Modeling of Radio Access Application Protocols for Mobile Network Traffic Generation

Suliman Kahled Albasheir

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Electrical & Computer Engineering)

at

Concordia University

Montréal, Québec, Canada

December 2008

Your file Votre référence
ISBN: 978-0-494-63199-7
Our file Notre référence
ISBN: 978-0-494-63199-7

ABSTRACT

Modeling of Radio Access Application Protocols for Mobile Network Traffic Generation

Suliman Kahled Albasheir

Telecommunication applications have become some of the most important aspects of our daily life, especially with the Internet-based applications that are available even on cell phones. One of the challenges faced by telecom companies is to provide robust and powerful servers that are capable to handle the great increase of the number of subscribers and to accomplish the heavy Internet-based applications that generate a tremendous traffic load. Telecom companies evaluate their products' performance before releasing them to the market by applying a large amount of *generated traffic* to the telecom servers in order to measure their capability under traffic load. To do this, powerful solutions are needed, which generate traffic by modeling different telecom protocols. In this thesis, we propose a new technique of modeling a traffic generator solution to load the Mobile Switching Center (MSC) for the Universal Mobile Telecommunications System (UMTS). This traffic generator is modeled to load the MSC through various mobile call scenarios such as location update, mobile call originating, mobile call terminating, and call clearing. Based on that, we model the Radio Access Network Application Part protocol' procedures to generate the radio access messages that carry and handle the mobile messages. These mobile messages will be represented through the Mobility Management and the Call Control protocols' models. To achieve the above goals, we utilize the UML Use Case Model to describe the functional behaviors of the traffic generator, also we present the UML Analysis Model that provides the logical implementation of the functional behaviors of the proposed traffic generator.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my great thanks to the Almighty God, Who gave me the strength and patience to complete this work.

I would like to extend my sincerest gratitude for my supervisor Dr. Sofiène Tahar, who has been a constant source of thoughtful guidance in pursuing this work. Because of his input, advice, and challenge, I have matured as a researcher and as a graduate student. I am very thankful for several people at Ericsson Montreal Canada, Samir Douik and Teresa Marchut-Wierzbica for giving me an opportunity to do my Master's project at Ericsson, Claude Gauthier, Jean Roussel Personna, Martin Kirouac, and America Arredondo Garza for their guidance, mentoring, and encouragement during my project. Also, many thanks to Marin Pin, Gennady Bayder, and Martin Robinson for their technical support. I would also like to acknowledge my thesis committee: Dr. Abdelwahab Hamou-Lhadj and Dr. Chadi Assi for their valuable feedback on the thesis.

I would also like to take this opportunity to express my sincere thanks to my colleagues in the Hardware Verification Group (HVG) of Concordia University for their motivation, constructive suggestions, and helpful comments. In particular, I would like to thank Naeem Abbasi for his time and valuable help during my thesis writing. Special thanks go also to Dr. Ali Shatnawi, from Jordan University of Science and Technology, for introducing me to the ECE Department at Concordia University and to Dr. Sofiène Tahar.

I would like to thank all my friends and relatives in Montreal, Jordan, and everywhere. Also, I am very grateful for my sisters, brothers, aunt, and grandparents for the support and happiness they always provide me with. Finally, my sincere thanks and deepest appreciation go out to my parents, Fatima and Khaled for their affection, love, support, encouragement, and prayers to success in my missions.

This thesis is dedicated to

My Mother Fatima Itwaiq

and

My Father Khaled Albasheir

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AAL5	ATM Adaptation Layer Type 5
AP	Access Point
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
BHCA	Busy Hour Call Attempts
CC	Call Control
CM	Connection Management
CN	Core Network
CPP	Ericsson Connectivity Packet Platform
CS	Circuit Switch
EBSDL	Entity Behavioral Specification and Description Language
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HLR	Home Location Register
ID	Identifier
IE	Information Element
IMSI	International Mobile Subscriber Identity
IP	Internet Protocol
Iu	Interface links the RNC to either a MSC or a SGSN
IuCS	Circuit Switched Interface links the RNC to a MSC
LAI	Location Area Identifier
LAN	Local Area Network
MGw	Media Gateway
MGwSim	Media Gateway Simulator
MM	Mobility Management

MOC	Mobile Originating Call
MS	Mobile Station
MSC	Mobile Switching Center
MTC	Mobile Terminating Call
NAS	Non Access Stratum
NNI	Network Node Interface
PDU	Protocol data unit
QoS	Quality of Service
RAB	Radio Access Bearer
RANAP	Radio Access Network Application Part
RNC	Radio Network Controller
RRC	Radio Resource Control
RSD	Rational Systems Developer
SAI	Service Area Identifier
SCCI	SCCP Interface
SCCP	Signalling Connection Control Part
SccpApfi	SCCP Access Point Facade Interface
SDL	Specification and Description Language
SGSN	Service GPRS Support Node
SNMP	Simple Network Management Protocol
SSCF	Service Specific Coordination Function
SSCOP	Service Specific Connection Oriented Protocol
TMSI	Temporary Mobile Subscriber Identity
UE	User Equipment
UMTS	Universal Mobile Telecommunications System
UTRAN	UMTS Terrestrial Radio Access Network
WCDMA	Wideband CDMA Code Division Multiple Access

Chapter 1

Introduction

1.1 Motivation

The Telecommunication industry has been growing tremendously during the last decades. The International Telecommunication Union (ITU) reported that worldwide mobile cellular subscribers are likely to reach the 4 billion mark before the end of this year with an estimated mobile penetration rate reaches 61% as shown in Figure 1.1 [21]. Also, it is reported in [22] that the global Telecommunication revenue will reach \$2.0 trillion USD by the end of 2008, an increase of 7.6% over revenues in 2007.

Based on the 61% estimated mobile penetration rate, this means there is a possibility to have more and more subscribers into the telecom networks. On the other hand, with the new next generation networking technology, more and more internet-protocol-based applications are introduced or planed to be implemented in the future.

Consequently, all these parameters require powerful telecom servers that are capable to handle the tremendous increase of the subscribers' number and to carry out the heavy next generation applications that generate huge traffic loads. It is extremely important for the telecom companies to have powerful servers that capable

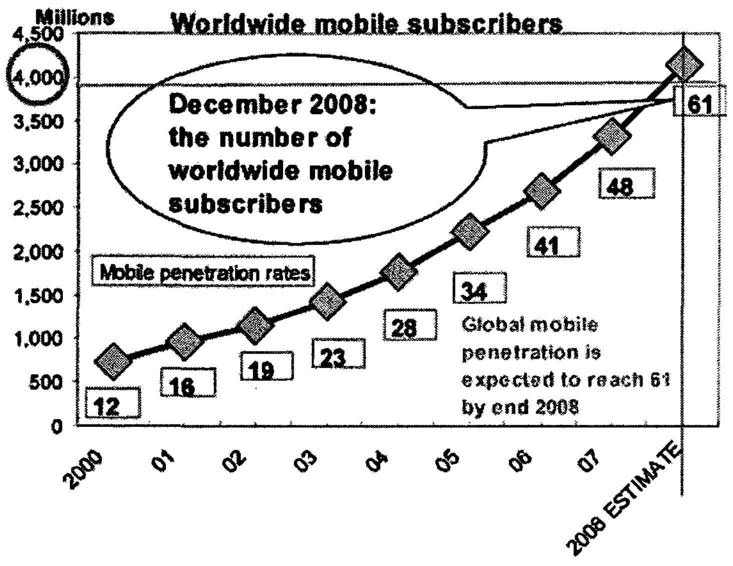


Figure 1.1: Worldwide Mobile Subscription Growth [21]

to carry out the huge telecom traffic loads, to achieve that, testing engineers in telecom companies apply huge amount of simulated traffic to their telecom devices to measure their capability and behavior under the pressure; this kind of test is called *Load Testing.* Based on the load testing measurements, design engineers can modify the design of the system to have more robust systems under the traffic load.

To do load testing, it is very important to have a powerful solution that simulates telecom protocols, initiates call scenarios, interacts with the telecom network, as well as generates traffic messages towards the corresponding device under test; we call this operation *Traffic Generation.*

The traffic generation can be achieved through several techniques such as; software solution, hardware solution, or hybrid solution. The software solution is very slow and the generated traffic is neither accurate nor realistic, also using a software solution is not easy to interface with the real system. By using a hardware solution to generate traffic with real components, we obtain a very fast, realistic, robust, but costly solution. Using a hybrid solution that contains hardware and software together, we may obtain a trade-off between speed, realistic traffic, robust system, and cost.

In this thesis, we introduce a technique of modeling a traffic generator to do load testing for the Mobile Switching Center that measures the capability and performance of the Mobile Switching Center under traffic pressure and does not actually verify or check the design of this Center. The proposed traffic generator provides models for the Radio Access Network Application Part, the Mobility Management, and the Call Control protocols for the *Circuit Switched* network.

Apply huge amount of traffic to the telecom devices to measure their capability under traffic pressure.

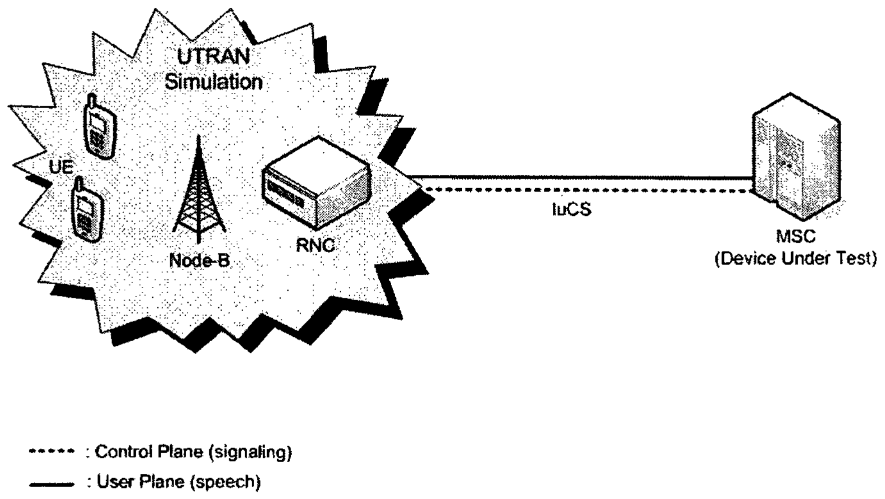


Figure 1.2: Traffic Generation Environment for the Mobile Switching Center

Figure 1.2 shows the traffic generation environment for loading the Mobile Switching Center (MSC). Typically, to generate traffic in order to load the MSC, many solutions model the whole Universal Mobile Telecommunications System Terrestrial Radio Access Network (UTRAN) components' protocols, these components are; the User Equipment (UE), radio base transceiver station (Node-B), and Radio

Network Controller (RNC). Obviously, it is not cost effective to simulate all these components and their protocols in order to load the MSC server.

Our proposed modeling technique simulates only the messages on the interface between the Radio Network Controller (RNC) and the MSC server; these messages are the Radio Access Network Application Part (RANAP) message, the Mobility Management message, and the Call Control message portions. The interface between the RNC and the MSC called IuCS, this interface is used for circuit switched data transfer.

Furthermore, our proposed design for traffic generation is modeled based on an Ericsson proprietary platform called CPP [5]. The Ericsson Connectivity Packet Platform (CPP) is a very powerful node used for emulation and protocol transportation in many applications within Ericsson globally, it provides a high rate of calls generation (1000 calls/sec). It is expected that our proposed design economizes in using real telecom equipments; since our design provides models for the radio access network messages, so it does not require real component such as the Radio Network Controller to generate the traffic.

1.2 Traffic Generation Design

One of the components of the Ericsson's WCDMA/GSM Core Network is the MSC Server, which is an entity that controls the setup and the release of the communication connections in the network. The actual connections are handled by the Media Gateway (MGw), which are driven by the MSC server using the Gateway Control Protocol (GCP) protocol.

To simulate an environment around the MSC for load testing first, we have to simulate the Media Gateway (MGw) that handles the GCP protocol. The MGw Simulation (MGwSim) is a tool developed by Ericsson, it interacts with the MSC and generates GCP protocol traffic, this tool is used for the MSC load testing [16].

Second, we have to generate the messaging traffic which simulates the RNC and other components of the UTRAN (Node-B and UE). To generate this traffic, testing engineers at Ericsson use either MGTS [27] or 3GSim [30] traffic generation tools along with the MGwSim to generate messaging traffic that properly load the MSC server.

In this thesis, we propose to model all messages between the MSC and RNC on the IuCS interface; this means no models will be provided to simulate the messages between the Node-B and the UE. All components in the proposed *Traffic Generator* will be modeled based on the CPP platform. Since the MGwSim is built on the CPP platform, this gives a capability to co-locate the MGwSim and our traffic generator on one CPP node to apply traffic load towards the MSC. The CPP platform version that is used for the current design is CPP 7.0 [5].

In real application, many call scenarios load the MSC server such as; mobile location update, mobile call originating, mobile call terminating, call waiting and forwarding, roaming call scenario, and others; but the mobile originating, mobile terminating, and location update call scenario can be responsible for most of the messages that load the MSC. Based on this, we consider those three call scenarios to generate the load traffic. In addition, most of the testing activities done by the industry for the MSC load actually concentrate on those three call scenarios.

In the traffic generator design, we concentrate on the modeling of the control plane messaging (signaling); no models will be provided for the user plane messaging (speech). An external tool for user plane traffic generation will be used to provide the speech load. In this thesis, we propose a model for the *RANAP Simulator Traffic Generator*, hereafter referred to as the *RanapSim*.

Figure 1.3 shows the proposed traffic generator architecture where it illustrates the main components, the CPP platform with its provided protocols, as well as the MSC server which is the device under test. In this figure, there are two different colors for the architecture components that can be explained as follow;

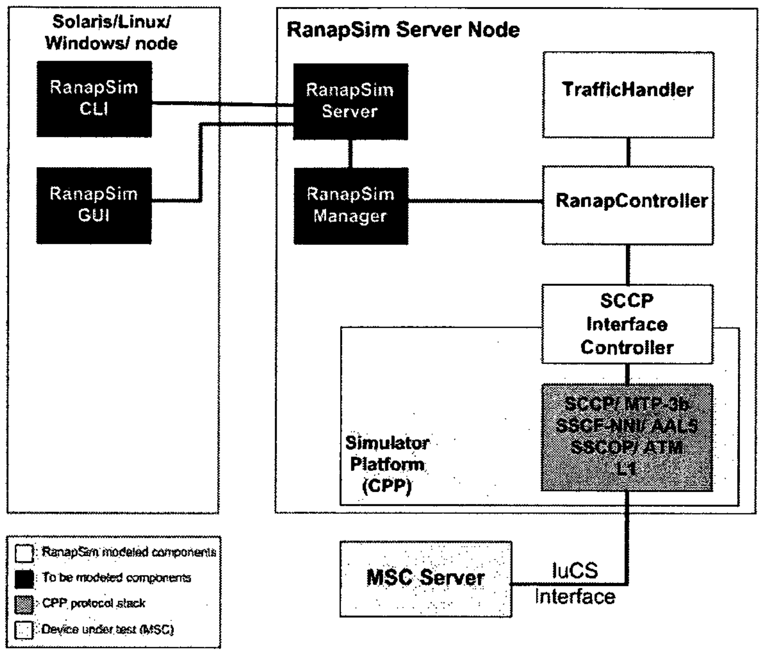

Solaris/Linux/ Windows/ node
RanapSim CLI
RanapSim GUI
RanapSim Server Node
RanapSim Server
RanapSim Manager
TrafficHandler
RanapController
SCCP Interface Controller
SCCP/ MTP-3b SSCF-NNI/ AAL5 SSCOP/ ATM L1
Simulator Platform (CPP)
MSC Server
IuCS Interface
: RanapSim modeled components
: To be modeled components
: CPP protocol stack
: Device under test (MSC)

Figure 1.3: Traffic Generator Architecture

The *white* color indicates that these components will be modeled in this thesis and they represent the core of our proposed traffic generator. These components are:

- Traffic Handler: The traffic handler is the main component for generating the mobile call scenarios' traffic by modeling all required protocols.
- RANAP Controller: This component controls and handles all interactions between the RanapSim components.
- SCCP Interface Controller: This component provides a connectivity functionality between the RanapSim traffic generator and the CPP platform in order to communicate with the MSC server.

These components will be explained in details in Chapters 2, 3, and 4.

The *black* color indicates that these components are assumed to be modeled later on. These components are:

- RanapSim Manager: The Manager is the central entity that controls the traffic generator simulation. Since this component is not modeled, we deal with it as an external system with clear specifications.
- RanapSim Server: The Server is responsible for forwarding the requests to the Manager, and transmitting the results back to the user.
- RanapSim GUI: The Graphical User Interface (GUI) provides a graphical interface to the simulator node for the user.
- RanapSim CLI: The Command Line Interface (CLI) provides a text-based interface to the simulator node for the user.

1.3 Methodology

In the following, we describe the proposed methodology that we adopted to model the traffic generator for the MSC load testing in the UMTS network. Figure 1.4 shows the UMTS network architecture and highlights the IuCS Interface that connects the RNC and the MSC, the transferred message over this interface contains many protocols' headers which are; the MM, CC, RANAP, SCCP, MTP-3b, SSCF-NNI, AAL5, SSCOP, and ATM [12].

Our methodology is to model only the Mobility Management (MM) and the Call Control (CC) protocols (generated by the UE) and model the RANAP protocol (generated by the RNC). The other protocols -that we have specified in the IuCS- are not needed to be modeled, by taking advantage of using the CPP platform. CPP is the Ericsson Connectivity Packet Platform, which is used for execution and transport purposes; the execution part provides a support for software application execution,

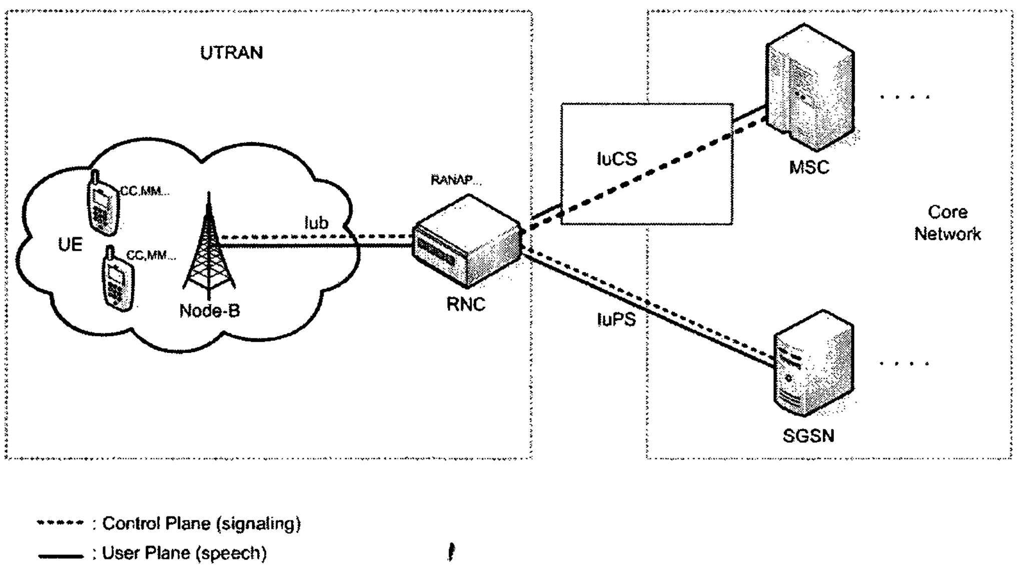


Figure 1.4: The UMTS Network Architecture

and the transport part provides connectivity functionality for several communication protocols [5], where it provides a connectivity function for the physical layer protocol, ATM protocol, up to the SCCP protocol.

Figure 1.5 shows the protocol stack provided by the CPP, the modeled protocols for the RanapSim, and the MSC protocol stack. This figure illustrates that RanapSim provides models for Mobility Management, and Call Control protocols which are responsible for representing the UE signalling messaging, these protocols should communicate with their corresponding protocols at the MSC server.

The RANAP protocol in the RanapSim represents the RNC signalling messages where the UE messages portions are carried as well. From Figure 1.5, it is obvious that RanapSim communicates with the CPP platform through the SCCP protocol that controls the signalling connections, where CPP platform provides an Application Programming Interface (API) based on the Unified Modeling Language (UML) to facilitate the communication with it [5]. By modeling an interface through the SCCP protocol, our model is able to communicate with and route messages to

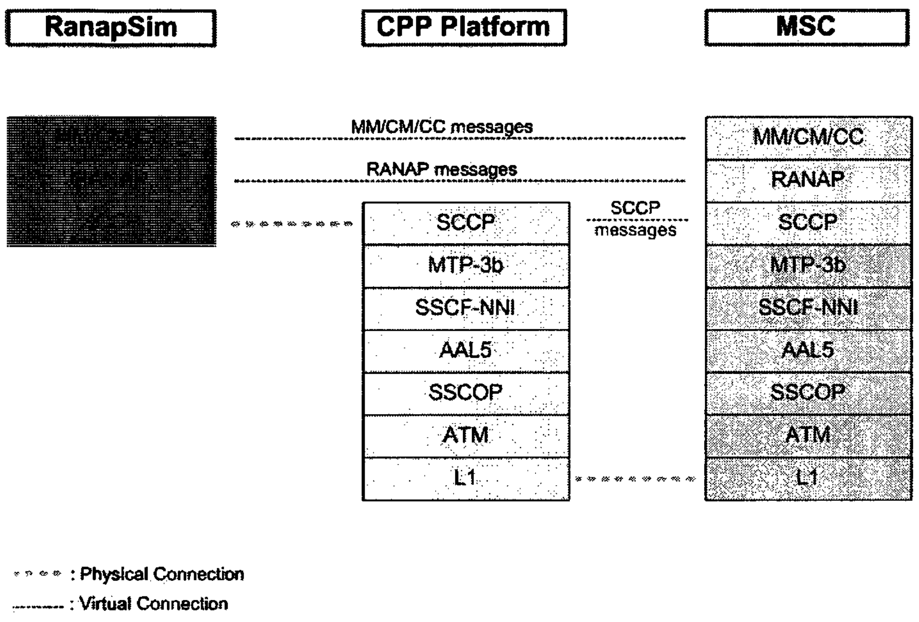


Figure 1.5: Protocol Stack

the CPP platform.

By handling the RANAP, MM, and CC messages to the CPP platform through the SCCP interface, the CPP is responsible for building the rest of the message by providing the headers of the SCCP, MTP-3b, SSCF-NNI, AAL5, SSCOP,ATM, and the physical layer protocols; the whole message will be forwarded by the CPP through the physical connection to the MSC server. Once the MSC receives the message from the CPP, it will decode and process the whole message contents including the generated RanapSim message portion. So, by utilizing the CPP transport functionality, we are exempted to model the rest of the protocol stack and this makes our model more efficient. Using this technique, we are able to communicate and load the MSC by the RANAP, the MM, and the CC messages without modeling the whole UTRAN. From what we have explained, our model should provide three main functionalities which are;

- Generate signalling traffic (RANAP, MM, and CC modeling).

- Interact and interface with the CPP platform (SCCP interfacing).
- Control the interacting components.

Several Modeling languages can be used to model the RanapSim Traffic Generator such as; the Unified Modeling Language (UML) [4] and the Specification Description Language (SDL) [41]; since our model is needed to be implemented as a tool using programming languages, so we consider the UML for modeling the traffic generator. Furthermore, we use the Rational Systems Developer (RSD) from IBM [3] to build the UML Use-Case Model and the UML Analysis Model which visually show the functional specifications, describe the structure of the system, and explain the interfacing between the system and its surroundings. The RSD tool is chosen because it is one of the most popular tools for designing UML models that is used in industry including Ericsson.

1.4 Related Work

In this section we present the related work in the area of protocol modeling and traffic generation for telecommunication networks using various techniques and methodologies. We will focus on traffic generation to load different telecommunication servers and components such as the MSC server. Also, we will show the work for telecommunication protocols modeling using different languages and techniques for various telecommunication standards; such as Universal Mobile Telecommunications System (UMTS) and Global System for Mobile communications (GSM). Furthermore, we will show the importance of the traffic generation for protocol performance testing and servers load testing in the telecommunication systems.

Since telecommunication networks and applications became more and more complex, there is a need for an efficient design for traffic generation that supports performance evaluation and load testing. For instance, the work in [32] proposes a model for a traffic generation tool; this generator has been used to evaluate the performance of some applications between GPRS and WLAN networks. This work generates traffic by having traffic profiles produced by user-level software modules; these traffic profiles may contain parametric traffic description such as; time distributions and packet size information. The traffic will be generated by the traffic engine by recognizing the parameterized information of the traffic profiles. This work is an extension of a previous work described in [33] that presents an architecture for a traffic generator capable of generating Asynchronous Transfer Mode (ATM) traffic according to stochastic models. Although this model generates ATM traffic, the main elements of the model can be used for generating traffic for other networking technologies. In the work presented in this thesis, we propose a design for traffic generator by modeling the RANAP, Mobility Management, and Call Control protocols without using traffic profiles information.

In [36], the author provides an overview of computer-based simulation for communications networks modeling, as well as some important related modeling issues. Also, the work presents a traffic modeling for the ATM networks to test multimedia and video services in the telecommunication networks. The Monte Carlo computer simulation has been used for characterizing network resources needed for traffic generation. The design we propose is not pure simulation since it is built based on a hardware platform.

Another work has been presented in [37] for traffic generation modeling for ad-hoc communication networks testing. It provides a Mobility Data Generator that generates packets of mobile transceivers that usually communicate via radio transmission. Along the same line of thought, the behavior of mobile transceivers has been modeled in [42] to test and evaluate Mobile Ad-hoc Networks. In contrast, the work in this thesis models the behavior of the mobiles call scenarios to generate the required traffic to load the MSC server.

