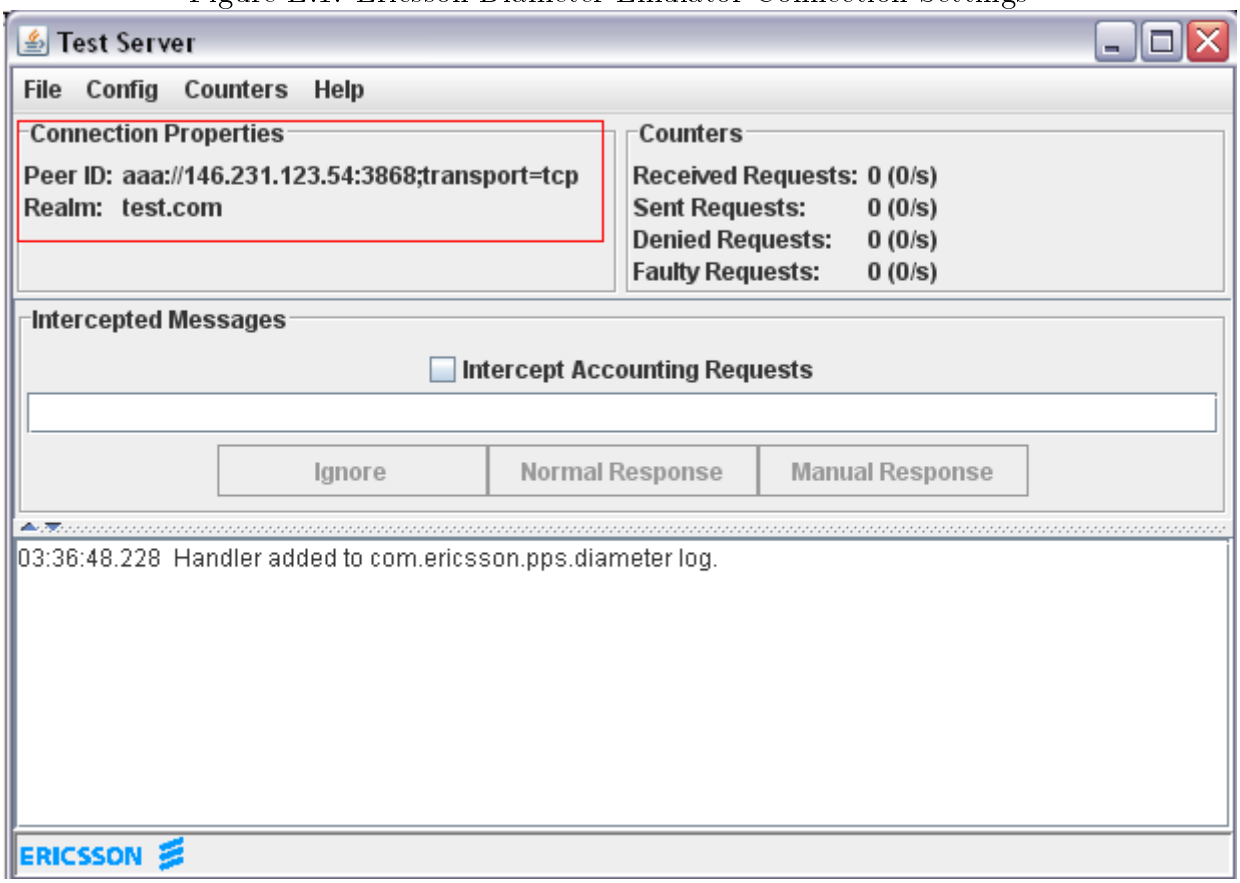
The Charging SDK 2 has been installed and unzipped to "Ericsson Server\Ericsson" in our machine. We open the command prompt and navigate to this installation directory. The Ericsson Diameter Emulator requires the java runtime environment to be installed (JRE 1.5 or higher). After this we execute the application with the following command "java -jar SCAPv2TestServer.jar".

In order to establish a connection between the client application and the Ericsson Diameter Emulator, the connection properties needs to be configured. We need to configure both the URI and the Realm.

In the options menu item we configure the peer ID to "aaa://146.231.123.54:3868;transport=tcp" and Realm to "test.com".

Figure E.1 shows connection settings.

Figure E.1: Ericsson Diameter Emulator Connection Settings



Appendix F

DVD Contents

F.1 Thesis\

The folder contains the final research report.

F.2 JavaDoc_MoBill_BillingComponent\

The folder contains Java Documentation for MoBill. Additionally it shows details of the package with the different classes and methods. We named the billing component MoBill because the naming happened later in the history of this project, portions of the code still use 'billing component'. So, while reading through the code and documentation, keep in mind that MoBill is referred to as 'Billing Component'.

F.3 JavaDoc_EricssonChargingAPI\

Folder contains the Ericsson Charging API java documentation, listing all the different packages and classes in the API.

F.4 Mobicents_Examples\

All the used examples and source code are placed here.

F.4.1 GoogleTalkBot_Example\

Google Talk Bot example used to demonstrate event-based charging.

F.4.2 SIPB2BUA_example\

SIPB2BUA example used to demonstrate time-based charging.

F.4.3 VoD_Example\

The folder contains video on demand example used to demonstrate reward-based charging.

F.5 Project_Proposal\

Folder contains a project proposal document for the research project.

F.6 Software\

Contains various software packages used within the research project.

F.7 Poster\

Folder contains a project poster.