The work in [35] presents a multi- application traffic generator that can be used to generate packets over a Local Area Network (LAN). The generator is capable to test other types of communication systems; this can be done by changing the main controller of the generator. In [38], the authors describe a software that simulates mobile call scenarios such as mobile originated call and mobile terminated call for the UMTS telecommunications system [40]. The provided simulator is capable to generate RANAP messages on the interface between the MSC and the RNC. Using the OMNeT++ simulator [39], some of the UMTS components have been simulated; such as mobile equipment, Node B, RNC, Media Gateway, MSC Server, and others. The generated messages by this work are extracted from some trace files and are restructured to form new messages, but still these messages not really generated as are those coming out from real network components. Also, this work describes the functionality of the simulated communication system using SDL (Specification and Description Language) [41]. The mobile originated call and the mobile terminated call scenarios in the UMTS system have been described and modeled using a sort of sequence diagrams that show all required RANAP, Mobility Management, and Call Control messages to simulate the mobile call scenarios. However, this software still cannot be used for testing real equipment, but in order to utilize it for real network load testing, it requires having connectivity functionalities to the real UMTS communication servers, for instance the MSC server. In this thesis, we provide models for the RANAP, Mobility Management, and Call Control protocols to emulate mobile originating call, mobile terminating call, and location update mobile call scenarios for the UMTS system in order to load the MSC server. Our model is capable to interact with the MSC server, since it is built based on the CPP platform which provides a connectivity functionality with the MSC.

In [43] [44], the authors propose a modeling technique for location operations in a UMTS Network. These operations belong to both core network and access network under the Mobility Management protocol. For location operations modeling, the work of [45] [46] provides functional blocks that represent various components in the UMTS, such as UE, Node B, RNC, MSC, and others to simulate the location operations. SDL [41] and MSC (Message Sequence Charts) [47] have been used for modeling in this work. The provided model still can be improved by implementing the designed test outputs in the presence of physical implementations of network elements. In contrast, our models are capable to communicate with the MSC through the CPP platform.

The work in [19], presents a simulation model for the UTRAN, the simulated model has been used to check the UTRAN Quality of Service (QoS) requirements of the ATM links that connect the Node-B and the RNC. Along the same line of thought, the simulation has been used in [17] [18] to model and simulate the UTRAN, which contains UE, Node-B, RNC, and the interfaces between them. An analytical approach has been used to characterize the traffic and to evaluate the UTRAN performance. In our work, we do not simulate the UTRAN to test its performance, we model some of the UTRAN protocol generate real traffic on the IuCS interface.

In [20], the author concentrates on modeling the Mobility Management traffic load and illustrates the impact of this traffic load on the MSC server. The author describes preliminary information of the Mobility Management protocol parameters that has an impact of load the MSC. This load represents a signaling traffic related to the services provided by Mobility Management protocol, such as location update, paging procedure, authentication, and ciphering process. Our work models the Mobility Management protocol procedures and also models the Call Control protocol procedures that build the mobile call scenarios.

In [23], the work provides a design for a protocol controller to handle the communication functions at the Data Link layer between the MSC and the Base Station Controller (BSC) in the GSM mobile networks. The Message Transfer Part (MTP) protocol is responsible for these communication functions at the Data Link layer. Our work concentrates on the protocol modeling between the RNC and the MSC for the UMTS networks.

The work in [48] indicates that the location update mobiles' calls can be responsible of about 80% of the random accesses to the radio channel in the GSM networks. This fact indicates the significant impact of the traffic load on the MSC due to the Mobility Management signaling messages. The work in [34] proposes new techniques of test generation for communication standards, it indicates that communication software requires a lot from load testing to get reliable systems. Telecommunication companies are spending a lot of money for testing activity (up to 50% of overall development costs). This indicates the necessity of providing and modeling traffic generators for various communication standards to support systems testing.

Several languages and techniques have been used to model the telecommunication standards protocols. As we have mentioned before, the work in [38] [45] [46] [49] have used SDL [41] and MSCs [47] in modeling. The modeling part in [24] uses Entity Behavioral Specification and Description Language (EBSDL) to model the SCCP protocol. The Simple Network Management Protocol (SNMP) has been modeled in [25] using the Unified Modeling Language (UML). The Use-Case Model and the Analysis Model have been described for the SNMP protocol modeling. Another work using UML in [26] provides an approach for network traffic modeling, which can be used in load testing to discover the faults of a system under test, Sequence Diagrams and Class Diagrams have been modeled to describe the logical implementation of the functional requirements. In this thesis, we make use of UML to model the traffic generator protocols and call scenarios where we build the Use Case Model, define the analysis classes, illustrate the sequence diagrams, and build the class diagrams.

Commercial Tools

Many companies have been working on designing and developing tools for traffic generation, which can be used for telecommunication systems load testing and performance evaluation. Those tools provide various solutions for traffic modeling of most of the telecommunication systems protocols using different methodologies and techniques.

The Message Generator Traffic Simulator (MGTS) testing tool [27] is a solution provided by CATAPULT Communications. Another solution is developed by the same company, which is the Digital Communications Test System (DCT2000) [28]. These tools provide a variety of models for protocol simulation and traffic generation. MGTS and DCT2000 provide a simulation for all nodes and their interfaces protocols in the UTRAN (UE, Node-B, and RNC). The simulated UTRAN is used to generate traffic towards the MSC in order to do the load testing. The RANAP, the Mobility Management, and the Call Control protocols have been modeled in the simulated UTRAN. MGTS and DCT2000 require lots of scripting efforts from the end-users to implement the test scenarios scripts which will represent the call scenarios for the mobile communications. Those solutions are designed to run on a proprietary platforms provided by Catapult.

The Polystar company developed the SOLVER System [29], which is a load test tool for various telecommunication networks. It provides the ability to simulate the UTRAN for GSM/GPRS and UMTS with a number of connected mobile stations providing load to the MSC and other devices in the Core Network. SOLVER does not require lots of scripting efforts to implement the test call scenarios but it is not flexible to have customized test scenarios, also it supports a limited number of call scenarios. A proprietary platform from Polystar is used to run SOLVER to execute the required call scenarios.

Ericsson has developed a Third Generation Simulator (3Gsim) solution [30], as a load generator for traffic simulation in the UMTS networks. 3Gsim is used for the RNC load testing. 3Gsim can simulate some nodes in the UMTS network: the UE, the Node-B, the MSC, the SGSN, as well as the interfaces between them.

By simulating these UMTS nodes, 3Gsim is capable of generating traffic to load the RNC. In order to be able to load the MSC it should use real RNC's to generate traffic towards the MSC server, but this is a costly strategy since it requires real components to generate traffic. 3Gsim provides command-based functionality to create traffic scenarios and to control the simulation. 3Gsim is built on CPP (Ericsson Connectivity Packet Platform) [5].

The UMTS Traffic Model Simulator (UTMS) [31], is another solution developed by Ericsson; it is used to generate traffic for the Media Gateway (MGw) load testing in the UMTS and GSM networks. UTMS simulates the signaling part of the radio access in the RNC and the BSC to load the MGw. UTMS is not designed to load the MSC, but it is possible to do that by using real MGw to simulate complete environment for the MSC. This makes the using of UTMS to load the MSC costly since it uses real component. UTMS is built on CPP (Ericsson Connectivity Packet Platform)[5].

In this thesis, we present a technique to model a traffic generator to load the MSC server in the UMTS networks. In this work, we use UML to model the RANAP, the Mobility Management, and the Call Control protocols on the IuCS Interface. Our technique does not require models for the UE, the Node-B, or the RNC; this makes our approach cost effective and more efficient than other techniques and models that we have highlighted earlier in this section. To load the MSC, those modeled protocols (RANAP, MM, and CC) provide mobile call scenarios; such as location update, mobile call originating, mobile call terminating, and call clearing scenarios. We build our models based on the Ericsson Connectivity Packet Platform (CPP) where the interfacing with this platform will be through the Signalling Connection Control Part (SCCP) protocol.

1.5 Thesis Contribution

The contribution of this thesis is as follows:

- We have provided an efficient technique to generate signaling messages towards the MSC server in order to do the load testing.

- We have modeled the *Radio Access Network Application Part* (RANAP) protocol's procedures in order to generate the radio access network messages in the IuCS Interface.

- We have modeled the mobile radio interface protocols in the UMTS network; one of these protocols are the Mobility Management, which has been utilized to model the mobile location update call scenario. In addition, the Call Control protocol has been modeled to generate signalling messages of the mobile originating call, the mobile terminating call, and the call disconnect scenarios. All of these call scenarios are required to simulate the mobile stations of the network.

- We have built the UML Use Case Model and the Analysis Model for the RanapSim traffic generator design, where we have described the logical implementation of the functional requirements for the proposed design.

- Through the UML models, we have modeled connectivity functionality for the traffic generator design to communicate with the Ericsson Connectivity Packet Platform (CPP) through the SCCP protocol.

1.6 Thesis Outline

This thesis is made up of six chapters. In Chapter 2, we provide a brief overview for the traffic generator architecture, where we introduce the main components of the architecture and we describe the most important functionalities for each component.

In addition, we present the Use Case Model to realize the functional behavior of the proposed architecture; also we define and explain the model's Actors and the Use Cases. In Chapter 3, we provide detailed explanation for the Traffic Handler component and describe deeply the modeling for the RANAP, the mobility management, and the call control protocols. Also, we provide the UML Analysis Model for this component, where we define the analysis classes, build the Use Case realization, and illustrate the class diagrams.

In Chapter 4, we describe the RANAP Controller component which is the main controller for the whole traffic generator design; we realize the main functionalities for this component through describing the analysis classes, the Use Case realization, and the class diagrams. Also, we present the SCCP Interfacing Controller component, where we explain the interfacing functionality with the CPP platform through illustrating the analysis classes, the Use Case realization, and the class diagrams. In Chapter 5, we summarize and conclude the achieved work, and we present some future work hints. Finally, Appendix A contains some UML analysis classes, sequence diagrams, and interfaces which are implemented for the Analysis Model.

Chapter 2

Proposed Architecture

This chapter gives a brief overview of the detailed architecture of the RANAP simulator traffic generator, also it provides the main functional description and responsibilities for all the RanapSim components. In addition, this Chapter interprets the RanapSim functional behavior into UML Use Case Model. Figure 2.1 shows the RanapSim main components which are the Traffic Handler, RANAP Controller, and SCCP Interface Controller, also it illustrates how these component are connected to the MSC server through the CPP platform; the RanapSim Manager is connected to the RANAP Controller component to manage all components' operations.

2.1 RanapSim Components Description

This section explains briefly the functional description of the RanapSim traffic generator that we are going to model, also it highlights the main components' functionalities of the RanapSim.

2.1.1 Traffic Handler

The Traffic Handler performs all traffic generation requests which come from the Manager through the RANAP Controller, these requests show all call scenarios that are supported by this model. This component is responsible for responding to any signaling messages received from the MSC through the RANAP Controller and sending the response back. Figure 2.2 shows the Traffic Handler's entities which represent the main functionalities provided by this Traffic Handler; those entities are the Main Traffic Handler, Location Update Entity, Call Originating Entity, and Call Terminating Entity. These entities are explained briefly as follows:

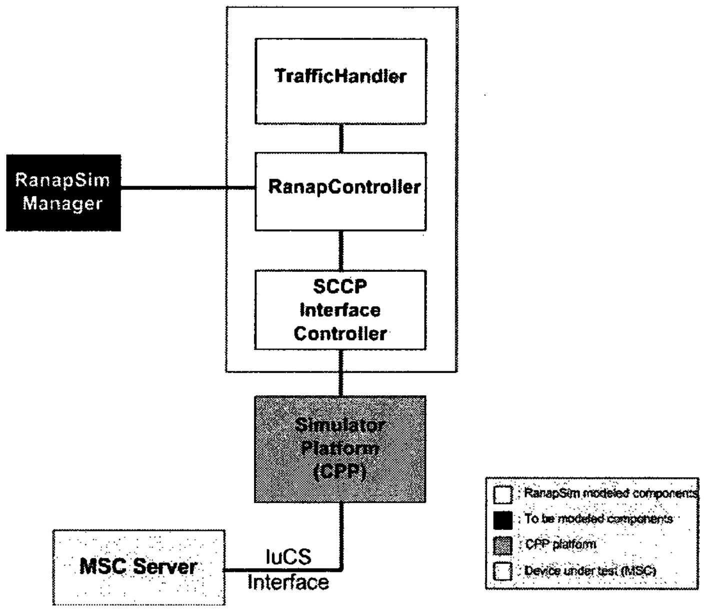


Figure 2.1: RanapSim Main Components

Main Traffic Handler

The Main Traffic Handler is the central entity that controls all signaling messages that are received from the RANAP Controller or from the call scenarios entities. This entity is responsible for forwarding messages to the corresponding call scenario entity. The Main Traffic Handler responsibilities are:

- Handle the Manager call scenario requests which are summarized as follows:

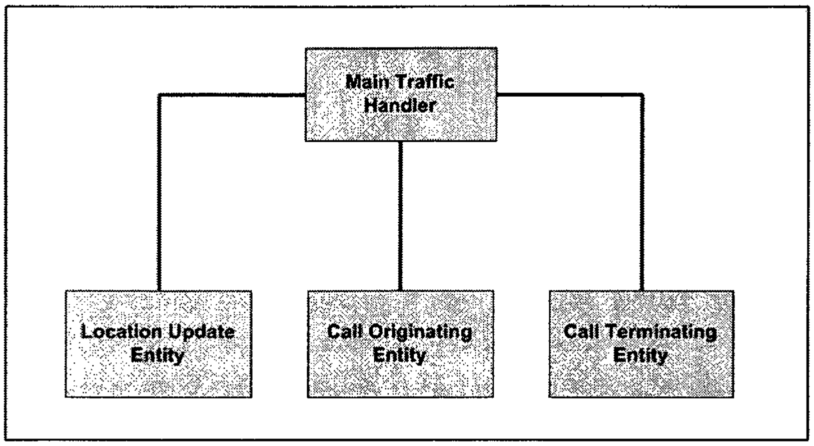


Figure 2.2: Traffic Handler Entities

- Handle the *location update* call scenario request, generate suggested connection identifier and connection state, and forward the call request to the Location Update entity. The generated connection ID along with connection state will be stored at the traffic handler and forwarded with all messages related to the generated call scenario. This ID will become the SCCP connection ID -at the CPP platform- which helps to distinguish between different messages that belong to different calls. More explanations will be given in Chapters 3 and 4.
- Handle the *mobile call originating* call scenario request, generate suggested connection ID, and forward the call request to the Mobile Originating entity. There is no call scenario request for the mobile terminating scenario, because it is triggered at some point in the mobile originating scenario.
- Handle the *call disconnect* requests and forward them to either the Mobile Originating entity or the Mobile Terminating entity, to force any of them to disconnect and clear the call.

- Identify the received messages from the MSC by using the SCCP connection ID which corresponds to a specific call scenario. After identifying the message, the RANAP portion will be extracted from the messages and forwarded to the right destination entity.

  If the SCCP connection ID is unknown or not attached in the message, the Handler will look deeper into the message contents to understand the *message type*. So, if the hexadecimal value of the *message type* equal to "0014", this means that the message is a *PAGING Request* which should be forwarded to the Mobile Terminating entity to start the terminating side of the call.

- Receive signaling messages from the call scenario entities which will be forwarded to the RANAP Controller. If the handler receives a message from the Mobile Terminating entity without having a connection ID, this means that the received message is the response for the *PAGING Request* message. So, a new connection ID and state will be created for the call scenario.

Location Update Entity

The location update procedure is used to update the registration of the actual location area of a mobile station in the network [7], the Location Update entity handles the traffic generation of the location update call scenarios for any request received from the Manager. This entity should model the RANAP and the mobility management (MM) protocols' messages; which is normally carried by the IuCS interface between the RNC and the MSC.

The Location Update entity is responsible for initiating the *Location Update* call scenario by sending the request message to the MSC; which will emulate the mobile request for location update. In addition, this entity authenticates the connection with the MSC and performs other security procedures to secure and cipher the connection with the MSC, all these procedures are usually done by the mobile station which updates its location. After that, the MSC sends a message that contains the International Mobile Subscriber Identity (IMSI), this message will be received by the entity to attach it to the emulated mobile station for identification purposes.

Call Originating Entity

The call originating entity is responsible for establishing the mobile originating call, which triggers the call terminating procedure to establish the mobile terminating call in order to simulate the circuit-switched call between two mobile stations. The Call Originating Entity handles the traffic generation of the originating call scenario by modeling the RANAP, the MM, and the CC protocols' messages [9].

The Call Originating Entity is responsible for the following messages and operations:

- Communicate with the MSC to establish the MM connection through performing the Connection Management (CM) service procedure, authenticating the connection with the MSC, and performing other security procedures to secure and cipher the connection with the MSC.
- Initiate the call setup procedure. Upon establishing the MM connection and receiving the IMSI, this entity shall send a SETUP message to the MSC.
- Receive a CALL PROCEEDING message from the MSC to indicate that the call is being processed, and handle the RAB ASSIGNMENT procedure which is normally initiated by the MSC to assign radio channel for specific mobile station [6].
- Once the Radio Access Bearer (RAB) ASSIGNMENT procedure is completed, the MSC will initiate a *PAGING Request* message to be sent to the Mobile

Terminating Entity -which is another entity from the Traffic Handler-. This message enables the MSC to request the RNC to contact the terminating mobile station. At this point, the call originating entity will wait for the MSC to establish the call with the call terminating entity.

- Receive ALERTING message from the MSC, this indicates that the call terminating entity has setup the call with the MSC and initiated the ALERTING message. After that, the call originating entity receives a CALL CONNECT message from the MSC, and sends back a CONNECT ACKNOWLEDGMENT message to the MSC.

Call Terminating Entity

The call terminating entity is responsible for responding to the originating call and to simulate the mobile terminating call, which will be triggered by the *PAGING Request* message.

The Call Originating entity is responsible for the following messages and operations:

- Communicate with the MSC to perform the PAGING procedure, where the call terminating entity receives the PAGING Request message (RANAP Message) - this message is used to find and contact the simulated terminating mobile in the call terminating entity- and then responds by sending the PAGING Response message (Mobility Management Message) [9].

- Authenticate the connection with the MSC server and perform the Security Mode procedure.

- Receive the Common ID message which contains the International Mobile Subscriber Identity (IMSI), interact with the MSC to complete the setup procedure for the incoming call, and then send the CALL CONFIRMED message to confirm the readiness to receive the call from the call originating entity.

- Complete the RAB ASSIGNMENT procedure with the MSC. The MSC initiates this procedure to assign radio channel for the call terminating entity.
- Send an ALERTING message to the MSC. Upon receiving this message, the MSC sends a corresponding ALERTING message to the call originating entity.
- Send a CALL CONNECT message to the MSC which indicates that the call terminating entity accepts the call. Upon receiving this message, the MSC will send a CALL CONNECT message to the call originating entity.

Upon the CONNECT ACKNOWLEDGMENT message is received from the MSC, the call will be started and the speech will be going on between the call originating and the call terminating entities until a call clearing procedure is initiated by the MSC or by any of the calling parties. The speech load is not provided by this system; still we need a tool to provide the data plane (speech).

2.1.2 RANAP Controller

The RANAP Controller represents the main controller of the system, since it controls all interactions between the traffic generator components and other external components. Also, it represents the connecting point between the RanapSim Manager, the Traffic Handler, and the SCCP interface Controller. The RANAP Controller's responsibilities are:

- Handle the *SCCP Attach* and the *SCCP Detach* requests. These requests are initiated by the Manger to attach/detach the SCCP Access Point Facade Interface (SCCPApfi), this interface handles the SCCP protocol's control plane (signalling) in the CPP platform.
- Handle the Manager call scenarios requests for the call scenarios by forwarding them to the Traffic Handler component in order to generate calls.

- Receive call scenario messages from the Traffic Handler and forward them to the Manager (for tracing and following up purposes). Also, RANAP Controller forwards these messages to the CPP platform which is connected to the MSC.
- Receive the MSC messages (responses/requests) which are sent through the CPP platform, forward them to the Manager (for tracing and following up purposes), as well as forward these messages to the Traffic Handler to complete the call scenarios.

More details about this component are given in Chapter 4.

2.1.3 SCCP Interface Controller

The SCCP Interface Controller is a component that deals with the CPP platform to control the SCCP connections, exchange messages, and interact with the CPP interfaces which are based on the SCCP protocol. This controller is responsible for handling all messages from/to the CPP, and the CPP itself will take the responsibility to interact with the MSC to accomplish our target of applying load to the MSC.

The CPP platform provides various protocols that we can interact with, and for each protocol there are many interfaces to deal with. In our case, we are dealing with the SCCP protocol. The CPP platform provides two interfaces for the SCCP; these interfaces are the control plane interface (SCCPApfi) and the user plane interface (SCCI). The SCCP Interface Controller's responsibilities can be summarized as follows:

- Attach the SCCPApfi interface in order to use the SCCP service. This attach request will be initiated by the Manager.
- Handle the SCCP connections for several call scenarios.
- Transfer data to the CPP platform in order to communicate with the MSC.

- Detach the SCCPApfi interface. No more SCCP services will be provided after detaching this interface.

More details about this component are given in Chapter 4.

2.2 RanapSim Use-Case Model

The UML Use-Case Model describes a system's functional requirements in terms of Use Cases. It consists of all Actors of the system and various Use Cases by which the Actor interacts with the system, thereby describing the total functional behavior of the system. Each Use Case describes the functionality to be built in the proposed system, which can *include* another Use Case's functionality or *extend* another Use Case with its own behavior [4].

The UML Use Case can be defined as a sequence of actions that a system performs to achieve a specific functionality [4]. System's Use Case can be identified by investigating the functions that the user wants from the system, communication information between Actors about changes or events that the system must know about, and information that must be modified or created.

The UML Actor can be defined as anyone or anything that interacts with the system (the application), also it represents human, machine, or another system. System's Actors can be identified by exploring anything uses or maintains the system, the system's external resources, and other systems that interact with our system.

The interactions between the system's Use Cases and Actors can be realized by UML Relationships. A UML relationship is a model element that defines the structure and behavior between other model elements. Some of these relationships are *Association*, *Include*, and *extend* relationships [1].

In this section, we make use of the Use-Case Model to interpret the functional description of the *RanapSim Traffic Generator* into UML modeling elements. The Use-Case Model considers the RanapSim system as a black box; this means that the building blocks within the RanapSim should not be mentioned in the Use-Case Model.

Figure 2.3 shows the Use Case diagram for the *RanapSim Traffic Generator* Use Case Model. It shows that the Traffic Handler functionalities are interpreted into seven Use Cases which are; the *Update Location*, *Originate MS Call*, *Terminate MS Call*, *Disconnect Originating Call*, *Disconnect Terminating Call*, *Distinguish Call Scenarios*, and *Handle Traffic* Use Cases. Also, The Use Case diagram illustrates the RANAP Controller functionalities which are interpreted into the *Attach SCCP*, *Detach SCCP*, *Check SCCP Service*, *Forward Call Scenario Messages*, and *Forward CPP Response Messages* Use Cases.

Furthermore, the Use Case diagram explains all Use Cases that provide the SCCP Interface Controller functionalities, these Use Cases are; the *Control CPP Interfacing*, *Disconnect SCCP Connection*, *Setup SCCPApfi Service*, and *Release SCCPApfi Service* Use Cases. On the other hand, the CPP platform and the RanapSim Manager component are represented as UML Actors in the Use Case diagram.

In next section, we describe in details the Use Case Model that reflects the functional behavior of the *RanapSim*.

2.2.1 Actors

An *Actor* expresses the role of a user (human or external system) interacting with the system. An Actor is not part of the system [1].

CPP

The CPP is an existing platform used for execution and transport with specified interfaces for application design. The execution part consists of support for the design of application hardware and software. The transport part, which can be seen as an internal application on the execution platform, consists of several protocols for communication and signaling. This traffic generator model shall be built based on

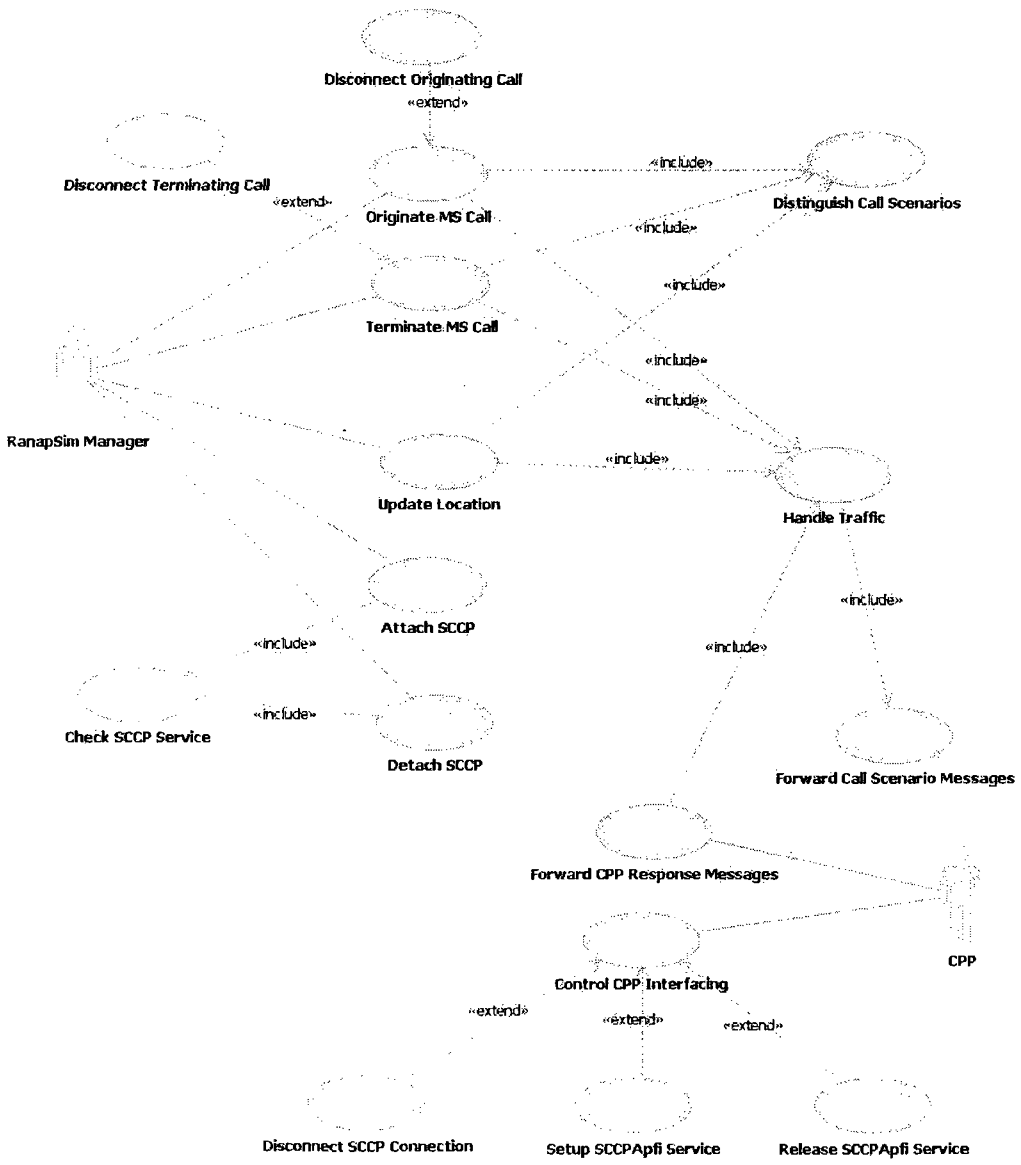


Figure 2.3: RanapSim Use Case Diagram

the CPP platform specifications. Also, the CPP platform provides several protocols and connectivity capabilities from the ATM protocol up to the SCCP protocol.

RanapSim Manager

The RanapSim Manager is an external system that manages the RanapSim resources; also it controls the simulation through command bases. To facilitate the dealing with this component, we assume that the RanapSim Manager is an external system interacting with our main components and it is responsible for initiating the call scenarios requests.

This Actor will be implemented to initiate the following call scenarios requests:

- SCCP Attach (CPP)
- SCCP Detach (CPP)
- Location Update
- Mobile Call Originating
- Originating Call Disconnect
- Terminating Call Disconnect

The Mobile Terminating call scenario will be initiated consequently by initiating the MS Call Originating request.

2.2.2 Use Cases

A Use Case should be used to express user-initiated functionality. All functionalities should be expressed as Use Cases. The *Include* and the *Extend* relationships can be used between Use Cases to express communications, options and possibilities of reuse. These kinds of relations between Use Cases should be visualized in the Use-Case Diagram [1].

Update Location

This Use Case describes how the Location Update call scenario can be generated. The modeled Location Update is specified for the IMSI Attach purpose. Also, this Use Case models all messages which normally go from the RNC to the MSC to emulate the Location Update call scenario. This Use Case interacts with the MSC to complete the Location Update call scenario.

Originate MS Call

This Use Case is started by the RanapSim Manager actor, it describes how the mobile originating call can be generated. It models all messages which normally go from the RNC to the MSC to emulate the originating call scenario. This Use Case responds to the received messages from the MSC, some of the response messages initiate the terminating call scenario at the terminating side to emulate a complete call between the originating and the terminating sides.

Terminate MS Call

This Use Case is started by the RanapSim Manager actor. It describes how a sequence of messages can be generated for establishing a call with the originating side through the MSC server. It describes how the mobile station receives a PAGING request from the MSC and how it responds to it. In addition, this Use Case properly responds to all messages received from the MSC to complete the terminating call scenario, these responses are forwarded to the MSC to connect the originating side to the terminating side of the call.

Disconnect Originating Call

This Use Case is started when the RanapSim Manager actor decides to disconnect and clear the call from the originating side; this Use Case releases all occupied resources for a call between the originating and terminating sides. This Use Case is connected to the *Originate MS Call* Use Case through the *extend* relationship.

Disconnect Terminating Call

This Use Case is started when the RanapSim Manager actor wants to disconnect and clear the call from the terminating side; this Use Case releases all occupied resources for a call between the originating and terminating sides. This Use Case is connected to the *Terminate MS Call* Use Case through the *extend* relationship [4].

Handle Traffic

This Use Case handles and controls the traffic issues of all the call scenarios, it describes a sequence of messages for distinguishing between the various CPP platform response messages in order to route them to the proper call scenario based on the SCCP connection identifier. This Use Case is connected to the *Update Location*, *Originate MS Call*, *Terminate MS Call*, and *Forward CPP Response Messages* Use Cases through the *include* relationships.

Distinguish Call Scenarios

This Use Case is responsible for generating an SCCP connection identifier for a corresponding call scenario, and initiating a call scenario traffic generation which can be identified based on the generated SCCP connection. This Use Case is connected to the *Update Location*, *Originate MS Call*, and *Terminate MS Call* Use Cases through the *include* relationships.

Forward Call Scenario Messages

This Use Case describes how the generated call scenario messages are transferred to the RanapSim Manager actor and the CPP platform actor. This Use Case is connected to the *Handle Traffic* Use Case through the *include* relationship.

Forward CPP Response Messages

This Use Case illustrates how the response messages received from the CPP platform actor can be forwarded to various call scenarios and to the RanapSim Manager actor. This Use Case is connected to the *Handle Traffic* Use Case through the *include* relationship [4].

Attach SCCP

This Use Case is started by the RanapSim Manager to initiates the SCCP service attachment operation; without attaching the service, call scenarios are not allowed to communicate with the CPP platform.

Detach SCCP

This Use Case is started by the RanapSim Manager to initiate the SCCP service detachment operation; call scenarios will not have any access to the CPP platform after detaching this service.

Check SCCP Service

This Use Case makes sure that the SCCP service is attached before any call scenario starts generating messages. This Use Case is included by the *Attach SCCP* and *Detach SCCP* Use Cases through the *include* relationships.

Control CPP Interfacing

This Use Case allows the traffic generator to exchange messages with the CPP platform through the SCCP protocol, it describes how to interact with the CPP platform through the SCCP interface. Furthermore, this Use Case explains how to establish an SCCP connection through the CPP platform for a specific call scenario, and how to forward/receive messages to/from the CPP platform.

Disconnect CPP Connection

This Use Case describes how to disconnect the SCCP connection for a specific call scenario at the CPP platform. This Use Case is connected to the *Control CPP Interfacing* Use Case through the *extend* relationship [4].

Setup SCCPApfi Service

This Use Case describes how the SCCPApfi interface can be setup and attached at the CPP platform actor to utilize the provided service. This Use Case is connected to the *Control CPP Interfacing* Use Case through the *extend* relationship.

Release SCCPApfi Service

This Use Case describes how the SCCPApfi interface can be released and detached at the CPP platform actor; no services will be provided after releasing this interface. This Use Case is connected to the *Control CPP Interfacing* Use Case through the *extend* relationship.

2.3 UML Analysis Model Preliminaries

This section describes the modeling elements of the UML Analysis Model, and it introduces the techniques that will be followed in Chapter 3 and 4 to build the Analysis Model for the *RanapSim Traffic Generator*'s main components.

The Analysis Model describes the structure of the system or application that we are modeling. It describes the logical implementation of the functional requirements that we identified in the Use Case Model. The main purposes of the Analysis Model are to: (1) identify the classes which perform a Use Case's flow of events, (2) distribute the Use Case behavior to those classes through Use Case realizations, (3) identify the responsibilities, attributes and associations of the classes, and (4) build the Class Diagrams of the system. The Analysis Model consists of three phases, which are Analysis Classes, Use Case realization, and Class Diagrams, Figure 2.4 shows the three phases that we follow to accomplish the Analysis Model [4].

The Analysis Classes represent an early conceptual model of the system which contains many classes. The class is a description of a set of objects that share the same attributes, operations, relationships, and semantics, any instance from a class can be called object, and the object is an entity with a well-defined boundary and identity that encapsulates state and behavior.

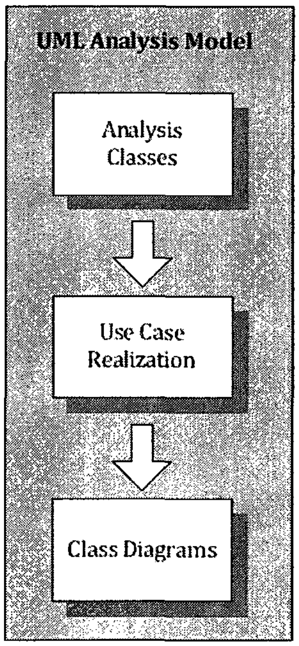


Figure 2.4: UML Analysis Model

The Analysis Classes is the first phase of the Analysis Model, it can be identified based on three perspectives, which are; (1) a class is used to model interaction between the system and its environment; this class represents a *Boundary Class.* The boundary class can be a user-interface class, a device-interface class, or a system-interface class. (2) A class is used to model the control behavior of one or more Use

Cases; this class represents a *Control Class.* (3) A class is used to model information that must be stored; this class represents an *Entity Class* [2].

The second phase of the Analysis Model which is the UML Use Case Realization, this phase can be used to describe the behavior of the Use Case and to identify the responsibilities, attributes and associations of the classes. The class responsibilities can be characterized as the actions that the object can perform, or the knowledge that the object maintains and provides to other objects [1].

To illustrate the Use Case realization, we use *Sequence Diagram* which is a UML diagram that illustrates sequence of messages between objects in an interaction. It consists of a group of objects that are represented by lifelines and messages that objects exchange within the interaction.

In most cases, we use sequence diagrams to illustrate use-case realizations to show how objects interact to perform the behavior of a Use Case. One or more sequence diagrams may illustrate the object interactions which represent a Use Case. A typical organization is to have one sequence diagram for the main flow of events and one Sequence Diagram for each independent sub-flow of the Use Case.

The UML Class Diagrams is the third phase of the Analysis Model which shows a collection of declarative model elements, such as classes, interfaces, and relationships. It is possible to use Class Diagrams to model the objects that make up the system, to display the relationships between the objects, and to describe what services provided by those objects [4].

Class Diagrams can be used to visualize, specify, and document structural features in our models. In addition, class diagrams help to show the class roles and responsibilities that define the behavior of the system, and it illustrates the structure of a model by using attributes, operations, signals, and relationships.

During the Analysis Model, we can create class diagrams to capture and define the structure of classes and to define relationships between classes. UML Relationships provide different types of connectivity between modeling elements, such as

Dependency, Association, Aggregation, Composition, Generalization, and Interface realization relationships [4].

2.4 Summary

In this chapter, we have provided the functional description of the RanapSim Traffic Generator architecture. We have described the main components of the proposed traffic generator namely; the Traffic Handler, the RANAP Controller, and the SCCP Interface Controller, where we have illustrated the main functionalities for each component. The defined functional description of the RanapSim' components have been interpreted into UML Use-Case Model, this model highlights the functional behavior of the RanapSim components in terms of UML Use Cases and Actors. The RanapSim's Use Case Model will be used by the Analysis Model to describe the logical implementation of the system. The UML Analysis Model preliminaries have been presented in this chapter; these preliminaries will be followed in Chapter 3 where we are going to introduce the detailed modeling of the Traffic Handler component.

Chapter 3

Traffic Handler

In this chapter, we present a detailed model for the Traffic Handler component which is one of the main components in the traffic generator design. The Traffic Handler is responsible for generating the signaling messages for some call scenarios, such as location update, mobile originating call, and mobile terminating call. To generate these signaling messages, Traffic Handler provides models for the RANAP, the Mobility Management (MM), and the Call Control (CC) protocols, these models build the contents of each message for the call scenarios. In this chapter, we present the UML Analysis Model phases for the Traffic Handler component.

3.1 UML Analysis Classes

In this section, we are going to introduce the analysis classes for Traffic Handler component. In UML, a class represents an object or a set of objects that share a common structure and behavior, the instantiated objects of these classes are used to build the interaction diagrams [4].

3.1.1 Messaging Proxies Classes

Traffic Handler component has three messaging proxies that are responsible for handling all operations related to the protocols' messages, these messages are; RANAP message [6], Mobility Management message, and Call Control message [7]. Each one of the messages has one *control* class that works as a proxy and many *entity* classes that represent the message data elements. In this section, we give detailed description about these messaging proxies' classes and their operations and data elements.

RANAP Message Proxy

The RANAP Message Proxy is represented by the control class which is shown below, this proxy is responsible for receiving, preparing, forwarding, decoding, and sending RANAP messages as well as performing a RANAP procedure that is requested by the incoming message. Several RANAP messages are represented in this model as UML *Entity* classes and controlled by this proxy class; these messages belong to different RANAP procedures that are required to build the signaling traffic for the modeled calls scenarios. These RANAP messages are; Direct Transfer, Initial UE, Common ID, Security Mode Command, Security Mode Complete, RAB Assignment, Paging Request, as well as Iu Release messages [8]. These messages represent the RANAP portion of the generated traffic to load the MSC.

Figure 3.1 shows the UML representation for the *RANAP Message Proxy class*, it illustrates the operations that are provided by the class. Generally, this class's responsibilities are to: (1) receive and prepare request/response RANAP messages, (2) communicate with the Mobility Management Message Proxy to get the MM message part, (3) communicate with the Call Control Message Proxy to get the CC message part, and (4) co-work with other control classes to perform the RANAP procedures; for example *get Direct Transfer Response.*

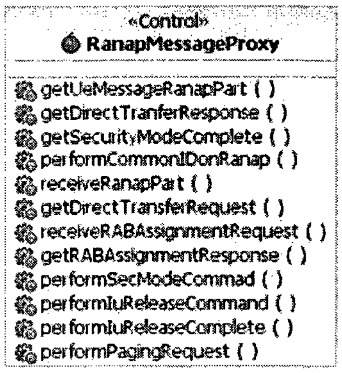

«Control»
RanapMessageProxy
getUeMessageRanapPart ( )
getDirectTranferResponse ( )
getSecurityModeComplete ( )
performCommonIDonRanap ( )
receiveRanapPart ( )
getDirectTransferRequest ( )
receiveRABAssignmentRequest ( )
getRABAssignmentResponse ( )
performSecModeCommad ( )
performIuReleaseCommand ( )
performIuReleaseComplete ( )
performPagingRequest ( )

Figure 3.1: RANAP Message Proxy Class

The *get Direct Transfer Response* operation is invoked usually to perform Direct Transfer procedure and get the response message for it; the purpose of the Direct Transfer procedure is to carry UE - MSC signaling messages over the IuCS Interface. The UE - MSC signaling messages are not interpreted by the RNC, also the UE - MSC signaling messages are transported as a parameter in the Direct Transfer messages [6]. The UE side is represented by the MM message proxy or the CC message proxy classes. More details about this class and RANAP data classes can be found in Appendix A.1.1.

Mobility Management Message Proxy

Mobility Management is one of the main functions of the GSM or UMTS system that allows UEs to communicate with the core network especially the MSC. The main responsibilities of the mobility management are to locate and track where the UEs are, so that any mobile phone services signaling messages can be delivered to them [9].

The Mobility Management Message Proxy is represented by the UML *control* class which is shown in figure 3.2, this proxy gets requests from the RANAP Message Proxy to prepare, send, decode Mobility Management messages, as well

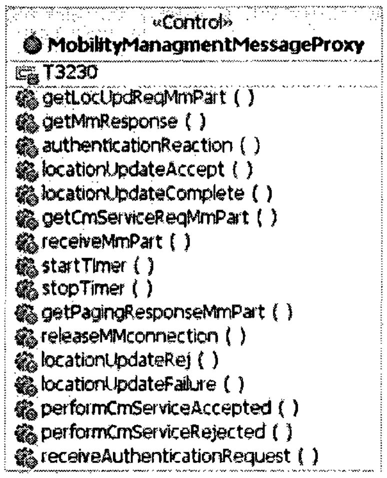

«Control»
MobilityManagmentMessageProxy
T3230
getLocUpdReqMmPart ( )
getMmResponse ( )
authenticationReaction ( )
locationUpdateAccept ( )
locationUpdateComplete ( )
getCmServiceReqMmPart ( )
receiveMmPart ( )
startTimer ( )
stopTimer ( )
getPagingResponseMmPart ( )
releaseMMconnection ( )
locationUpdateRej ( )
locationUpdateFailure ( )
performCmServiceAccepted ( )
performCmServiceRejected ( )
receiveAuthenticationRequest ( )

Figure 3.2: Mobility Management Message Proxy Class

as perform a Mobility Management procedure that is requested by the RANAP Message Proxy. In this model, the MM message proxy represents the UE mobility portion, so it models and replaces the mobility portions of the UE. Various Mobility Management messages are represented as UML *Entity* classes, each one of these classes represent a specific type of mobility management actual message that is related to one of the MM functionalities. In this model, the MM message proxy class represents and models the mobility management portion of the real UE. The MM messages that are provided by this model are; Location Update Request/Accept/Reject, Authentication Request/Response/Reject, Connection Management Service Request/Accept/Reject, as well as Paging Response messages [7]. These messages represent the UE mobility management portion message of the generated signaling traffic to load the MSC. These messages' classes can be controlled and accessed by the MM message proxy class to perform and complete the MM requested services.

Figure 3.2 illustrates the operations and a data member that are provided by the class. In general, this class's responsibilities are to: (1) receive and prepare request/response MM messages, (2) provide mobility management messages to the RANAP Message Proxy class to generate the UE message portion, (3) perform the connection management services which is responsible for generating a MM connection, (4) perform the authentication procedure at the UE side, and (5) perform the location update procedure.

The location update procedure is handled by this class through some operations. Normally, the location updating procedure is used to update the registration of the actual location area of a UE in the network [9], the *receive Location Update Accept* operation indicates that the UE's IMSI (International Mobile Subscriber Identity) is recognized and activated by the MSC. Also, *receive Location Update Reject* operation indicates that the UE's IMSI is not activated by the MSC. More details about this class and MM data classes can be found in Appendix A.1.1.

Call Control Message Proxy

Call Control is one of the GSM or UMTS system which use the mobility management connection to allow UEs to establish and clear calls with the core network especially MSC. The main responsibilities of call control are to allow the UE to originate mobile call, terminate mobile call, and clear mobile call [9].

The Call Control Message Proxy is represented by the *control* class which is shown in Figure 3.3, this proxy gets requests from the RANAP Message Proxy to prepare, send, decode Call Control messages, as well as perform the call establishment or clearing procedures that are requested by the RANAP Message Proxy.

Different Call Control messages are represented as UML Entity classes in this model, each one of these classes represents a specific type of call control actual message that is related to one of the CC functionalities. In this model, the CC message proxy class represents and models the call control portion of the real UE, so it replaces the call control portions of the UE. The CC messages that are provided by this model; Alerting, Setup, Call Proceeding, Call Confirm, Call Connect, Connect Acknowledgment, Disconnect, Release, as well as Release Complete messages [7].

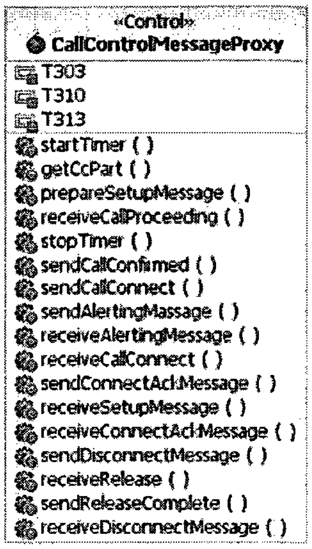


Figure 3.3: Call Control Message Proxy Class

These messages represent the UE call control portion message of the generated signaling traffic to load the MSC. These messages' classes can be controlled and accessed by the CC message proxy class to perform and complete the CC requested functions.

The UML representation of the CC Message Proxy class is shown in Figure 3.3; it demonstrates the operations and data members that are provided by the class. In this model, the CC message proxy class's responsibilities are: (1) receive and prepare request/response call control messages, (2) provide the call establishment and call clearing messages to the *RANAP Message Proxy class* to generate the UE message portion, (3) prepare and send the Setup message to MSC to initiate a mobile call originating establishment, (4) perform both call proceeding and call confirming requests that come from the MSC side, (5) perform the alerting, call connect, and connect acknowledgment procedures at both terminating and originating call sides, and (6) perform the call clearing procedure from for a specific CC entity which represent the call control portion of a UE [10].

In this UML model, we provide the call connect and connect acknowledgment through some operations that are provided by the CC proxy class. The CC proxy class -at the mobile terminating side- indicates the MSC that the call has been accepted at the called entity by invoking the *send Call Connect* operation. The CC proxy class - at the mobile originating side- receives the call connect message from the MSC by invoking the *receive Call Connect* operation, the received message by this operation indicates that the call connection has been established by the MSC [7]. More details about this class and CC data classes can be found in Appendix A.1.1.

3.1.2 Traffic Handler Control Classes

In this section, we give details about the UML control classes for the Traffic Handler component. These control classes are designed to handle and control all signaling messages that are received from the RANAP Controller classes or from the Messaging proxies' classes. These classes contain the call scenarios classes and the RANAP procedures classes. The following are the Traffic Handler's analysis classes with their operations and data elements.

Traffic Handling Controller Class

The *Traffic Handling Controller* class is the main control class for the Traffic Handler Component.

The *Traffic Handling Controller* class controls all interactions between the Traffic Handler component and the RANAP Controller component, where this class handles all signaling messages between the main call scenarios' control classes and the RANAP Controller classes. Figure 3.4 shows the UML representation for this class, it illustrates the operations and data members that are provided by the class.

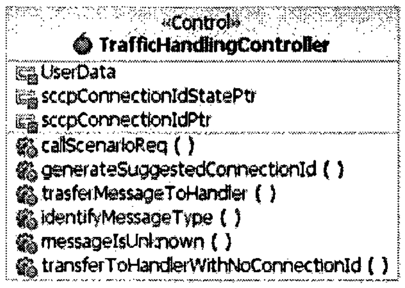

«Control»
TrafficHandlingController
UserData
sccpConnectionIdStatePtr
sccpConnectionIdPtr
callScenarioReq ( )
generateSuggestedConnectionId ( )
trasferMessageToHandler ( )
identifyMessageType ( )
messageIsUnknown ( )
transferToHandlerWithNoConnectionId ( )

Figure 3.4: Traffic Handling Controller Class

As data members, this class provides the *User Data* that implies the RANAP message frame, the *SCCP Connection ID* that identifies the SCCP connection for each call scenario, and the *SCCP Connection ID State*, which is an enumeration data element, this data member is transferred between various components along with the *SCCP Connection ID.* The *SCCP Connection ID State* holds values from 1 to 4, we summaries these values implications for the corresponding SCCP connection as follows:

1: "*connected*": the SCCP connection is connected.

2: "*disconnected*": the SCCP connection is disconnected.

3: "*generated*": the SCCP connection is only generated.

4: "*to be disconnected*" the SCCP connection is intended to be disconnected.

Generally, the *Traffic Handling Controller* class is responsible for (1) controlling the behavior of the call scenarios classes, (2) communicating with the RANAP Controller classes to receive and forward the RANAP protocol message portion (*User Data*), (3) identifying the received message -through the *SCCP connection ID*- and forward it to the proper class object. More details about these class's operations and data elements can be found in Appendix A.1.2.

Location Update Controller Class

The *Location Update Controller* class controls the location update call scenario functionality in the Traffic Handler component.

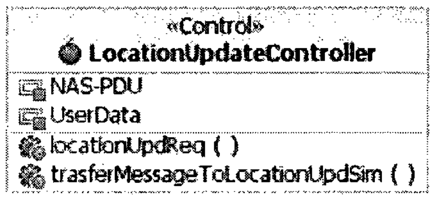


Figure 3.5: Location Update Controller Class

Figure 3.5 shows the UML representation for the location update controller class, this class communicates with the *Traffic Handling Controller* class to get the relevant messages for a specific UE to accomplish the location update call scenario. Also, it interacts with the RANAP proxy, mobility management proxy, and other control classes to generate the sequence of signaling messages that provides the location update traffic in order to load the MSC. More information about these class responsibilities can be found in Section 3.2.1; and more details about the class's operations and data member can be found in Appendix A.1.2.

Call Originating Controller Class

The *Call Originating Controller* class controls the mobile originating call scenario functionality in the Traffic Handler component.

The UML representation for the call originating class is shown in Figure 3.6, the data elements and operations in this class contribute to model the mobile originating call scenario. This class interacts with the *Traffic Handling Controller* class to send and receive the mobile originating call messages, also it cooperates with the RANAP, mobility management, and call control proxies classes to generate the data messages. These messages are used by the *Call Originating Controller* class as requests and responses to originate a call from UE and to interact with MSC to

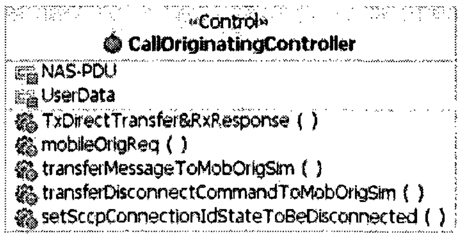

«Control»
CallOriginatingController
NAS-PDU
UserData
TxDirectTransfer&RxResponse ( )
mobileOrigReq ( )
transferMessageToMobOrigSim ( )
transferDisconnectCommandToMobOrigSim ( )
setSccpConnectionIdStateToBeDisconnected ( )

Figure 3.6: Call Originating Controller Class

establish the call with the terminating mobile side. More information about the mobile originating call scenario can be found in Section 3.2.2. Also, this class provides a capability to disconnect the call by releasing all related connections. More details about the class's operations and data member can be found in Appendix A.1.2.

Call Terminating Controller Class

The mobile terminating call scenario is controlled by the *Call Terminating Controller* class.

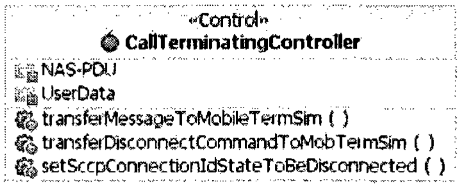

«Control»
CallTerminatingController
NAS-PDU
UserData
transferMessageToMobileTermSim ( )
transferDisconnectCommandToMobTermSim ( )
setSccpConnectionIdStateToBeDisconnected ( )

Figure 3.7: Call Terminating Controller Class

As we mentioned before that the call originating controller class initiates the call by interacting with the MSC, the MSC interacts with the *Call Terminating Controller* Class to establish a call between originating and terminating mobiles. The *Call Terminating Controller* class is provided by this model to represent the mobile terminating side; this class interacts with the *Call Originating Controller* class through the MSC server to generate all signalling messages for mobile calls.

By modeling originating and terminating calls, we will be able to load the MSC with the generated messages [9].

Figure 3.7 shows the UML representation for the call terminating controller class. It cooperates with the RANAP proxy, mobility management proxy, and call control proxy classes to generate the required sequence of messages to simulate the terminating mobile side. More information about the mobile terminating call scenario can be found in Section 3.2.3. Also, this class is able to disconnect the call through clearing all connections and releasing all occupied resources. More details about the class's operations and data member can be found in Appendix A.1.2.

RANAP Procedures Control Classes

In this model, we provide control classes for various RANAP elementary procedures that are required to represent the RANAP protocol for the supported call scenarios that we provide by this model. Each one of these classes is responsible for providing RANAP procedure functionality and interacting with the RANAP message proxy to generate the needed message. One of these classes is the *Initial UE Message Controller* class; this class interacts with the call scenario control classes and the RANAP message proxy to initiate the initial UE message procedure and generate the first RANAP message for the corresponding call scenario.

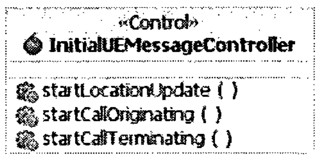


Figure 3.8: Initial UE Message Controller Class

Figure 3.8 shows the UML representation for the *Initial UE Message Controller* class. Other RANAP procedures control classes are; *Direct Tranfer Controller*, *Security Mode Controller*, *Common ID Controller*, *RAB Assignment Controller*, *Paging Controller*, and *Iu Release Controller* class [8], more details about these classes' operations and data members can be found in Appendix A.1.2.

3.2 UML Use-Case Realization

We illustrate the Use Cases' realizations of the Traffic Handler components through classes/objects interactions. Traffic Handler functionalities are realized into seven Use Cases in the Use Case Model, these Use Case are; the *Update Location*, *Originate MS Call*, *Terminate MS Call*, *Disconnect Originating Call Disconnect Terminating Call*, *Distinguish Call Scenarios* and *Handle Traffic* Use Cases. The sequence diagrams describe the logical implementation of the functional specifications that we identified in the Use Case Model. Also, sequence diagrams realize the Use Cases by describing the flow of events in the Use Cases when they are executed. In this section, we describe the sequence diagrams for some of the Traffic Handler Use Cases. More information about other sequence diagrams can be found in Appendix A.2.

3.2.1 Location Update Realization

We present the *Update Location* Use Case realization by illustrating sequence of messages between interacting objects to provide a location update call scenario for a UE. This sequence of messages is demonstrated through the UML sequence diagram.

Figure 3.9 shows the main flow of the location update sequence diagram, it contains the main objects that are instantiated form their classes, and it also shows a sequence of messages or events with numbers on it. The sequence is started by the *location update controller* object (event 1) which asks the *Initial UE Messages controller* object to prepare the location update initial message by contacting the RANAP message proxy and the MM message proxy, this happens through the *get Location Update Request MM Part* operation. The generated data message will be returned back to the location update controller object (event 2), this object sends the message to other control classes to handle it to the MSC (this will be explained in other sequence diagrams).

In events 3 and 4, a *direct transfer RANAP* message will be received and forwarded to the *direct transfer message controller* object to check the requested operation to perform and to send it to the RANAP message proxy then the MM message proxy to perform the operation. The MM message proxy realizes that it is an authentication request message, so it invokes the *authentication Reaction* operation to authenticate the MSC and calculate ciphering keys [7]. An authentication message response will return back to indicate the acceptance or rejection.

In events 5 and 6, the *security mode command* message will be received and sent to the *security mode controller* object (RANAP procedure). After decoding the message, this object cooperates with RANAP message proxy object to perform the *security mode complete* operation by extracting the encryption information and the integrity protection information, then choosing appropriate ciphering and integrity algorithms [6]. A message will be sent to the MSC to confirm these configurations.

In event 7, the common ID message will be received by the location update controller object and sent to the common ID controller object; this message contains the (IMSI) which is sent by the MSC to identify and locate the UE (in this case the UE is represented by the MM message proxy). After identifying the message, the common ID controller object invokes the *perform Common ID on Ranap* operation to forward the message to the RANAP message proxy and store the IMSI and attach it for a specific UE; this operation is called Location Update - IMSI Attach [7][9]. In real application, the RNC creates a reference between the IMSI of the UE and the RRC connection of the same UE to be used in RNC paging procedure [13].

In event 8, the location update controller object receives a direct transfer message that should be forwarded to the MM message proxy. As we explain in the diagram, the MM message proxy uses *alternative combined fragments* (switch condition provided by UML 2.0) to identify the received message type [4]. By

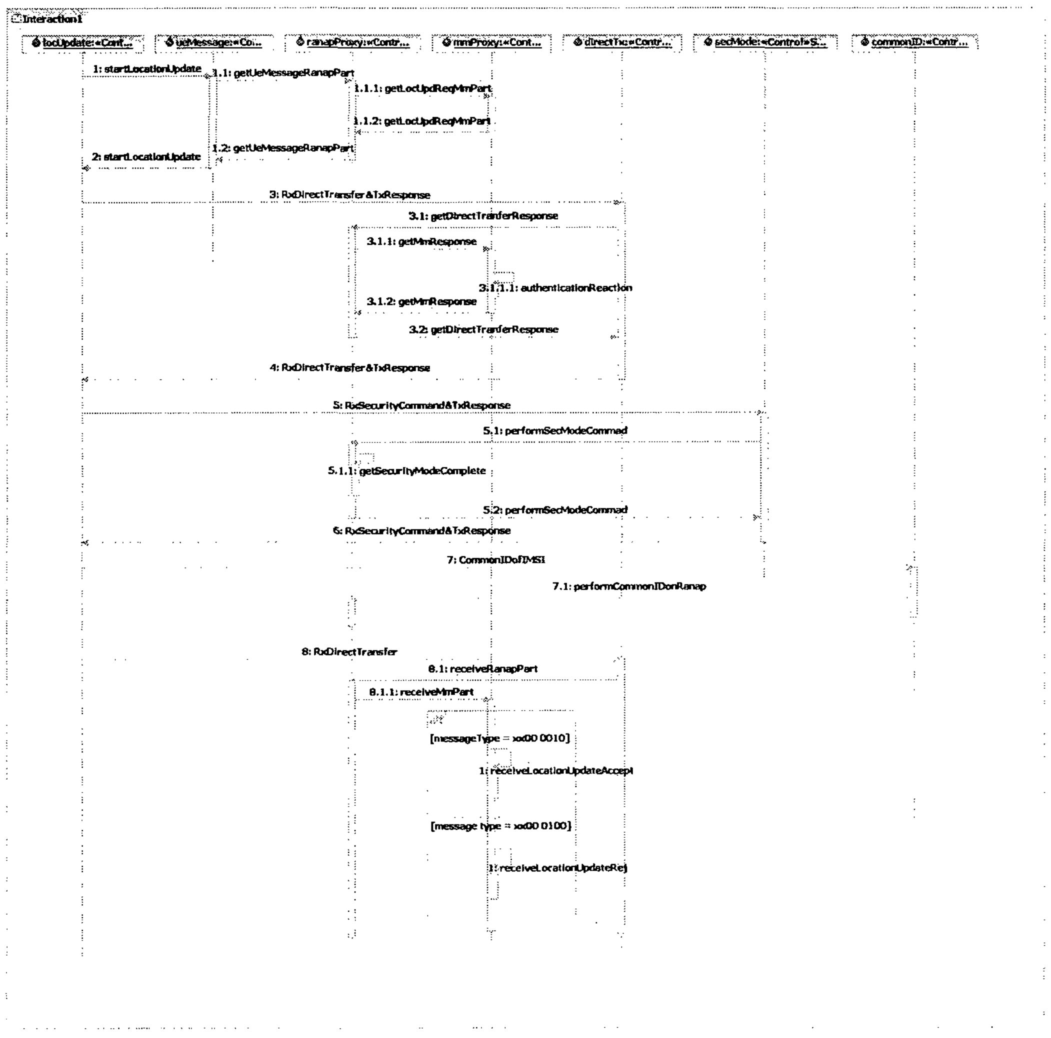

Interaction1
locUpdate:«Cont...
ueMessage:«Co...
ranapProxy:«Contr...
mmProxy:«Cont...
directTx:«Contr...
secMode:«Control»S...
commonID:«Contr...
1: startLocationUpdate
1.1: getUeMessageRanapPart
1.1.1: getLocUpdReqMmPart
1.1.2: getLocUpdReqMmPart
1.2: getUeMessageRanapPart
2: startLocationUpdate
3: RxDirectTransfer&TxResponse
3.1: getDirectTranferResponse
3.1.1: getMmResponse
3.1.1.1: authenticationReaction
3.1.2: getMmResponse
3.2: getDirectTranferResponse
4: RxDirectTransfer&TxResponse
5: RxSecurityCommand&TxResponse
5.1: performSecModeCommad
5.1.1: getSecurityModeComplete
5.2: performSecModeCommad
6: RxSecurityCommand&TxResponse
7: CommonIDofIMSI
7.1: performCommonIDonRanap
8: RxDirectTransfer
8.1: receiveRanapPart
8.1.1: receiveMmPart
[messageType = xx00 0010]
1: receiveLocationUpdateAccept
[message type = xx00 0100]
1: receiveLocationUpdateRej

Figure 3.9: Location Update Sequence Diagram

evaluating the condition, the MM message proxy will be indicated that whether the UE location is updated on the MSC side [7]. If this is the case, the *receive Location Update Accept* operation will be invoked, if the location update is rejected by the MSC, then the *receive Location Update Reject* operation will be invoked. More detailed information about these events' operations can be found in Appendix A.1.

3.2.2 Mobile Originating Call Realization

The proposed model generates messages for the mobile originating call scenario; this functionality is represented by the *Originate MS Call* Use Case.

Figure 3.10 shows the basic flow of the Mobile Call Originating sequence diagram, it shows also the participating objects and the sequence of messages or events with numbers on them. The sequence starts by the call originating controller object (event 1) which asks the Initial UE Messages controller object to originate the call and contact the RANAP messages proxy to generate the Initial UE message for this scenario [6], the RANAP message proxy -through *get CM Service Req Part* operation- asks the MM message proxy to generate the CM service request message, this message is required to establish MM connection in the MSC side, and it will be carried by the Initial UE message to be sent to the MSC (event2).

In event 3, the MM message proxy receives the CM service response message that is forwarded by the direct transfer object and the RANAP message proxy. As it is explained in the figure, the MM message proxy checks the received *message type* to know whether the MM connection is established or not. In events 4 and 5, the sequence of messages represents the authentication operation that is required by the MM message proxy to authenticate the MSC and calculate ciphering key, also events 6 and 7 illustrate the security mode command operation to extract the encryption information and the integrity protection information and to choose appropriate ciphering and integrity algorithms [9]. Through event 8, the call scenario receives the IMSI for the corresponding MM message proxy.

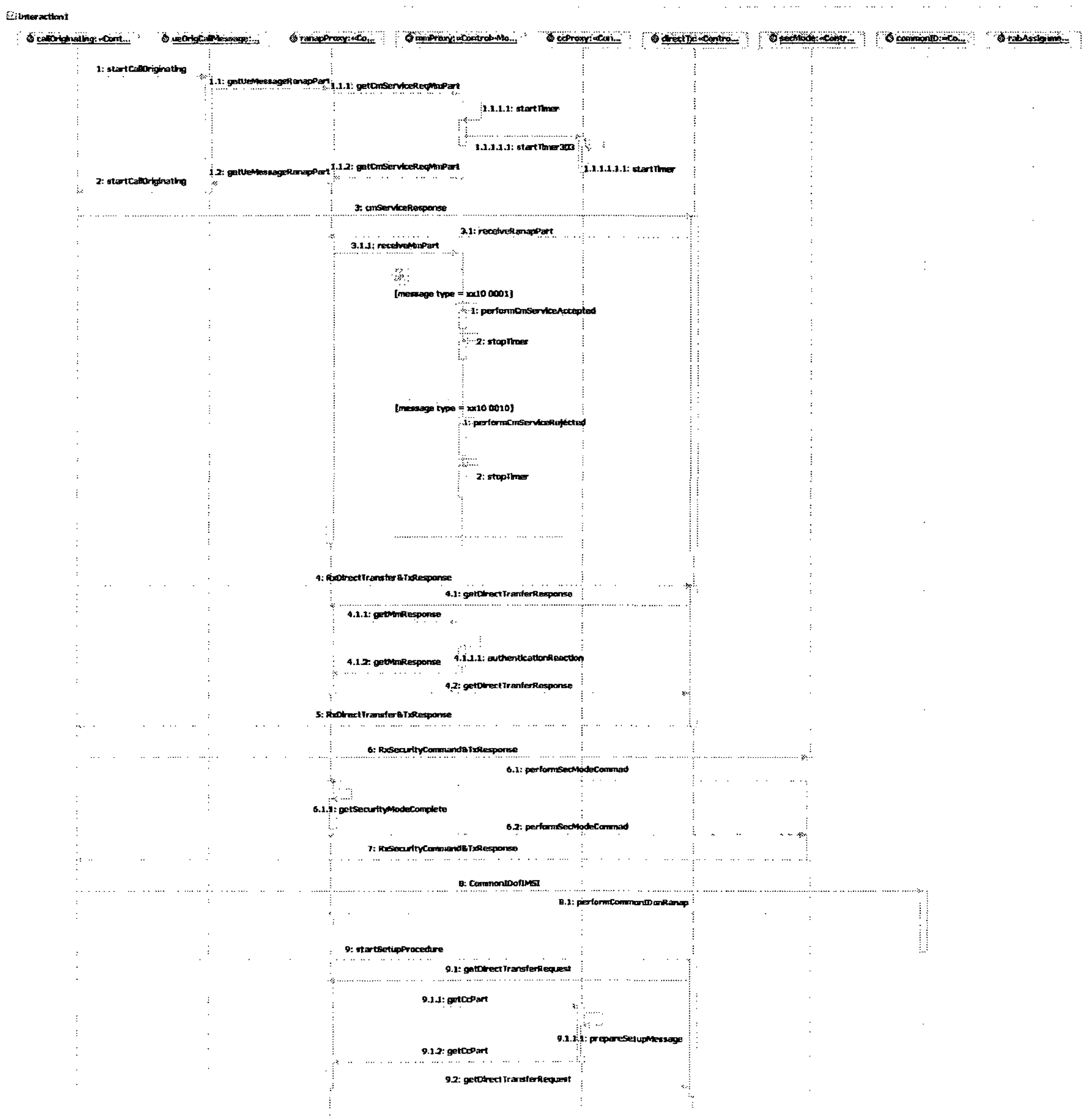

interaction1
callOriginating: «Cont...
ueOrigCallMessage:...
ranapProxy:«Co...
mmProxy:«Controll»Mo...
ccProxy:«Con...
directTx:«Contro...
secMode:«Contr...
commonID:«Co...
rabAssignm...
1: startCallOriginating
1.1: getUeMessageRanapPart
1.1.1: getCmServiceReqMmPart
1.1.1.1: startTimer
1.1.1.1.1: startTimer303
1.1.1.1.1.1: startTimer
1.2: getUeMessageRanapPart
1.1.2: getCmServiceReqMmPart
2: startCallOriginating
3: cmServiceResponse
3.1: receiveRanapPart
3.1.1: receiveMmPart
[message type = xx10 0001]
1: performCmServiceAccepted
2: stopTimer
[message type = xx10 0010]
1: performCmServiceRejected
2: stopTimer
4: RxDirectTransfer&TxResponse
4.1: getDirectTranferResponse
4.1.1: getMmResponse
4.1.1.1: authenticationReaction
4.1.2: getMmResponse
4.2: getDirectTranferResponse
5: RxDirectTransfer&TxResponse
6: RxSecurityCommand&TxResponse
6.1: performSecModeCommad
6.1.1: getSecurityModeComplete
6.2: performSecModeCommad
7: RxSecurityCommand&TxResponse
8: CommonIDofIMSI
8.1: performCommonIDonRanap
9: startSetupProcedure
9.1: getDirectTransferRequest
9.1.1: getCcPart
9.1.1.1: prepareSetupMessage
9.1.2: getCcPart
9.2: getDirectTransferRequest

Figure 3.10: Mobile Originating Call Sequence Diagram (part1)

In event 9, the direct transfer controller object will be ordered by the call originating controller object to prepare a SETUP message, this message shall be generated through invoking the *prepare Setup Message* operation at the call control message proxy. The SETUP message contains the calling and the called party addresses information, also this message will be sent back to the direct transfer controller class. In event 10, the SETUP message will be sent through the direct transfer message to initiate a mobile originating call establishment at the MSC side [7]. In this context, the call control message proxy represents and models the UE's call control portion.

Figure 3.11 shows the completion of the mobile call originating sequence diagram. In events 11 and 12, the call control message proxy receives a CALL PROCEEDING message and invokes the *receive Call Proceeding* operation. This message indicates that the requested call is being processed at the MSC side [7]. In events 13 and 14, the RAB assignment controller object receives a RAB assignment request message which contains the radio access carrier information; the RAB assignment controller object cooperates with the RANAP message proxy to confirm the RAB request that comes from the MSC and send the established/modified RAB ID to the MSC as a RAB assignment response message; the operation *get RAB Assignment Response* will be invoked to perform that [6] [8].

Once the MSC receives the RAB assignment response message, the call control entity of the MSC will send a PAGING request message to page and locate the mobile terminating side; this message will trigger and start the mobile terminating call scenario, the sequence diagram for this call scenario will be explained in Section 3.2.3. At this point the mobile originating call sequence diagram waits until the MSC receives an ALERTING message from the mobile terminating call sequence diagram. Once the MSC receives an ALERTING message, the call control entity of the MSC will send a corresponding ALERTING message to the mobile originating call and the *receive Alerting Message* will be invoked.

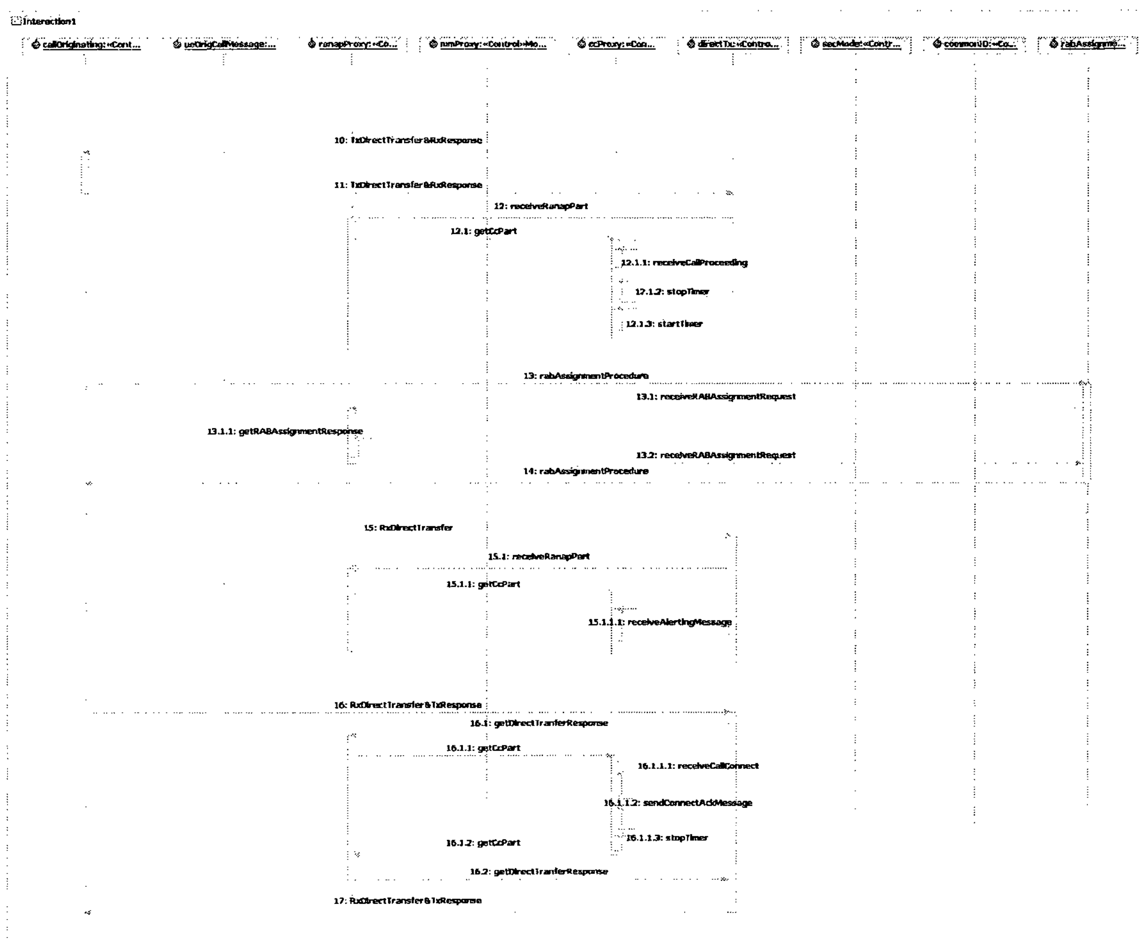

Interaction1
callOriginating:«Cont...
ueOrigCallMessage:...
ranapProxy:«Co...
nmProxy:«Control»Mo...
ccProxy:«Con...
directTx:«Contro...
secModel:«Contr...
commonID:«Co...
rabAssignme...
10: TxDirectTransfer&RxResponse
11: TxDirectTransfer&RxResponse
12: receiveRanapPart
12.1: getCcPart
12.1.1: receiveCallProceeding
12.1.2: stopTimer
12.1.3: startTimer
13: rabAssignmentProcedure
13.1: receiveRABAssignmentRequest
13.1.1: getRABAssignmentResponse
13.2: receiveRABAssignmentRequest
14: rabAssignmentProcedure
15: RxDirectTransfer
15.1: receiveRanapPart
15.1.1: getCcPart
15.1.1.1: receiveAlertingMessage
16: RxDirectTransfer&TxResponse
16.1: getDirectTranferResponse
16.1.1: getCcPart
16.1.1.1: receiveCallConnect
16.1.1.2: sendConnectAckMessage
16.1.1.3: stopTimer
16.1.2: getCcPart
16.2: getDirectTranferResponse
17: RxDirectTransfer&TxResponse

Figure 3.11: Mobile Originating Call Sequence Diagram (part2)

After sending the ALERTING message to the mobile originating side, the MSC receives a CONNECT message from the mobile terminating side. Once the MSC receives that, it will send a corresponding CONNECT message to the CC message proxy object, so the *receive Call Connect* operation will be invoked to indicate call acceptance by the terminating side. Also, the CC message proxy object invokes the *send Connect Acknowledgment Message* operation that sends the CONNECT ACKNOWLEDGMENT message to the MSC to acknowledge the offered connection [7], this sequence can be seen in events 16 and 17. More detailed information about these events' operations and other sequence diagrams can be found in Appendices A.1 and A.2, respectively.

3.2.3 Mobile Terminating Call Realization

In this section, we present the mobile terminating call scenario realization that is represented by the *Terminate MS Call* Use Case in the Use Case Model.

In Figures 3.12 and 3.13, we illustrate the basic flow sequence diagram of the *Terminate MS Call* Use Case; it shows the participating objects and the sequence of messages or events with numbers on them. This sequence diagram interacts with the mobile originating call sequence diagram through the MSC to provide a complete scenario of UE - UE call and to load the MSC consequently.

This sequence diagram is triggered and started by receiving PAGING request message (RANAP message) from the MSC. In event 1 and 2, the call terminating controller object forwards that PAGING request to the paging controller object which communicate with the RANAP message proxy through *perform Paging Request* operation to understand the message elements. In event 3, once the call terminating controller is indicated that PAGING request message has been received, it will ask the Initial UE message controller object and the RANAP message proxy to initiate the first message to be sent to the MSC, then the RANAP message proxy invokes *get Paging Response MM Part* operation to ask the MM message to create

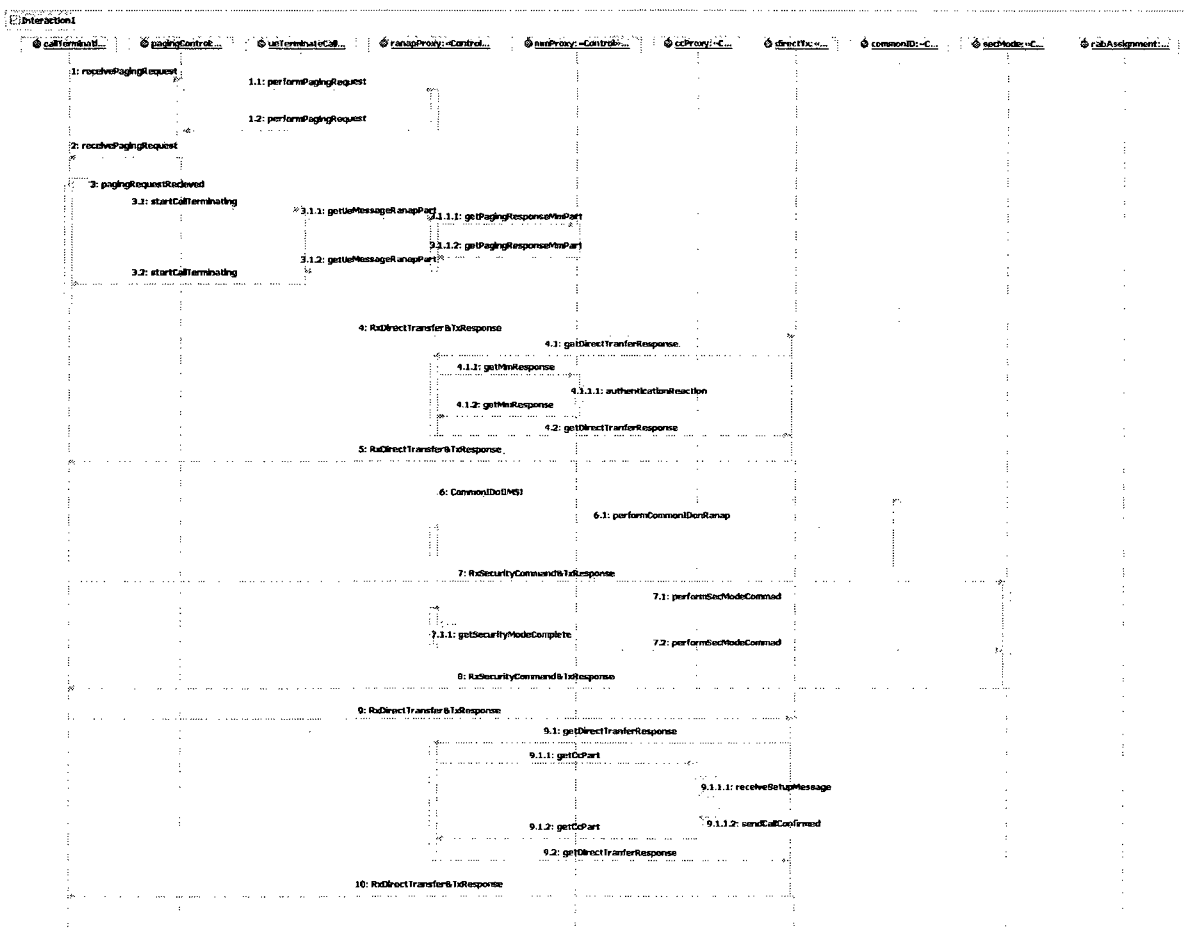

Interaction1
callTerminati...
pagingControl...
ueTerminateCall...
ranapProxy: «Control...
mmProxy: «Control»...
ccProxy: «C...
directTx: «...
commonID: «C...
secMode: «C...
rabAssignment:...
1: receivePagingRequest
1.1: performPagingRequest
1.2: performPagingRequest
2: receivePagingRequest
3: pagingRequestRecieved
3.1: startCallTerminating
3.1.1: getUeMessageRanapPart
3.1.1.1: getPagingResponseMmPart
3.1.1.2: getPagingResponseMmPart
3.1.2: getUeMessageRanapPart
3.2: startCallTerminating
4: RxDirectTransfer&TxResponse
4.1: getDirectTranferResponse
4.1.1: getMmResponse
4.1.1.1: authenticationReaction
4.1.2: getMmResponse
4.2: getDirectTranferResponse
5: RxDirectTransfer&TxResponse
6: CommonIDofIMSI
6.1: performCommonIDonRanap
7: RxSecurityCommand&TxResponse
7.1: performSecModeCommad
7.1.1: getSecurityModeComplete
7.2: performSecModeCommad
8: RxSecurityCommand&TxResponse
9: RxDirectTransfer&TxResponse
9.1: getDirectTranferResponse
9.1.1: getCcPart
9.1.1.1: receiveSetupMessage
9.1.1.2: sendCallConfirmed
9.1.2: getCcPart
9.2: getDirectTranferResponse
10: RxDirectTransfer&TxResponse

Figure 3.12: Mobile Terminating Call Sequence Diagram (part1)

the PAGING response message (MM message). This message will be sent to the MSC to indicate that the required UE is located, the paging procedure is completed, and the MSC can start contacting that UE [6] [7]. As we mentioned before, this model provides the MM and CC message proxies to represent the UE or the mobile station.

In events 4 and 5, the sequence of messages represents the authentication operation that is required by the MM message proxy to authenticate the MSC and calculate ciphering key; through event 8, the call scenario receives the IMSI for the corresponding MM message proxy. Also events 7 and 8 illustrate the security mode command operation to extract the encryption information and the integrity protection information and to choose appropriate ciphering and integrity algorithms [6].

In events 9 and 10, the mobile terminating controller forwards a direct transfer message to the direct transfer controller which sends it to the RANAP message proxy, this message will be forwarded to the CC message proxy. The *receive Setup Message* operation will be invoked at the CC message proxy to indicate that it is a SETUP message; this message is sent by the MSC to initiate a mobile terminated call establishment. In this sequence, once the CC message proxy object receives the SETUP message, it will invoke the *send Call Confirmed* operation to prepare the CALL CONFIRM message; this message will be sent back to the MSC to indicate that the SETUP message has been received properly and the incoming call request has been confirmed [7].

In events 11 and 12, the RAB assignment controller object receives a RAB assignment request message which contains the radio access carrier information; a RAB assignment response message will be sent to MSC to confirm the radio configuration. In events 13 and 14, the mobile terminating controller object asks the RANAP and CC messaging proxies to generate an ALERTING message to be sent to the MSC; this message indicates that the alerting procedure has been

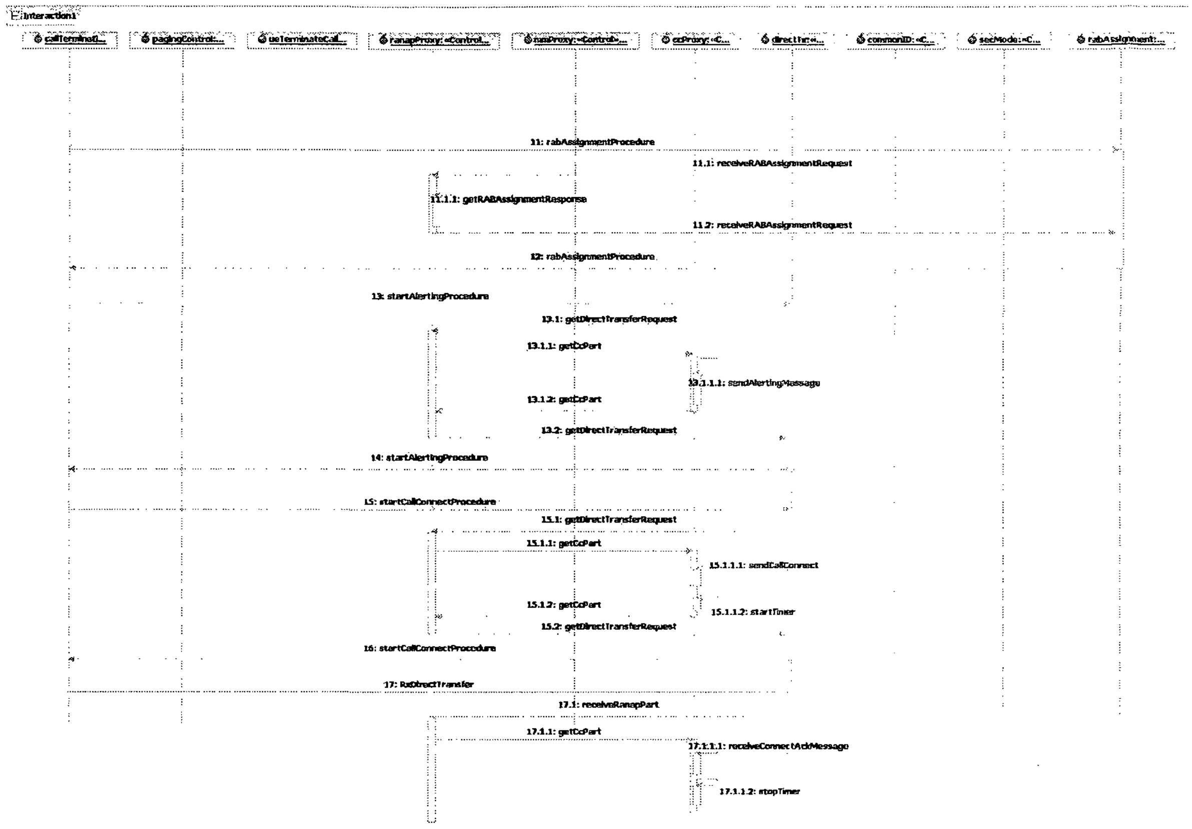

Interaction1
callTerminati...
pagingControl:...
ueTerminateCall...
ranapProxy: «Control...
mmProxy: «Control»...
ccProxy: «C...
directTr: «...
commonID: «C...
secMode: «C...
rabAssignment:...
11: rabAssignmentProcedure
11.1: receiveRABAssignmentRequest
11.1.1: getRABAssignmentResponse
11.2: receiveRABAssignmentRequest
12: rabAssignmentProcedure
13: startAlertingProcedure
13.1: getDirectTransferRequest
13.1.1: getCcPart
13.1.1.1: sendAlertingMessage
13.1.2: getCcPart
13.2: getDirectTransferRequest
14: startAlertingProcedure
15: startCallConnectProcedure
15.1: getDirectTransferRequest
15.1.1: getCcPart
15.1.1.1: sendCallConnect
15.1.2: getCcPart
15.1.1.2: startTimer
15.2: getDirectTransferRequest
16: startCallConnectProcedure
17: RxDirectTransfer
17.1: receiveRanapPart
17.1.1: getCcPart
17.1.1.1: receiveConnectAckMessage
17.1.1.2: stopTimer

Figure 3.13: Mobile Terminating Call Sequence Diagram (part2)

initiated at the terminating side. Based on that, the MSC sends an ALERTING message to CC message proxy object in the call originating sequence diagram as it is explained in Section 3.2.2.

In events 15 and 16, the CC message proxy object invokes the *send Call Connect* operation to prepare a CONNECT message and send it to the MSC; this message indicates that the call has been accepted at the called entity. Once the MSC receives this message, it will send a corresponding CONNECT message to CC message proxy object in the originating side which responds by sending back the CONNECT ACKNOWLEDGMENT. In event 17, the CC message proxy object receives the CONNECT ACKNOWLEDGMENT message by invoking the *receive Connect Acknowledgment Message* which indicates that the CC message proxy has been awarded the call [7]. More detailed information about these events' operations and other sequence diagrams are given in Appendices A.1 and A.2, respectively.

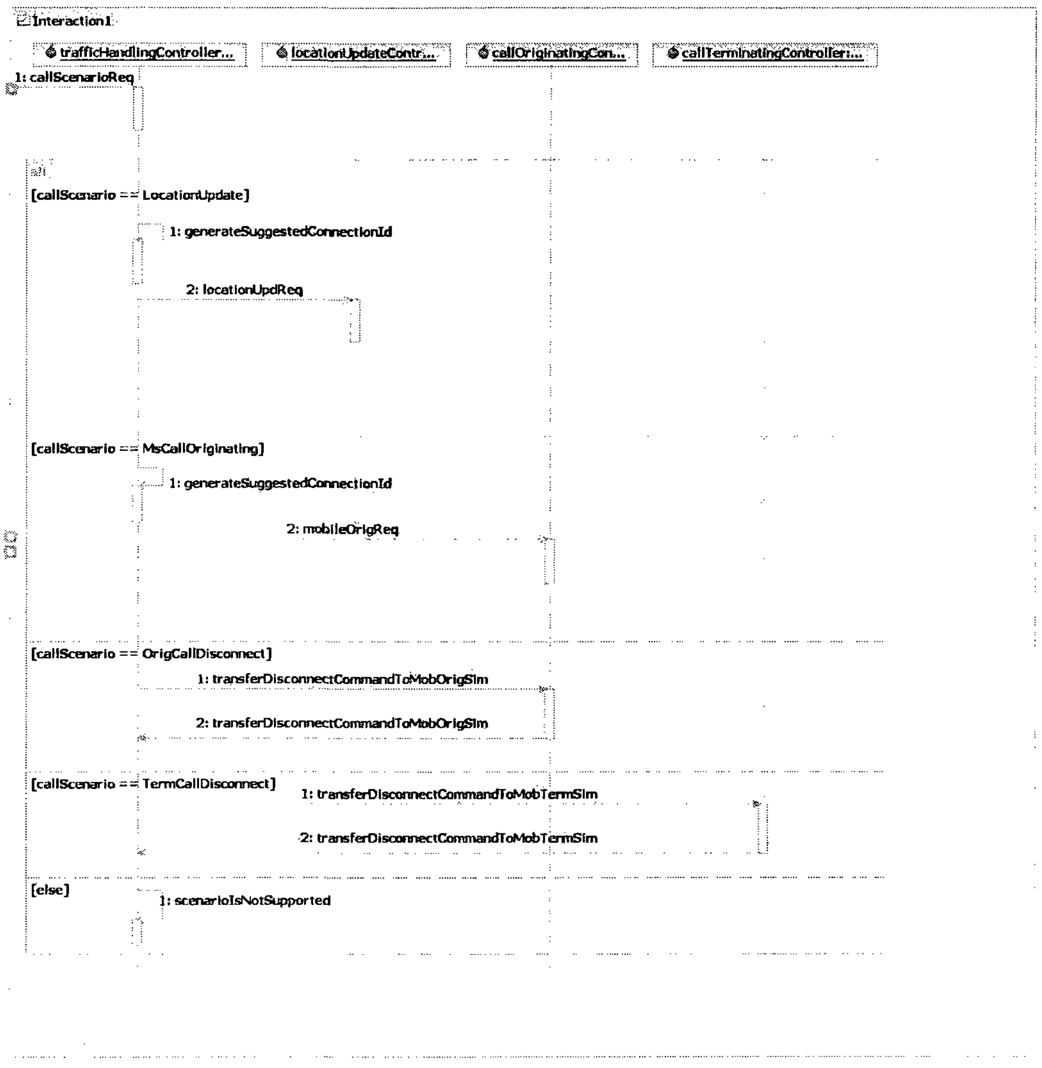
3.2.4 Traffic Handling Realization

The *Handle Traffic* and *Distinguish Call Scenarios* Use Cases plays an important role to handle the traffic in the Traffic Handler component.

The *Distinguish Call Scenarios* Use Case is realized through the the Distinguish Call Scenarios sequence diagram, this sequence is responsible for receiving the call scenario requests -that are forwarded by the RANAP Controller component- to deliver each of them to the right call scenario controller object to initiate the required call scenario or sequence, also for each new call scenario request, a connection ID will be generated to keep track of the call scenario messages. These call requests are originally initiated by the *RanapSim Manager* Actor.

The *Handle Traffic* Use Case is realized by two sequence diagrams one of them is the *Transfer to Handler* sequence diagram, this sequence is functioning to receive the signaling messages from the RANAP Controller component and forward them to the running call scenarios that are interacting with the MSC through the RANAP Controller component. These messages will be forwarded to the right destination based on the connection ID for each call scenario. The other sequence diagram which realizes the *Handle Traffic* Use Case is the *Transfer to RANAP Controller* sequence diagram, this sequence explain how the generated messages from the call scenarios can be forwarded to the RANAP Controller component through the traffic handler controller object.

In this section, we present the *Distinguish Call Scenarios* sequence diagram which is shown in Figure 3.14. In event 1, this sequence starts when the traffic handler controller object receives a call scenario request through the *Call Scenario*



Interaction1
trafficHandlingController...
locationUpdateContr...
callOriginatingCon...
callTerminatingController:...
1: callScenarioReq
alt
[callScenario == LocationUpdate]
1: generateSuggestedConnectionId
2: locationUpdReq
[callScenario == MsCallOriginating]
1: generateSuggestedConnectionId
2: mobileOrigReq
[callScenario == OrigCallDisconnect]
1: transferDisconnectCommandToMobOrigSim
2: transferDisconnectCommandToMobOrigSim
[callScenario == TermCallDisconnect]
1: transferDisconnectCommandToMobTermSim
2: transferDisconnectCommandToMobTermSim
[else]
1: scenarioIsNotSupported

Figure 3.14: Distinguish Call Scenarios Sequence Diagram

parameter, as explained in the diagram, the traffic handler controller object uses *alternative combined fragments* (switch condition provided by UML 2.0) to identify the received call scenario request [4], this switch condition provides blocks for different cases.

In the first block of the switch condition, the *call Scenario* will be checked whether the request is location update, if this the case, the *location Update Request* operation will be invoked to initiate the call and the *generate Suggested Connection ID* operation will be invoked to generate a new connection ID and connection ID state to be attached with all messages that are related to the initiated call scenario. The new connection ID is indicated by the *SCCP Connection ID* data element, and the new connection ID state is indicated by the *SCCP Connection Id state* data element which holds the "*generated*" state, more details will be given in Chapter 4.

The second block does the same for the mobile call originating, so if the request is to originate a call, the mobile originating call scenario will be triggered. In the third and forth blocks, we explain if the call scenario request is to disconnect the call on the mobile originating side or disconnect the call on the terminating side, consequently, a corresponding sequence diagram will be initiated to disconnect the call. Since the mobile terminating call is initiated by PAGING request message from the MSC, so there is no call scenario request indicates a block for the terminating call scenario. More detailed information about these events' operations and other sequence diagrams for the Traffic Handler component can be found Appendices A.1 and A.2, respectively.

3.3 UML Class Diagram

This section describes the static structure of the Traffic Handler component by illustrating the class diagrams. In this context, a class diagram helps to understand the requirements of the Traffic Handler and to describe exactly how this component works. Furthermore, a class diagram defines the relationships between classes and illustrates the structure of the model by using attributes, operations, signals, and interfaces. Also, it shows an inheritance hierarchy among classes [1].

In class diagrams, we use interfaces to facilitate the job and give more details about the classes relationships. Interfaces are model elements that define sets of operations that other classes must implement. We can use interfaces in class diagrams to specify a contract between the interface and the class that realizes the interface. Each interface specifies a well-defined set of operations that have public visibility. Those operations will be provided to another class through a *dependency/use* relationship [4]. In UML, we call this relationship between the interface and its implementing class *interface realization* relationship. In next sections we provide the class diagrams for the Traffic Handler component.

3.3.1 Location Update Class Diagram

We describe the class diagram for the location update call scenario that is provided by the Traffic Handler component, Figure 3.5 shows the UML representation for this class diagram.

The location update class diagram contains all classes that participate to generate the signaling messages for the location update call scenario, also it consists of some interfaces to handle and explain the relationships between the classes.

From the class diagram shown in Figure 3.5, the Location Update Controller class uses the *IRanapProcedures_LocUpd* Interface through a *dependency/use* relationship [4]. This interface is implemented through an *interface realization* relationship by the Direct Transfer Controller, Security Mode Controller, Common ID Controller, and Initial UE Message Controller classes. Those classes use the *IRanapMessage_LocUpd* Interface which is realized or implemented by the RANAP message proxy class, through this interface, the RANAP message proxy class is able to specify the required operations to provide them to other classes.

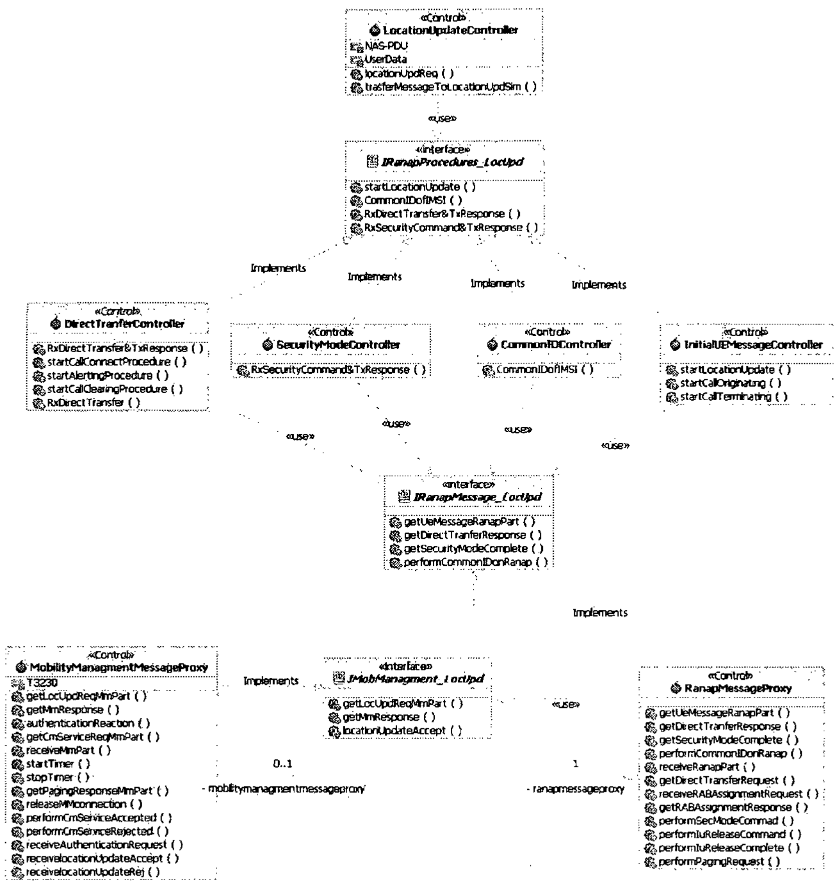

«Control»
LocationUpdateController
NAS-PDU
UserData
locationUpdReq ( )
trasferMessageToLocationUpdSim ( )
«use»
«interface»
IRanapProcedures_LocUpd
startLocationUpdate ( )
CommonIDofIMSI ( )
RxDirectTransfer&TxResponse ( )
RxSecurityCommand&TxResponse ( )
Implements
Implements
Implements
Implements
«Control»
DirectTranferController
RxDirectTransfer&TxResponse ( )
startCallConnectProcedure ( )
startAlertingProcedure ( )
startCallClearingProcedure ( )
RxDirectTransfer ( )
«Control»
SecurityModeController
RxSecurityCommand&TxResponse ( )
«Control»
CommonIDController
CommonIDofIMSI ( )
«Control»
InitialUEMessageController
startLocationUpdate ( )
startCallOriginating ( )
startCallTerminating ( )
«use»
«use»
«use»
«use»
«interface»
IRanapMessage_LocUpd
getUeMessageRanapPart ( )
getDirectTranferResponse ( )
getSecurityModeComplete ( )
performCommonIDonRanap ( )
Implements
«Control»
MobilityManagmentMessageProxy
T3230
getLocUpdReqMmPart ( )
getMmResponse ( )
authenticationReaction ( )
getCmServiceReqMmPart ( )
receiveMmPart ( )
startTimer ( )
stopTimer ( )
getPagingResponseMmPart ( )
releaseMMconnection ( )
performCmServiceAccepted ( )
performCmServiceRejected ( )
receiveAuthenticationRequest ( )
receivelocationUpdateAccept ( )
receivelocationUpdateRej ( )
Implements
«interface»
IMobManagment_LocUpd
getLocUpdReqMmPart ( )
getMmResponse ( )
locationUpdateAccept ( )
«use»
0..1
- mobilitymanagmentmessageproxy
1
- ranapmessageproxy
«Control»
RanapMessageProxy
getUeMessageRanapPart ( )
getDirectTranferResponse ( )
getSecurityModeComplete ( )
performCommonIDonRanap ( )
receiveRanapPart ( )
getDirectTransferRequest ( )
receiveRABAssignmentRequest ( )
getRABAssignmentResponse ( )
performSecModeCommad ( )
performIuReleaseCommand ( )
performIuReleaseComplete ( )
performPagingRequest ( )

Figure 3.15: Location Update Class Diagram

The RANAP message proxy class uses some operations from the Mobility Management message proxy class which provides these operations through the *IMobManagment_LocUpd* Interface. The RANAP message proxy class has an *association/aggregation* relationship [4] with the MM message proxy class with (1 - 0..1) multiplicity. In this context, this notation means that one object of the RANAP message proxy may have zero or one object of the MM message proxy. Based on the used RANAP procedure, some of the RANAP messages may have a MM message inside. More information about the Interfaces in this class diagram can be found in Appendix A.3.

3.3.2 Mobile Originating Call Class Diagram

The mobile originating call scenario is provided by the Traffic Handler component, and we have presented the classes and the Use Case realization of modeling this scenario. In this section, we present the class diagram for the mobile originating call scenario; Figure 3.16 shows the UML representation for the mobile originating call class diagram.

This class diagram consists of all classes that participate to originate a call and consequently generate a signaling traffic for this call scenario, also several UML interfaces and relationships have been used to show more details about the classes' communications. From what is shown in the class diagram; the main class for this scenario is the Call Originating Controller class which has a *dependency/use* relationship with the *IRanapProcedures_MobOrig* Interface to use the interface provided services. This interface is implemented through an *interface realization* relationship by the Direct Transfer Controller, Security Mode Controller, Common ID Controller, Initial UE Message Controller, RAB Assignment Controller, Iu Release Controller classes. Those classes are having some messaging data services through using the *IRanapMessage* Interface which is realized or implemented by the RANAP message proxy class, so the RANAP message proxy class specifies -using this interface- the required operations to provide.

The RANAP message proxy class interacts with the Mobility Management message proxy to use some operations that are provided by the MM message proxy class.

The *IMobManagment_MobOrig* Interface is implemented by the MM message proxy class through the *interface realization* relationship and used by the RANAP message proxy class through the *dependency/use* relationship. The same thing will be applied for services that are provided by the call control message proxy class to the RANAP message proxy class, the *ICallControl_MobOrig* Interface will be implemented by the CC message proxy class, and used by the RANAP message proxy

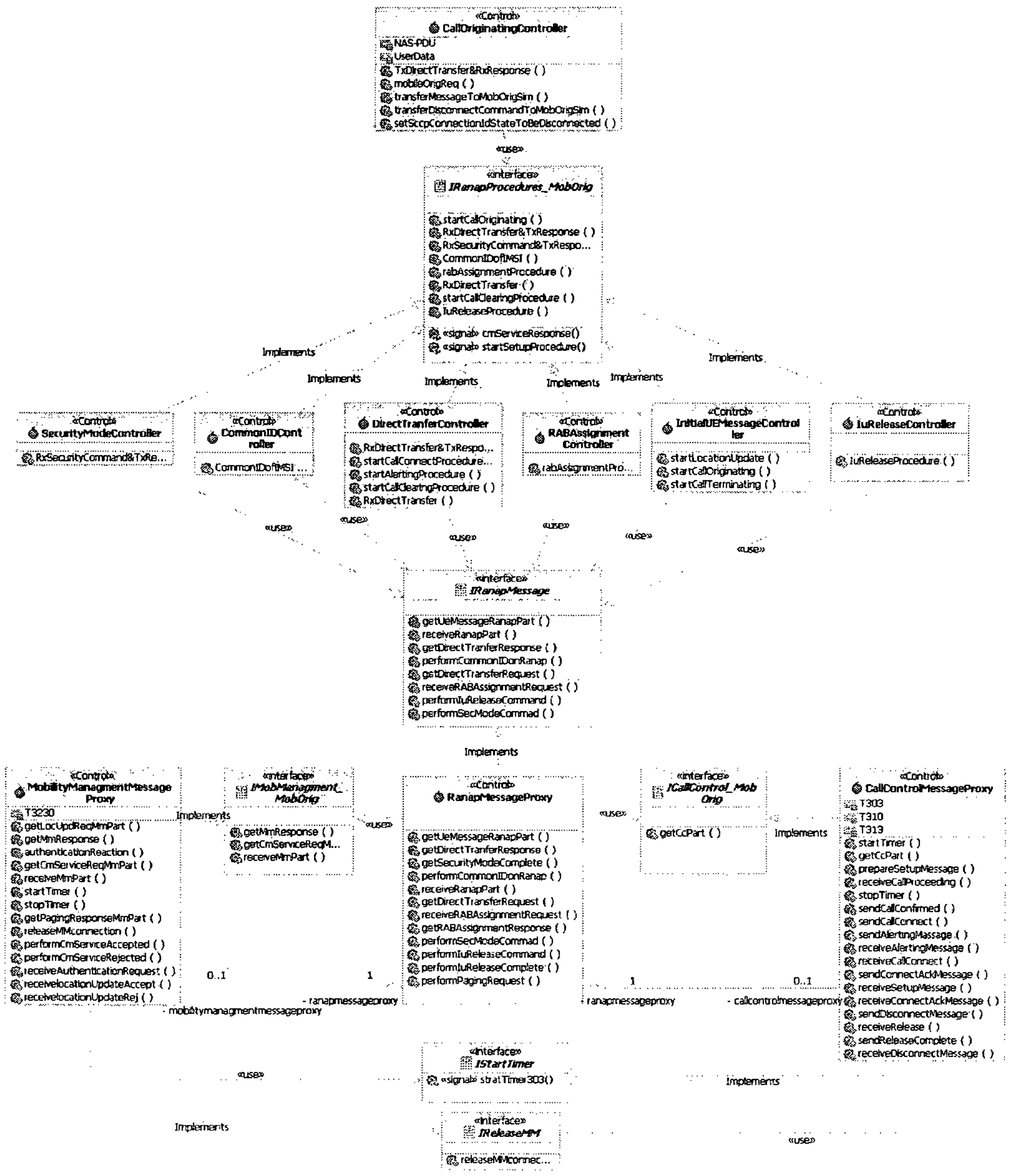


Figure 3.16: Mobile Originating Call Class Diagram

class. The RANAP message proxy class has an *association/aggregation* relationship [4] with the MM message proxy classes with (1 - 0..1) multiplicity, the RANAP message proxy class also has the same relationship with CC message proxy class. The *association/aggregation* relationship for the RANAP message proxy class has been explained in the previous section. On the other hand, the CC and the MM messaging proxies' classes interacts with each other directly through the *IStartTimer* and the *IReleaseMM* Interfaces. More information about the Interfaces in this class diagram can be found in Appendix A.3.

3.3.3 Mobile Terminating Call Class Diagram

This section illustrates the class diagram for the mobile terminating call scenario; the class diagram for this call scenario is represented in Figure 3.17.

As described in the figure, the main controller class for this scenario is the Call Terminating Controller class that uses the *IRanapProcedures_MobTerm* Interface to interact with the Direct Transfer Controller, Security Mode Controller, Common ID Controller, Initial UE Message Controller, RAB Assignment Controller, Iu Release Controller, and Paging Controller classes; those classes also interact with the RANAP message proxy class through the *IRanapMessage* Interface, this interface provides all operations required by these RANAP procedures' classes.

In this class diagram, we illustrate that the RANAP message proxy class uses some operation that are provided by the MM message proxy class through the *IMob-Managment_MobTerm* Interface, also the RANAP message proxy class interacts with the CC message proxy class through the *ICallControl_MobTerm* Interface. These two interfaces show exactly what the RANAP message proxy class needs from the MM and CC messaging proxies, in addition the RANAP message proxy class has two *association/aggregation* relationships with the MM and CC messaging proxies; these relationships are similar to the relationships that are explained in the Mobile Originating Call Class Diagram Section. More details about these interfaces can be

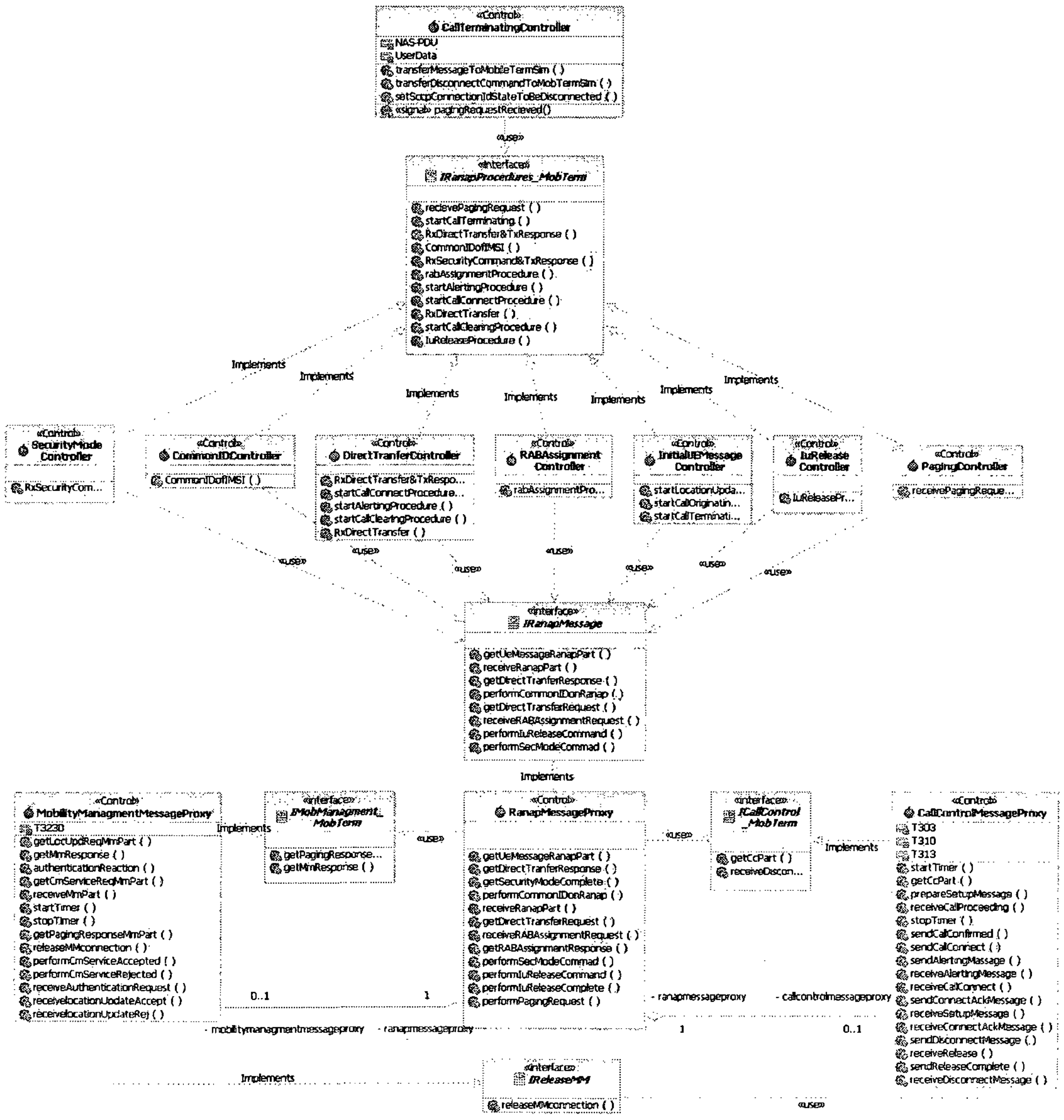

«Control»
CallTerminatingController
NAS-PDU
UserData
transferMessageToMobileTermSim ( )
transferDisconnectCommandToMobTermSim ( )
setSccpConnectionIdStateToBeDisconnected ( )
«signal» pagingRequestRecieved()
«use»
«interface»
IRanapProcedures_MobTerm
recievePagingRequest ( )
startCallTerminating ( )
RxDirectTransfer&TxResponse ( )
CommonIDofIMSI ( )
RxSecurityCommand&TxResponse ( )
rabAssignmentProcedure ( )
startAlertingProcedure ( )
startCallConnectProcedure ( )
RxDirectTransfer ( )
startCallClearingProcedure ( )
IuReleaseProcedure ( )
Implements
«Control»
SecurityMode Controller
RxSecurityCom...
«Control»
CommonIDController
CommonIDofIMSI ( )
«Control»
DirectTranferController
RxDirectTransfer&TxRespo...
startCallConnectProcedure...
startAlertingProcedure ( )
startCallClearingProcedure ( )
RxDirectTransfer ( )
«Control»
RABAssignment Controller
rabAssignmentPro...
«Control»
InitialUEMessage Controller
startLocationUpda...
startCallOriginatin...
startCallTerminati...
«Control»
IuRelease Controller
IuReleasePr...
«Control»
PagingController
receivePagingReque...
«use»
«interface»
IRanapMessage
getUeMessageRanapPart ( )
receiveRanapPart ( )
getDirectTranferResponse ( )
performCommonIDonRanap ( )
getDirectTransferRequest ( )
receiveRABAssignmentRequest ( )
performIuReleaseCommand ( )
performSecModeCommad ( )
Implements
«Control»
MobilityManagmentMessageProxy
T3230
getLocUpdReqMmPart ( )
getMmResponse ( )
authenticationReaction ( )
getCmServiceReqMmPart ( )
receiveMmPart ( )
startTimer ( )
stopTimer ( )
getPagingResponseMmPart ( )
releaseMMconnection ( )
performCmServiceAccepted ( )
performCmServiceRejected ( )
receiveAuthenticationRequest ( )
receivelocationUpdateAccept ( )
receivelocationUpdateRej ( )
«interface»
IMobManagment_MobTerm
getPagingResponse...
getMmResponse ( )
«Control»
RanapMessageProxy
getUeMessageRanapPart ( )
getDirectTranferResponse ( )
getSecurityModeComplete ( )
performCommonIDonRanap ( )
receiveRanapPart ( )
getDirectTransferRequest ( )
receiveRABAssignmentRequest ( )
getRABAssignmentResponse ( )
performSecModeCommad ( )
performIuReleaseCommand ( )
performIuReleaseComplete ( )
performPagingRequest ( )
«interface»
ICallControl_MobTerm
getCcPart ( )
receiveDiscon...
«Control»
CallControlMessageProxy
T303
T310
T313
startTimer ( )
getCcPart ( )
prepareSetupMessage ( )
receiveCallProceeding ( )
stopTimer ( )
sendCallConfirmed ( )
sendCallConnect ( )
sendAlertingMessage ( )
receiveAlertingMessage ( )
receiveCallConnect ( )
sendConnectAckMessage ( )
receiveSetupMessage ( )
receiveConnectAckMessage ( )
sendDisconnectMessage ( )
receiveRelease ( )
sendReleaseComplete ( )
receiveDisconnectMessage ( )
0..1
1
- mobilitymanagmentmessageproxy
- ranapmessageproxy
- callcontrolmessageproxy
«interface»
IReleaseMM
releaseMMconnection ( )

Figure 3.17: Mobile Terminating Call Class Diagram

found in Appendix A.3.

3.3.4 Traffic Handling Class Diagram

We describe the class diagram for the control functionality of the Traffic Handler component; Figure 3.18 shows the UML representation for this class diagram.

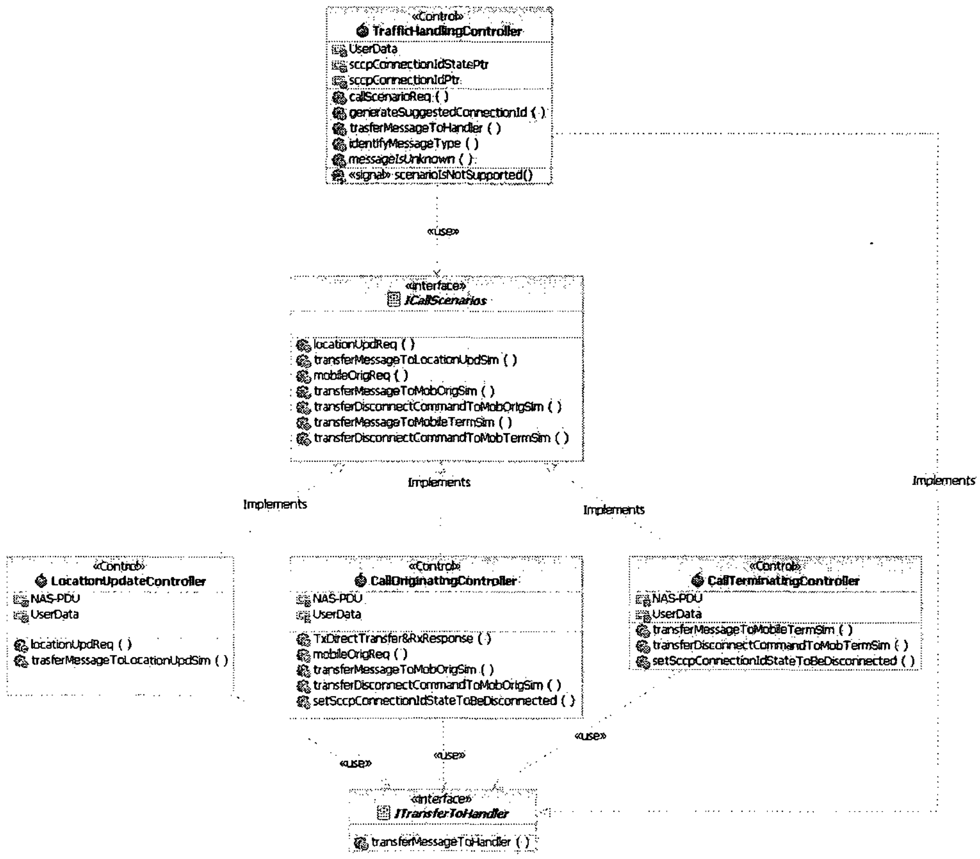

«Control»
TrafficHandlingController
UserData
sccpConnectionIdStatePtr
sccpConnectionIdPtr
callScenarioReq ( )
generateSuggestedConnectionId ( )
trasferMessageToHandler ( )
identifyMessageType ( )
messageIsUnknown ( )
«signal» scenarioIsNotSupported()
«use»
«interface»
ICallScenarios
locationUpdReq ( )
transferMessageToLocationUpdSim ( )
mobileOrigReq ( )
transferMessageToMobOrigSim ( )
transferDisconnectCommandToMobOrigSim ( )
transferMessageToMobileTermSim ( )
transferDisconnectCommandToMobTermSim ( )
Implements
Implements
Implements
Implements
«Control»
LocationUpdateController
NAS-PDU
UserData
locationUpdReq ( )
trasferMessageToLocationUpdSim ( )
«Control»
CallOriginatingController
NAS-PDU
UserData
TxDirectTransfer&RxResponse ( )
mobileOrigReq ( )
transferMessageToMobOrigSim ( )
transferDisconnectCommandToMobOrigSim ( )
setSccpConnectionIdStateToBeDisconnected ( )
«Control»
CallTerminatingController
NAS-PDU
UserData
transferMessageToMobileTermSim ( )
transferDisconnectCommandToMobTermSim ( )
setSccpConnectionIdStateToBeDisconnected ( )
«use»
«use»
«use»
«interface»
ITransferToHandler
transferMessageToHandler ( )

Figure 3.18: Traffic Handling Class Diagram

The class diagram shows the Traffic Handling Controller class uses services that are provided by the controllers' classes of the call scenarios through the *ICallScenario* Interface; the Traffic Handling Controller class has a *dependency/use* relationship with this Interface as illustrated in this Figure. The *ICallScenario* Interface is realized by the Location Update Controller, Call Originating Controller, and Call Terminating Controller classes. On the other hand, the Traffic Handler Controller class implements the *ITransferToHandler* Interface that is used by call scenarios controller classes.

3.3.5 Traffic Messaging Class Diagrams

The Traffic Handler component provides models for call scenarios to generate signaling traffic, the control classes for each call scenario model should interact with the messaging proxies' classes to handle the request and create the required message. Those messaging proxies are; the RANAP message proxy, the Mobility Management proxy, and the Call Control proxy classes. Next, we are going to present the class diagrams for those messages.

RANAP Message

The RANAP Message class diagram shows the several types of RANAP messages where each one is used for specific service; Figure 3.19 shows the UML representation for the RANAP message class diagram.

This class diagram shows different types of RANAP messages classes as *Entity* classes, these classes are; Direct Transfer, Initial UE, Common ID, Security Mode Command, Security Mode Complete, RAB Assignment, Paging Request, as well as Iu Release messages' classes [6]. Each one of these classes has an *association/composition* relationship with the RANAP message proxy class with (1 - 0..1) multiplicity [4]. This relationship indicates that those messages classes always belong to the RANAP message proxy class. The multiplicity (1 - 0..1) means that one object of the RANAP message proxy may own zero or one object of each type of the RANAP messages' objects. Through this relationship, the RANAP message proxy class is able to access the messages' classes through its operation. More information about these classes can be found in Appendix A.1.1.

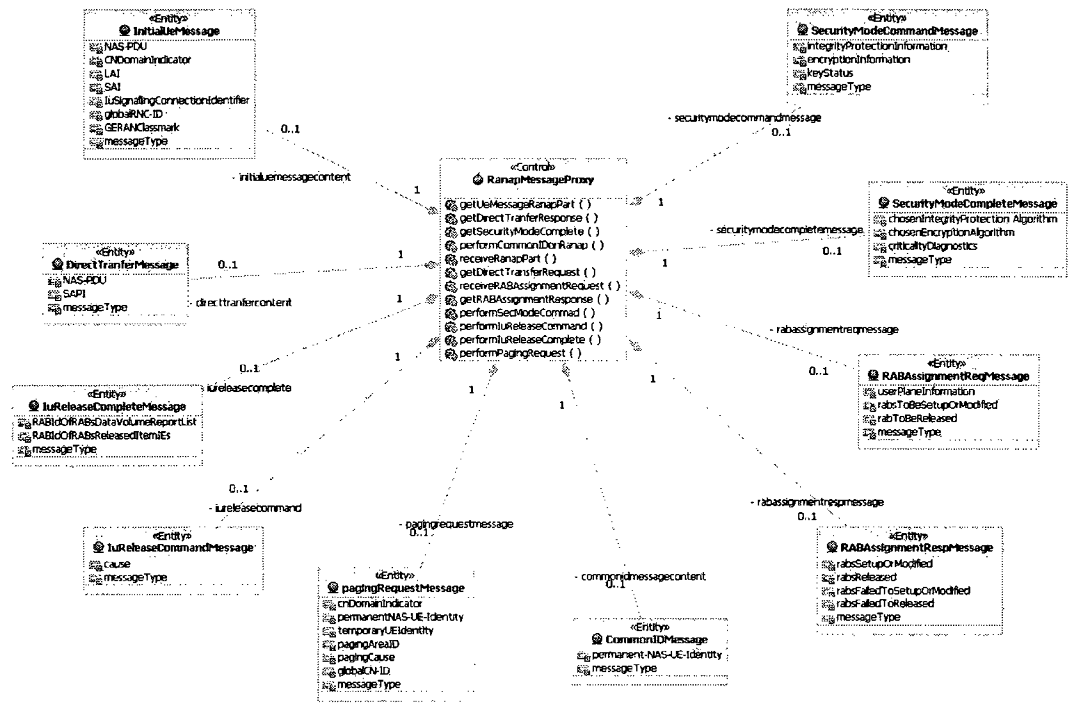

«Entity» InitialUeMessage
NAS-PDU
CNDomainIndicator
LAI
SAI
IuSignallingConnectionIdentifier
globalRNC-ID
GERANClassmark
messageType
- initialuemessagecontent
«Control» RanapMessageProxy
getUeMessageRanapPart ( )
getDirectTranferResponse ( )
getSecurityModeComplete ( )
performCommonIDonRanap ( )
receiveRanapPart ( )
getDirectTransferRequest ( )
receiveRABAssignmentRequest ( )
getRABAssignmentResponse ( )
performSecModeCommad ( )
performIuReleaseCommand ( )
performIuReleaseComplete ( )
performPagingRequest ( )
«Entity» SecurityModeCommandMessage
integrityProtectionInformation
encryptionInformation
keyStatus
messageType
- securitymodecommandmessage
«Entity» SecurityModeCompleteMessage
chosenIntegrityProtection Algorithm
chosenEncryptionAlgorithm
criticalityDiagnostics
messageType
- securitymodecompletemessage
«Entity» DirectTranferMessage
NAS-PDU
SAPI
messageType
- directtranfercontent
- rabassignmentreqmessage
«Entity» RABAssignmentReqMessage
userPlaneInformation
rabsToBeSetupOrModified
rabToBeReleased
messageType
«Entity» IuReleaseCompleteMessage
RABIdOfRABsDataVolumeReportList
RABIdOfRABsReleasedItemIEs
messageType
- iureleasecomplete
- iureleasecommand
«Entity» IuReleaseCommandMessage
cause
messageType
- pagingrequestmessage
«Entity» pagingRequestMessage
cnDomainIndicator
permanentNAS-UE-Identity
temporaryUEIdentity
pagingAreaID
pagingCause
globalCN-ID
messageType
- commonidmessagecontent
«Entity» CommonIDMessage
permanent-NAS-UE-Identity
messageType
- rabassignmentrespmessage
«Entity» RABAssignmentRespMessage
rabsSetupOrModified
rabsReleased
rabsFailedToSetupOrModified
rabsFailedToReleased
messageType
0..1
1

Figure 3.19: RANAP Message Class Diagram

Mobility Management Message

In this section, we illustrate the Mobility Management Message class diagram that shows different types of the Mobility Management messages, each one of these messages is utilized for a specific mobility service.

The Mobility Management Message class diagram is shown in Figure 3.20. In this class diagram, several *Entity* classes have been shown to represent several types of the Mobility Management messages, these classes are; Location Update Request/Accept/Reject, Authentication Request/Response/Reject, Connection Management Service Request/Accept/Reject, as well as Paging Response messages' classes [7]. The Mobility Management message proxy class is shown as a central control class which is able to control and handle all mobility management requests through having access to all these messages' classes; this access is achieved through

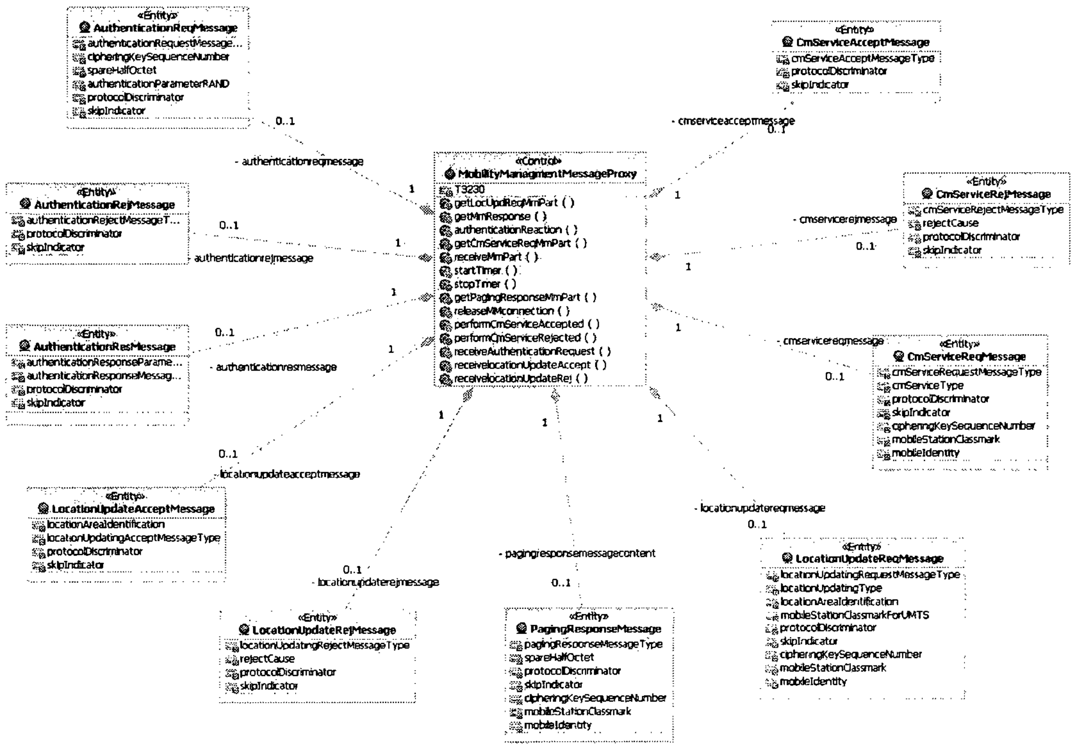


Figure 3.20: Mobility Management Message Class Diagram

the *association/composition* relationship with (1 - 0..1) multiplicity [4].

This relationship indicates that the Mobility Management message proxy class always contains the mobility management messages classes, and the (1 - 0..1) multiplicity indicates that one object of the Mobility Management message proxy may own zero or one object of each type of the MM messages' objects. More information about these classes can be found in Appendix A.1.1.

Call Control Message

The Call Control Message class diagram shows all control and entity classes that cooperate to build the call control message; Figure 3.21 shows the UML representation for the Call Control message class diagram.

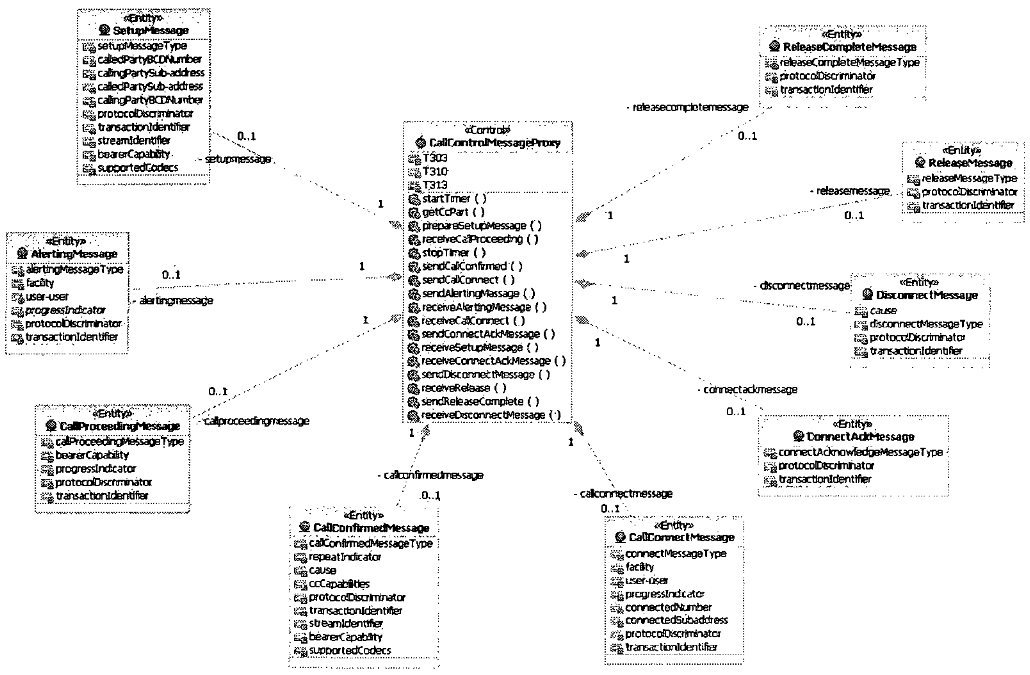


Figure 3.21: Call Control Message Class Diagram

This class diagram shows several types of the call control messages classes which are modeled as *Entity* classes, these classes are; Alerting, Setup, Call Proceeding, Call Confirm, Call Connect, Connect Acknowledgment, Disconnect, Release, as well as Release Complete messages' classes [7].

Each one of these classes has an *association/composition* relationship with the proxy class of the call control message with (1 - 0..1) multiplicity [4]. This relationship indicates that those messages classes always belong to the Call Control message proxy class and the (1 - 0..1) multiplicity means that one object of the call control message proxy may own zero or one object of each type of the call control messages' objects. More information about these classes can be found in Appendix A.1.1.

By modeling the MM and the CC messaging in the Traffic Handler components, our models are able to provide the functional behavior of the UE device which is necessary to generate the traffic to load the MSC.

3.4 Summary

In this chapter, we provided the *UML Analysis Model* for the Traffic Handler component. This model provides the logical implementation of the functional description for the Traffic Handler's Use Cases which are; *Update Location*, *Originate MS Call*, *Terminate MS Call*, *Disconnect Originating Call Disconnect Terminating Call*, *Distinguish Call Scenarios* and *Handle Traffic* Use Cases.

To achieve that, we introduced the UML analysis classes for the messaging proxies and the call scenarios. Also, we described the Use Cases' realizations which are provided by the Traffic Handler component; the realization introduced all possible interactions and sequence flows that reflect the functional behavior of each Use Case. Furthermore, we also introduced all class diagrams that illustrate the structure of the Traffic Handler component by showing the component's classes, their attributes, and UML relationships between the classes.

Through the Analysis Model, the *Radio Access Network Application Part* (RANAP) protocol's procedures have been modeled, those procedures generate the radio access network messages in the IuCS Interface. Also, the mobile radio interface protocols in the UMTS network have been modeled, these protocols are the Mobility Management and the Call Control protocols which are responsible for representing the UE calls' scenarios. For the Traffic Handler component, all UML diagrams and other modeling details have been checked by a committee of senior software engineers at Ericsson Research Canada through an internal *formal check* process [50], where they evaluate the correctness of technical contents of the design based on the standard specification. In next chapter, we are going to introduce the detailed modeling of the RANAP and SCCP Controllers.

Chapter 4

RANAP and SCCP Controllers

In this chapter, we introduce detailed modeling for the RANAP Controller component, this component controls and handles all interactions between the RanapSim components. We also present the modeling of the SCCP Interface Controller component which provides a connectivity functionality between the RanapSim traffic generator and the CPP platform.

4.1 RANAP Controller

This section presents a detailed model for the RANAP Controller component which is the main controller in the traffic generator design. The RANAP Controller is responsible for controlling all interactions between the Traffic Handler and the SCCP Interface Controller components. The RANAP Controller handle all requests from the RanapSim Manager which is represented as an Actor in this model; this component is realized in the Use Case Model through the *Attach SCCP*, *Detach SCCP*, *Check SCCP Service*, *Forward Call Scenario Messages* and *Forward CPP Response Messages* Use Cases. In this section, we present the *UML Analysis Model* for the RANAP Controller component.

4.1.1 UML Analysis Classes

In this section, we are going to introduce the analysis classes for RANAP Controller component that cooperate to achieve the main objectives of this component. In UML, the instantiated objects of these classes are used to build the interaction diagrams [4].

RANAP Interface Controller Class

The *Ranap Interface Controller* Class is the main control class of the RANAP Controller component.

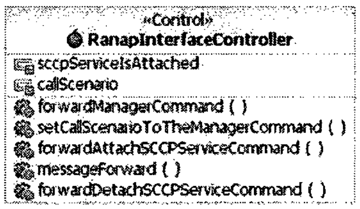

«Control»
RanapInterfaceController
sccpServiceIsAttached
callScenario
forwardManagerCommand ( )
setCallScenaioToTheManagerCommand ( )
forwardAttachSCCPServiceCommand ( )
messageForward ( )
forwardDetachSCCPServiceCommand ( )

Figure 4.1: RANAP Interface Controller Class

Figure 4.1 shows the UML representation for the *Ranap Interface Controller* class, this class interacts with the *Traffic Handling Controller* class and the *SCCP Interface Controller* class to distribute messages between them and route the received message to the right destination. Also, this class is responsible for handling requests from the *Ranap Interface Form* class. The class's responsibilities are provided by its operations, one of these operations is the *message Forward* operation; this operation is used by the *Traffic Handling Controller* and the *SCCP Interface Controller* classes to transfer the RANAP message to the *Ranap Interface Controller* class. This class also provides the *sccp Service Is Attached* and the *call Scenario* data members which are used by some operations for the interaction diagrams. More details about these class's operations and data elements can be found in Appendix A.1.3.

RANAP Interface Form Class

The *Ranap Interface Form* Class is a *boundary* class that is used to model interaction between the RanapSim Manager Actor and the *RANAP Interface Controller* class. This Actor can only communicate with the *Ranap Interface Form* class, the class UML representation is shown in Figure 4.2.

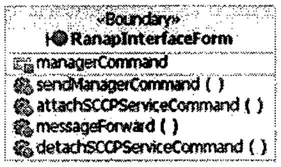


Figure 4.2: RANAP Interface Form Class

This class is responsible for handling all communications with the RanaSim Manager, so it cooperates with *Ranap Interface Controller* class to forward the Manager's commands to the proper destination. On the other hand, this class forwards all generated traffic messages to the RanapSim Manager for following up purposes. More details about these class's operations and data elements can be found in Appendix A.1.3.

4.1.2 UML Use-case Realization

In this model, we provide five sequence diagrams that realize the functional behavior for the Use Cases that represent the RANAP Controller in the Use Case Model.

These sequence diagrams explain the main interactions for this component to achieve its responsibilities, some of these sequence diagrams are; *Attach SCCP* and *Detach SCCP* sequence diagrams, these sequences work to handle the manager commands to attach or detach the SCCP Interface connection on the CPP platform which is represented as an Actor in the model, more information about these tasks will be explained in sequence diagrams in Section 4.1.2. The *Call Scenario Commands* is another sequence diagram that is provided by this model, this sequence is responsible for handling the call scenario commands that are issued by the Manager by forwarding them to *Traffic Handler Controller* class in order to initiate the corresponding call scenario for each command. More information about those sequence diagrams is given in Appendix A.2.

Next, we are going to present the sequence diagrams for the *Forward Call Scenario Messages* and the *Forward CPP Response Messages* Use Cases.

Forward Call Scenario Messages

Figure 4.3 shows the *Forward Call Scenario Messages* sequence diagram.

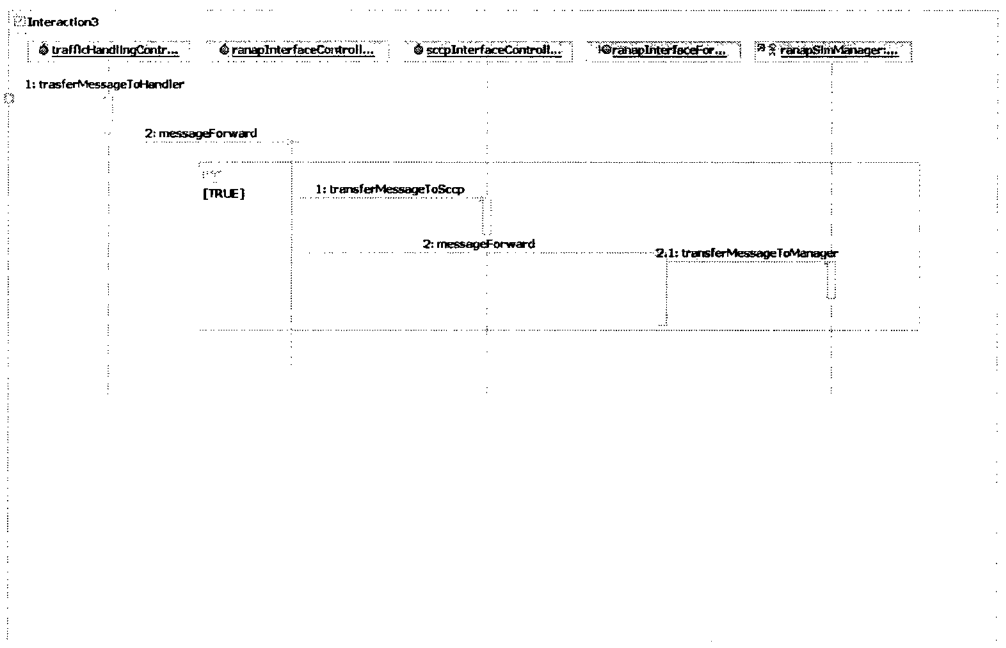


Figure 4.3: Forward Call Scenario Messages Sequence Diagram

The main objective of this sequence is to forward and transfer the generated call scenarios' messages from traffic handler controller object to other classes. The sequence is initiated in event 1 when the traffic handler controller object receives a generated message from a call scenario through the *transfer Message To Handler* operation, this message is a RANAP message that may contain a MM or CC message inside. In event 2, the message will be forwarded to the *RANAP Interface Controller* object which forwards the message in parallel to the SCCP Interface Controller object -using *transfer Message To Sccp* operation- and to the *RANAP Interface Form* object in order to forward it to the RanapSim Manager. Furthermore, the *RANAP Interface Controller* object uses the *parallel combined fragments* technique from UML 2.0 to indicate the parallel forwarding messages [4]. More detailed information about these events' operations can be found Appendix A.1.

Forward CPP Response Messages

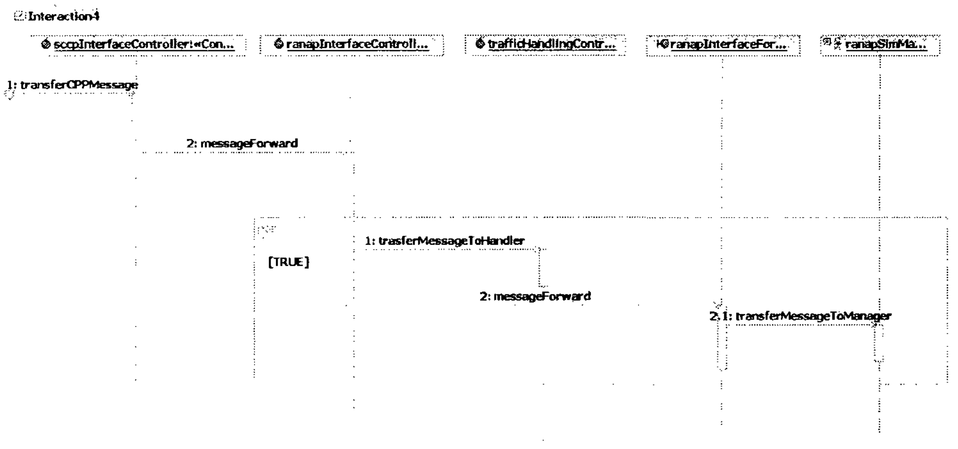


Figure 4.4: Forward CPP Response Messages Sequence Diagram

Figure 4.4 shows the *Forward CPP Response Messages* sequence diagram. This sequence is initiated in event 1 when the *SCCP Interface Controller* object receives a message from the CPP platform through the *transfer CPP Message* operation, this message is a RANAP message (it may contains MM or CC message) that comes from the MSC server through the CPP platform. In event 2, the received message will be delivered to the *Ranap Interface Controller* object which uses the *parallel combined fragment* technique to forward the message to other objects. This message will be forwarded to the *traffic handler controller* object in order to transfer it to the corresponding call scenario control class, in parallel with this, the message will be forwarded to the *Ranap Interface Form* object in order to be sent to the RanapSim Manager. More detailed information about these events' operations can be found in Appendix A.1.

4.1.3 UML Class Diagram

In this section, we illustrate the class diagram for the RANAP Controller component's functionality; Figure 4.5 shows the UML representation for this class diagram.

The class diagram shows the main control classes in the RanapSim Model, it illustrates that the *RANAP Interface Controller* class has many relationships with other several classes in the model.

The *Ranap Interface Controller* class uses the *IRanapForm* Interface that allows it to interact with the *Ranap Interface Form* class; this interface is implemented by the *Ranap Interface Form* class through the *interface realization* relationship and used through the *dependency/use* relationship [4]. On the other hand, the *Ranap Interface Form* class is able to access some operations through the *IRanapController* Interface which is implemented by the *Ranap Interface Controller* class.

Furthermore, the *Ranap Interface Controller* class uses the *ISCCP* Interface; this interface provides a set of operations that are implemented by the *SCCP Interface Controller* class, also this class is uses the *ITransferMessage* Interface which is implemented by the *Ranap Interface Controller* class. Those interfaces allow the *Ranap Interface Controller* class to cooperate with the *SCCP Interface Controller* class in order to interact with the CPP platform and consequently to interact with

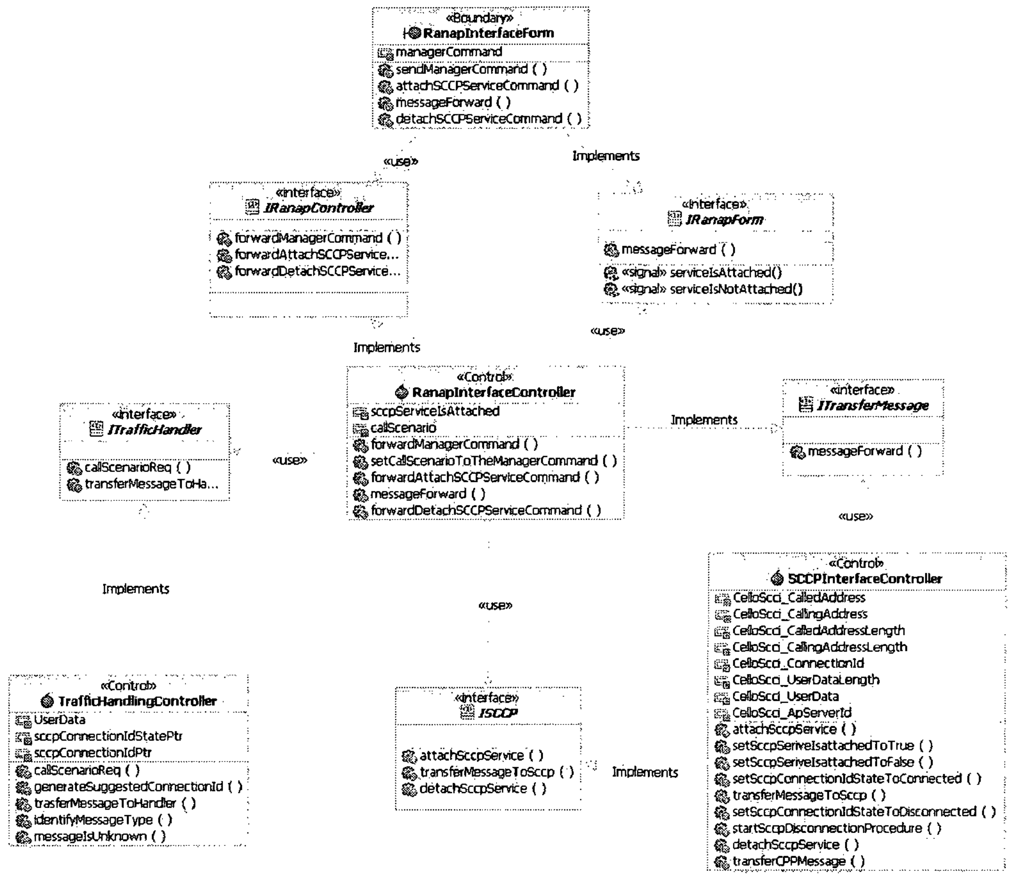

«Boundary» RanapInterfaceForm
managerCommand
sendManagerCommand ( )
attachSCCPServiceCommand ( )
messageForward ( )
detachSCCPServiceCommand ( )
«use»
Implements
«interface» IRanapController
forwardManagerCommand ( )
forwardAttachSCCPService...
forwardDetachSCCPService...
«interface» IRanapForm
messageForward ( )
«signal» serviceIsAttached()
«signal» serviceIsNotAttached()
«use»
Implements
«Control» RanapInterfaceController
sccpServiceIsAttached
callScenario
forwardManagerCommand ( )
setCallScenarioToTheManagerCommand ( )
forwardAttachSCCPServiceCommand ( )
messageForward ( )
forwardDetachSCCPServiceCommand ( )
«interface» ITrafficHandler
callScenarioReq ( )
transferMessageToHa...
«use»
Implements
«interface» ITransferMessage
messageForward ( )
«use»
Implements
«use»
«Control» SCCPInterfaceController
CelloSccl_CalledAddress
CelloSccl_CallingAddress
CelloSccl_CalledAddressLength
CelloSccl_CallingAddressLength
CelloSccl_ConnectionId
CelloSccl_UserDataLength
CelloSccl_UserData
CelloSccl_ApServerId
attachSccpService ( )
setSccpSeriveIsattachedToTrue ( )
setSccpSeriveIsattachedToFalse ( )
setSccpConnectionIdStateToConnected ( )
transferMessageToSccp ( )
setSccpConnectionIdStateToDisconnected ( )
startSccpDisconnectionProcedure ( )
detachSccpService ( )
transferCPPMessage ( )
«Control» TrafficHandlingController
UserData
sccpConnectionIdStatePtr
sccpConnectionIdPtr
callScenarioReq ( )
generateSuggestedConnectionId ( )
trasferMessageToHandler ( )
identifyMessageType ( )
messageIsUnknown ( )
«interface» ISCCP
attachSccpService ( )
transferMessageToSccp ( )
detachSccpService ( )
Implements

Figure 4.5: Control RANAP Class Diagram

the MSC server. On the other hand, the *Traffic Handling Controller* class implements the *ITrafficHandler* Interface which provides a set of operations that can be used by the *Ranap Interface Controller* class. More information about the Interfaces can be found in Appendix A.3.

4.2 SCCP Interface Controller

This section provides a detailed explanation for the *SCCP Interface Controller* component modeling. The *SCCP Interface Controller* interacts with the CPP platform through the SCCP protocol to control the call scenarios connections, also this controller exchanges messages with the CPP platform through SCCP interfaces, these messages will be routed by the CPP to the MSC server.

Several protocols provided by CPP platform to interact with, and for each protocol there are many interfaces to communicate with. As we have explained in the methodology section, we deal with the CPP platform through through the SCCP protocol using two interfaces [5], these interface are:

- The Control plane interface, also called the SCCP Access Point Facade Interface (SCCPApfi). This interface should be attached before using its service through calling a C function that goes to the CPP. CPP platform provides the *CpxSccpApAttachP* class to facilitate the communication with this interface.

- The user plane interface, also called the SCCP Interface (SCCI). This interface requires establishing a connection with the SCCP Access Point in the CPP, this connection can be called SCCP connection and each call scenario should have an established SCCP connection to start transferring data using this interface. This interface provides a connection-oriented data transfer. CPP platform provides the *CpxSccIP* class to facilitate the communication with this interface.

This component is realized by the Use Case Model through the *Control CPP Interfacing*, *Disconnect SCCP Connection*, *Setup SCCPApfi Service*, and *Release SCCPApfi Service* Use Cases. In this section, we present the *UML Analysis Model* for the SCCP Interface Controller component.

4.2.1 UML Analysis Classes

In this section, we are going to introduce the analysis classes for the SCCP Interface Controller component that cooperate with other classes in the components' interactions to provide the CPP interfacing functionality.

SCCP Interface Controller Class

The *SCCP Interface Controller* Class is the main control class of the SCCP Interface Controller component.

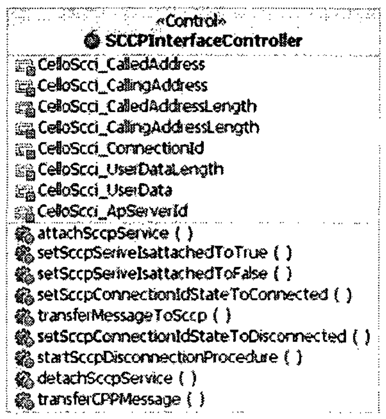

«Control»
SCCPInterfaceController
CelloScci_CalledAddress
CelloScci_CallingAddress
CelloScci_CalledAddressLength
CelloScci_CallingAddressLength
CelloScci_ConnectionId
CelloScci_UserDataLength
CelloScci_UserData
CelloScci_ApServerId
attachSccpService ( )
setSccpSeriveIsattachedToTrue ( )
setSccpSeriveIsattachedToFalse ( )
setSccpConnectionIdStateToConnected ( )
transferMessageToSccp ( )
setSccpConnectionIdStateToDisconnected ( )
startSccpDisconnectionProcedure ( )
detachSccpService ( )
transferCPPMessage ( )

Figure 4.6: SCCP Interface Controller Class

Figure 4.6 shows the UML representation for this class, it illustrates all operation and data members that are provided by the class, these data members have corresponding and similar data members in the *CpxScciP* library that is provided by the CPP platform [5].

In this class, we have the *CelloScci_CalledAddress* and *CelloScci_CallingAddress* data members that imply the destination or originating SCCP node, also the *CelloScci_ConnectionId* data member identifies the SCCP connections for several call scenarios. This class provides the *CelloScci_UserData* data member that implies the data to be transferred over the SCCP connection (which is the RANAP message in our case). The *SCCP Interface Controller* class is responsible for (1) handling all interactions with the CPP platform, (2) providing data transfer functionalities, and (3) controlling the SCCP interfaces. More details about these class's operations and data elements can be found in Appendix A.1.3.

CPP System

The *CPP System* is a *boundary* class that handles and facilitates the communication with the CPP platform. The *SCCP Interface Controller* class can only communicate with the *CPP System* class in order to interact with the CPP platform.

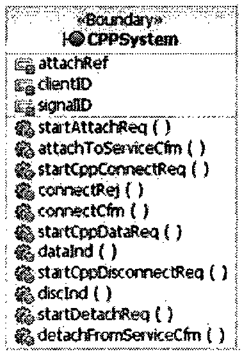


Figure 4.7: CPPSystem Class

Figure 4.7 shows the UML representation for the *CPP System* class, also it shows the class data members' *attachRef*, *clientID*, and *signalID*, all these data members have similar and corresponding in the *CpxScciApfiProxy* library that is provided by the CPP platform [5], these data members are used to attach and detach the SCCPApfi interface. This class cooperates with other classes to perform the attachment and detachment operations with the CPP, forward the SCCP messages' frames from/to the CPP, and handle the SCCP connections. More details about these class's operations and data elements can be found in Appendix A.1.3.

4.2.2 UML Use-Case Realization

In this model, we provide four sequence diagrams that realize the functional behavior for the Use Cases that represent the SCCP Interface Controller in the Use Case Model. Next, we are going to describe the sequence diagrams for the *Control CPP Interfacing*, *Setup SCCPApfi Service* Use Cases.

Setup SCCPApfi Service

In this sequence, we illustrate the interaction messages with the CPP platform to attach the SccpApfi Interface which is a prerequisite to communicate with the CPP platform. The *CpxScciApfiAttachP* object will be included to participate with the interacting objects.

Figure 4.8 shows the *Setup SCCPApfi Service* sequence diagram. This sequence starts in event 1 when the *SCCP Interface Controller* object receives a request to attach the SccpApfi interface, this request is received through the *attach Sccp Service* operation which is initiated originally by the RanapSim Manager. Afterward, the request will be forwarded to the *CPP System* object which invokes the *attach To Service Req* operation from the *CpxSccpApfiAttachP.*

Upon receiving this request, the *CpxScciApfiAttachP* object asks the CPP platform to attach the SccpApfi through the *CelloSccpApfi_ attchToServiceReq* operation [5]. Once the interface is attached properly, the *CpxScciApfiAttachP* object receives an attachment confirmation message from the CPP platform, this message allows the *CPP System* and the *SCCP Interface Controller* objects to start communication with the CPP. More detailed information about these events' operations can be found in Appendices A.1 and A.4.

Control CPP Interfacing

This sequence illustrates the basic flow that realizes the interfacing functionality with the CPP platform's interfaces. The CPP platform presents the *SCCP*

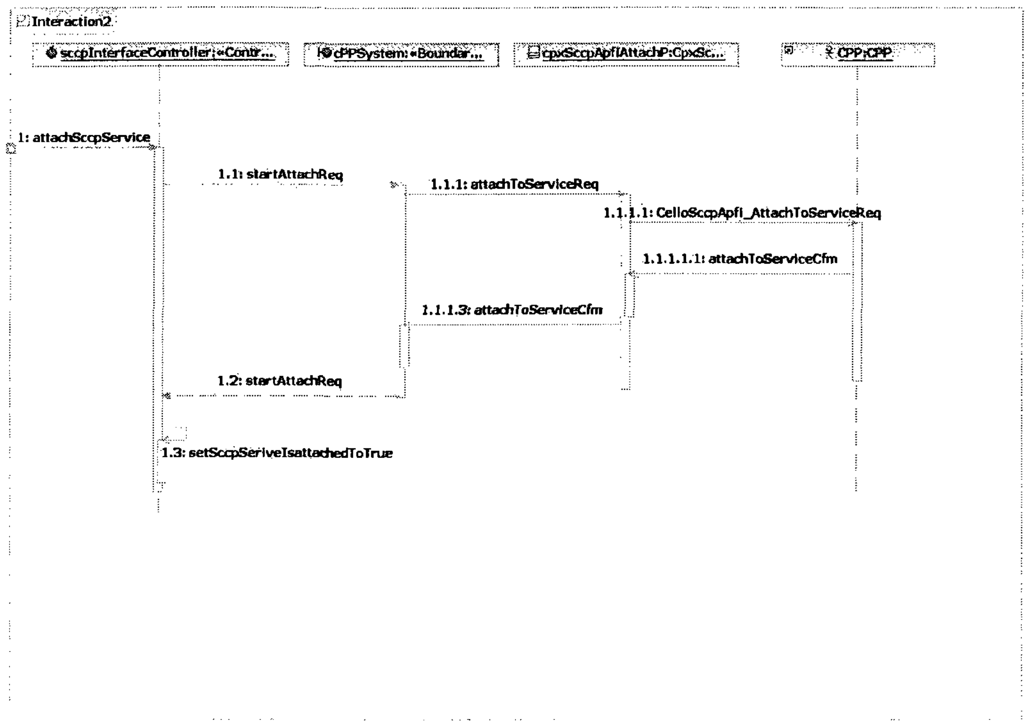

Interaction2
sccpInterfaceController:«Contr...
cPPSystem:«Boundar...
cpxSccpApfiAttachP:CpxSc...
CPP:CPP
1: attachSccpService
1.1: startAttachReq
1.1.1: attachToServiceReq
1.1.1.1: CelloSccpApfi_AttachToServiceReq
1.1.1.1.1: attachToServiceCfm
1.1.1.3: attachToServiceCfm
1.2: startAttachReq
1.3: setSccpSeriveIsattachedToTrue

Figure 4.8: Setup SCCPApfi Service Sequence Diagram

Interface (SCCI) which provides the Connection-Oriented data transfer service that is given by SCCP protocol procedures [14]. The Connection-Oriented data transfer procedure divided into phases which are; the connection establishment phase, the data transfer phase, and the connection release phase, these phases will be followed to exchange data with the CPP platform.

In this sequence, we are making use of the *SCCP Interface* (SCCI) through including the *CpxScciP* library. In addition, the *CpxScciP* library participates to facilitate interactions with the CPP.

Figure 4.9 and 4.10 present the basic flow for communicating with the CPP platform to establish an SCCP connection and transfer data to/from the CPP. The sequence starts when the *SCCP Interface Controller* is triggered by the *transfer Message To SCCP* operation, this operation transfers the *User Data* element that

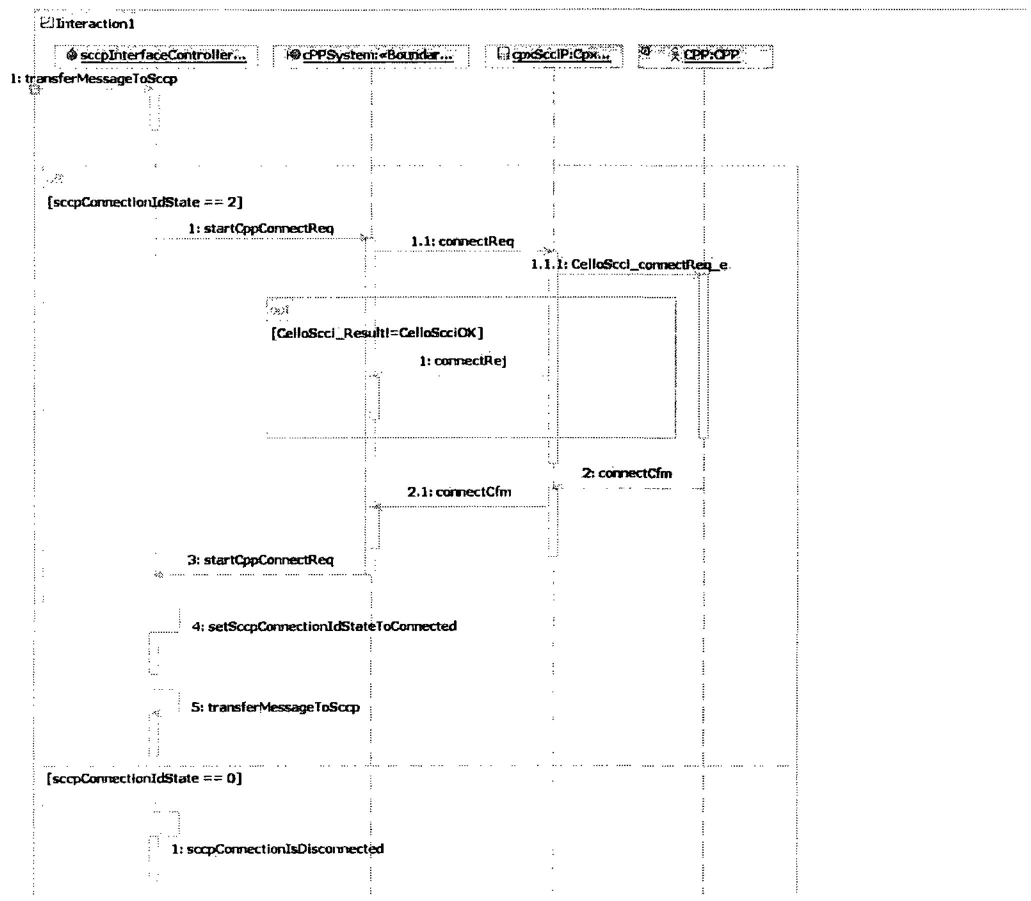

Interaction1
sccpInterfaceController...
cPPSystem:«Boundar...
cppSccIP:Cpx...
CPP:CPP
1: transferMessageToSccp
[sccpConnectionIdState == 2]
1: startCppConnectReq
1.1: connectReq
1.1.1: CelloSccI_connectReq_e
[CelloSccI_Resultl=CelloSccIOK]
1: connectRej
2: connectCfm
2.1: connectCfm
3: startCppConnectReq
4: setSccpConnectionIdStateToConnected
5: transferMessageToSccp
[sccpConnectionIdState == 0]
1: sccpConnectionIsDisconnected

Figure 4.9: Interfacing for SCCP Sequence Diagram (part1)

implies the RANAP message, the *SCCP Connection ID* that identifies the call scenario, and the *SCCP Connection ID State* that indicates the state of the connection. As explained in the diagram, the *SCCP Interface Controller* object uses *alternative combined fragments* (provided by UML 2.0) to identify which block to execute for the received message [4]. This conditional fragment divides the sequence diagram into conditional blocks, each block's condition is evaluated based on the value of the *SCCP Connection ID State* data member which has been explained in details in Section 3.1.2.

The first block evaluates if the *SCCP Connection ID State* is *generated*, this state indicates that the received message is the first message for a corresponding call scenario and the SCCP connection is not establish yet with the CPP. So, the *SCCP Interface Controller* object asks the *CPP System* to establish a connection through the *start CPP Connect Req* operation, by receiving this request; the *CPP System* invokes the *connect Request* operation from the *CpxScciP* object. By invoking the *CelloScci_connectReq_e*, the *CpxScciP* object requests the CPP to establish a connection through the *SCCI* interface for the corresponding call scenario [5]. If the SCCP connection is established, the confirmation message will be received to indicate a successful case. Once the SCCP connection is established, the *SCCP Interface Controller* object changes the *SCCP Connection ID State* to "*connected*" and invokes the *transfer Message To SCCP* operation. The second block checks whether the *SCCP Connection ID State* is "*disconnected*", if this is the case, the *SCCP Connection Is Disconnected* operation will be invoked to indicate the situation.

The third block will be taken whether the SCCP connection is established and the *SCCP Connection ID State* is "*connected*". If this is the case, then the received message will be transferred through the *start CPP data Request* operation to the *CPP System.* Once the *CPP System* receives this message, it asks the *CpxScciP* object to transfer the data to the CPP platform through the *CelloScci_dataReq_e* operation. The transferred data is the SCCP message that contains the RANAP message portion and most likely the mobility management or the call control message portion [15]. Once the CPP receives the message, it forwards it to the MSC where the processing of the message and the message response will be performed. Once the MSC process the message, it transfers its response message to the CPP platform which forwards the message to the *CPP System* through the *CpxScciP* object. By having this sequence, we achieve the connectivity functionality to the CPP and consequently to the MSC server.

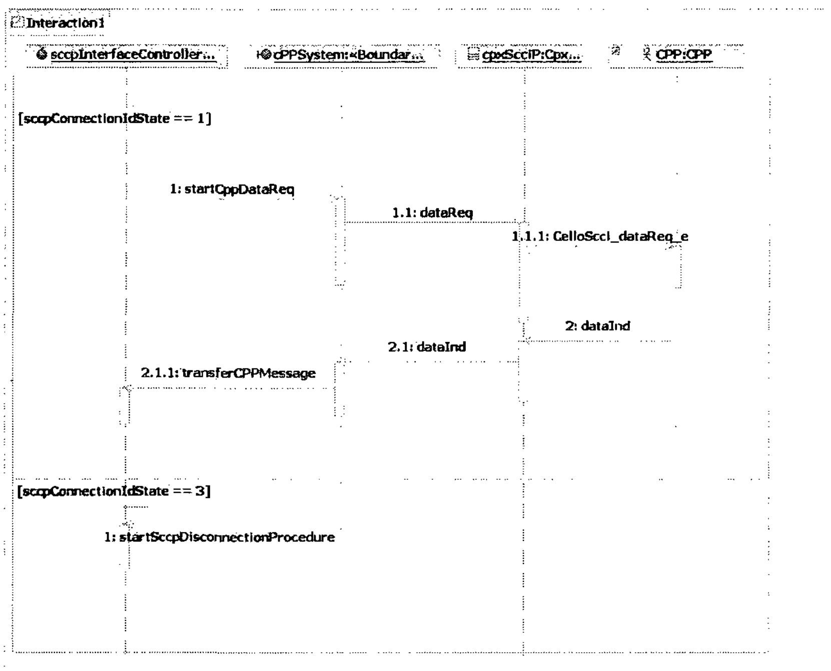

Interaction1
sccpInterfaceController...
cPPSystem:«Boundar...
cpxSccIP:Cpx...
CPP:CPP
[sccpConnectionIdState == 1]
1: startCppDataReq
1.1: dataReq
1.1.1: CelloSccI_dataReq_e
2: dataInd
2.1: dataInd
2.1.1: transferCPPMessage
[sccpConnectionIdState == 3]
1: startSccpDisconnectionProcedure

Figure 4.10: Interfacing for SCCP Sequence Diagram (part2)

The forth block checks whether the SCCP connection state is "*to be disconnected*", if this is the case, the *SCCP connection disconnection* sequence diagram shall be started for the corresponding call scenario. The "*to be disconnected*" state is originally set by the *call disconnection* sequence diagrams in the Traffic Handler component.

The *Detach SCCPApfi Service* is another sequence diagram provided by the *Detach SCCPApfi Service* Use Case, it illustrates a sequence of messages to detach the SccpApfi interface. No more SCCP services will be provided after detaching this interface. The *Disconnect SCCP connection* is another sequence diagram that works to disconnect the SCCP connection for a corresponding call scenario at the *SCCI* interface. No more SCCP services will be provided for that call scenario after that. More detailed information about these events' operations and sequence diagrams can be found in Appendices A.1, A.2, and A.4.

4.2.3 UML Class Diagram

This section describes the class diagram for the interfacing functionality with the CPP platform through the SCCP protocol. Figure 4.11 shows the UML representation for this class diagram.

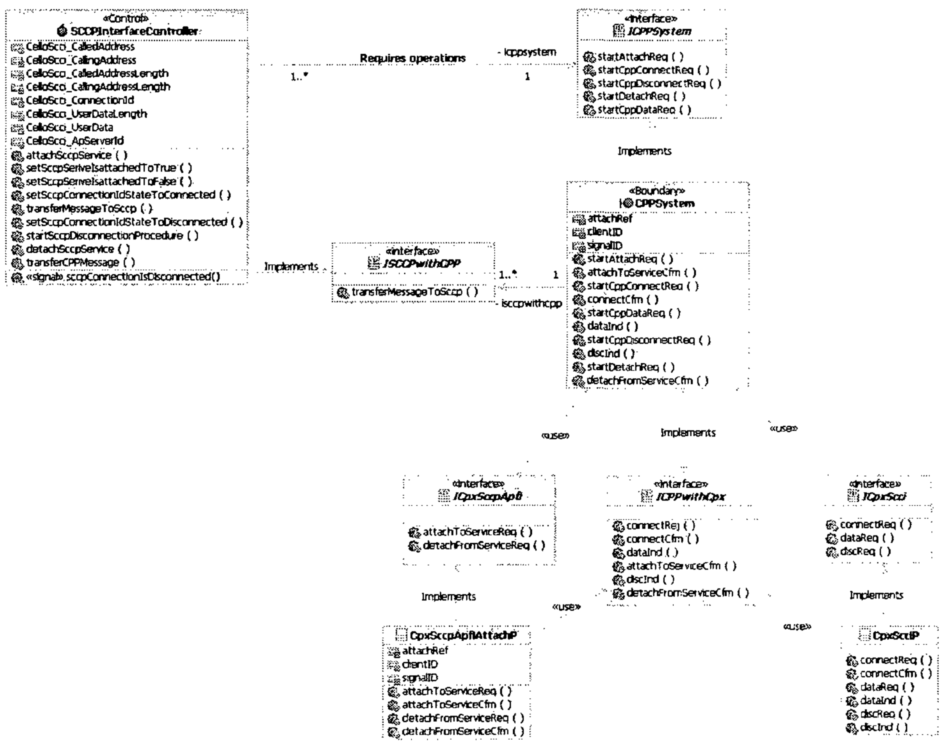


Figure 4.11: Interfacing for SCCP Class Diagram

The class diagram shows all classes that participate to communicate and interact with the CPP platform. It shows the *SCCP Interface Controller*, the *CPP System*, and the CPP library classes and how they cooperate through UML relationships. The *SCCP Interface Controller* class is associated with the *CPP System* class through the *ICPPSystem* interface, where the *SCCP Interface Controller* has an *association* relationship with (1 - 0..*) multiplicity [4]. This notation means that one object of the *CPP System* is associated with one or many objects of the *SCCP*

Interface Controller.

The *SCCP Interface Controller* class uses services from the *ICPPSystem* interface which is implemented by the *CPP System* class. In the same way, the *CPP System* uses the *ISCCPwithCPP* interface which is implemented by the *SCCP Interface Controller.* Furthermore, the *CPP System* class implements the *ICPPwithCpx* interface to provide services for the CPP library classes which are the *CpxSccpApfiAttachP* and the *CpxScciP* classes, in the same way, these classes implement the *ICpxSccpApfi* and *ICpxScci* interfaces respectively to be used by *CPP System* class in order to communicate with the CPP platform. More information about Interfaces and classes can be found in Appendices A.1, A.3, and A.4.

4.3 Summary

In this chapter, we modeled the functional behavior of the RANAP Controller and the SCCP Interface Controller components through the *UML Analysis Model.* We provided the UML logical implementation for the RANAP Controller component which behaves as the main coordinator for all interactions that occur in the RanapSim system. On the other hand, we modeled the SCCP Interface Controller component that handles the CPP platform interfacing through the SCCP protocol, as a consequence of this, the RanapSim components will be able to communicate with the MSC server.

In the *UML Analysis Model* for those two components, we identified the analysis classes and built the Use Cases' realizations that illustrate all possible interactions that reflect the functional behavior of each Use Case. In addition, we presented all class diagrams that show the structure of the provided model. The correctness of the RANAP Controller and the SCCP Interface Controller components' models has been checked through the *formal check* process at Ericsson [50].

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we presented a model for a *Traffic Generator* to load the Mobile Switching Center (MSC) by generating the control plane traffic (signalling) on the Universal Mobile Telecommunications System (UMTS), the traffic generator model can be used for the MSC load testing applications. We proposed high level analysis and design for the Radio Access Network Application Part (RANAP), the Mobility Management, the Call Control protocols for the *Circuit-Switched* network.

We have modeled the *RanapSim Traffic Generator* that provides the functional behavior of the RANAP, the MM, and the CC protocols to generate traffic on the IuCS interface between the RNC and the MSC. Most of the traffic generator models -that are used for the same purpose- provide a solution by modeling the whole UTRAN components' protocols to generate traffic towards the MSC. Based on this fact and based on our technique of modeling, we can say that the *RanapSim* model is an efficient solution to generate traffic towards the MSC server.

We provided a *UML Use Case Model* which describes the *RanapSim* functional requirements in terms of Use Cases; the Use Case Model illustrates the main specifications and the functional behavior of the proposed traffic generator. Furthermore, we provided a *UML Analysis Model* which describes the structure of the *RanapSim* system, in this model, we identified the analysis classes, illustrated the sequence diagrams, and built the class diagrams that provide the *RanapSim*'s logical implementation of the functional requirements that we identified in the Use Case Model. The RanapSim traffic generator models have been checked through a process called *formal check*, this process is used to evaluate the correctness for technical contents for any design at Ericsson.

In terms of functionalities, the *RanapSim* provides models for many call scenarios that represents the mobile station. These call scenarios are; Location Update, Mobile Originating Call, Mobile Terminating Call, and Call Disconnect. Through these call scenarios, the signaling traffic can be generated to load the MSC in order to measure the performance of the MSC; this traffic generator is not intended for verifying the actual MSC design functionality.

Furthermore, we modeled the RANAP protocol's procedures that have been used to handle the call scenarios messages and to generate the RANAP messages. Also, those procedures' models have been used to represent the radio access network signalling part of the RNC server. The RANAP procedures that have been modeled are; Paging functionality, Radio Access Bearer (RAB) Management, UE-CN direct transfer, Security Mode Control, Initial UE Message, Iu Release, as well as Common ID.

Finally, the *RanapSim Traffic Generator* has been modeled based on the CPP platform specification through the SCCP protocol. This platform provides two functionalities; the protocol transport functionality which has been utilized in this model to communicate with the MSC, and the execution functionality which can be utilized to execute the real implementation of this model.

5.2 Future Work

As a future work, the real implementation for the *RanapSim Traffic generator* models can be done by making use of the UML Use Case and Analysis Models that have been provided in this thesis. The models implementation shall present the *RanapSim Tool* that can be used for the MSC load testing. In the real implementation, traffic consists of many test cases that can be generated out of location update, mobile originating, or mobile terminating call scenarios models; by generating test cases, we can make sure that the MSC will be properly loaded. Besides, the *RanapSim Tool* should have the RanapSim Manager, Server, GUI, and CLI components implemented.

Furthermore, this traffic generator can be modeled using the executable UML, the resulting model using this language is composed by a set of modeling elements which are; *domain chart*, *class diagram*, *action language*, and *statechart diagram* [4].

In addition, as a future work, more mobile call scenarios can be modeled using the call control and the mobility management protocols' models; the Call Forwarding and Call Waiting are examples of other mobile call scenarios that can be modeled and integrated easily within the *RanapSim* system.

The traffic generation models that are proposed in this thesis handle the traffic of a Radio Network Controller (RNC) that is communicating with one MSC server. As a future work, the traffic distribution in a multi-processor application can be modeled to generate traffic similar to multiple RNCs traffic to have extremely high traffic load generation.

Finally, as a future work, it is possible to reuse some of the modeled components to model a traffic generator to load the Service GPRS Support Node (SGSN). Generating SGSN call scenarios will require models for the GPRS Mobility Management protocol and the GPRS Session Management protocol; these protocols will be carried through the RANAP protocol and the call connections can be handled through the SCCP protocols. Based on that, many components from the RanapSim model can be reused for this purpose such as; the SCCP Interface Controller and the RANAP Controller.

Appendix A

A.1 UML Analysis Classes

The UML Analysis Classes for the *RanapSim* Model are illustrated in this section, the following are the analysis classes with their operations and data elements for the messaging proxies' classes:

A.1.1 Messaging Proxies Classes

RanapMessageProxy

Figure A.1 shows the UML representation for this class.

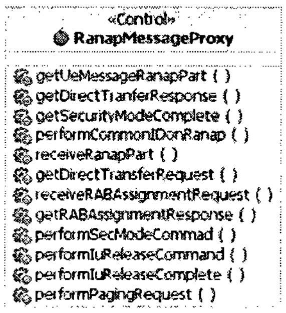

«Control»
RanapMessageProxy
getUeMessageRanapPart ( )
getDirectTranferResponse ( )
getSecurityModeComplete ( )
performCommonIDonRanap ( )
receiveRanapPart ( )
getDirectTransferRequest ( )
receiveRABAssignmentRequest ( )
getRABAssignmentResponse ( )
performSecModeCommad ( )
performIuReleaseCommand ( )
performIuReleaseComplete ( )
performPagingRequest ( )

Figure A.1: RANAP Message Proxy Class

operations for *RanapMessageProxy*:

RanapMessageProxy::getUeMessageRanapPart

Operation Parameters: *requestType*; specifies which call scenario is required. The requestType could be location update request, CM service request, or paging response.

Possible return messages: *InitialUeMessage*

The purpose of the Initial UE Message procedure is to establish an Iu signaling connection between the MSC domain and the RNC -for specific UE- and to transfer the initial NAS-PDU message to the MSC. The *InitialUEMessageController* object will call this operation to get an Initial UE message for certain call scenario specified in the *requestType*. Based on the *requestType*, this operation will invoke an appropriate operation in the MM proxy to get the proper NAS-PDU message. Upon receiving the NAS-PDU message (MM), this operation will initiate the *InitialUeMessage* (RANAP) by adding other data fields such as; LAI, SAI, and Iu signaling connection Identifier.

RanapMessageProxy::getDirectTranferResponse

Operation Parameters: UserData; points to the RANAP actual message.

Possible return messages: *DirectTranferMessage*

The *DirectTransferController* object will call this operation to get the response of the direct transfer request which will usually carry the NAS-PDU message (e.g., CC or MM). The purpose of the Direct Transfer procedure is to carry UE-MSC signalling messages over the Iu Interface. The UE-MSC signalling messages are not interpreted by the RNC. The UE-MSC signalling messages are transported as a parameter in the Direct Transfer messages. The UE side is represented by the MM message proxy or the CC message proxy.

RanapMessageProxy::getDirectTransferRequest

Operation Parameters: None

Possible return messages: *DirectTransferMessage*

The *DirectTransferController* object will call this operation to create direct transfer message request that will carry a CC message to start a call control procedure such as; call setup, call connect, or call alerting. *typeOfCcService* parameter will be passed -through this operation- to the getCcPart operation.

RanapMessageProxy::performSecModeCommad

Operation Parameters: UserData; points to the RANAP actual message.

Possible return messages: *SecurityModeCompleteMessage*

The *SecurityModeController* object will call this operation. This operation carries the Security Mode Command to the RANAP Message Proxy to decode the message and generate the appropriate response. For more information, see the *getSecurityModeComplete* operation described next.

RanapMessageProxy::getSecurityModeComplete

Operation Parameters: UserData; points to the RANAP actual message.

Possible return messages: *SecurityModeCompleteMessage*

This operation will be called by the class's object itself to perform the security mode complete procedure which is a RANAP procedure. This operation responds to the *SecurityModeCommandMessage* by extracting the Encryption Information IE and the Integrity Protection Information IE, then choosing appropriate ciphering and integrity alternative algorithms. When the integrity and the ciphering configuration are successfully chosen for the radio interface procedure, the object of this class shall return a SECURITY MODE COMPLETE message to the caller in order to send it to the MSC.

RanapMessageProxy::performCommonIDonRanap

Operation Parameters: UserData; points to the RANAP actual message.
Possible return messages: None

The *CommonIDController* object will call this operation to forward the Common ID message to the RANAP Proxy object, and to perform the Common ID procedure there. In real life, the purpose of the Common ID procedure is to inform the RNC about the permanent NAS UE Identity (i.e., IMSI) of a user. This is used by the RNC to create a reference between the permanent NAS UE identity of the user and the RRC connection of that user for RNC paging procedure. This operation will save the received "IMSI" in the *CommonIDMessage* object, in order to use it for other operations for that UE or user. We call this operation of attaching the "IMSI" to the UE, Location Update - IMSI attach.

RanapMessageProxy::receiveRanapPart

Operation Parameters: *UserData*; points to the RANAP actual message.
Possible return messages: None
The *DirectTransferController* object will call this operation to forward the received RANAP message to the RANAP proxy only, without expecting any return message.

RanapMessageProxy::receiveRABAssignmentRequest

Operation Parameters: *UserData*; points to the RANAP actual message.
Possible return messages: *RABAssignmentRespMessage*
The *RABAssignmentController* object will call this operation to handle this request. This operation will check the *messageType* to recognize the message. Based on the message type it call an appropriate operation to perform the RAB Assignment procedure and generate the response. Also, this request contains the list of RABs to be established or modified. In this case the *getRABAssignmentResponse* operation will be called.

RanapMessageProxy::getRABAssignmentResponse

Operation Parameters: *UserData*; points to the RANAP actual message.
Possible return messages: *RABAssignmentRespMessage*
The *receiveRABAssignmentRequest* operation will call this operation, which will forward the RAB list to be established or modified. This operation will generate the *RABAssignmentRespMessage* -which will confirm the RAB request that comes from the MSC- and send the established/modified RAB ID to the MSC. This operation will simulate that the RABs are established by the RNC for a given UE; this operation will be understood by the MSC.

RanapMessageProxy::performPagingRequest

Operation Parameters: UserData; points to the RANAP actual message.
Possible return messages: TRUE when the paging request received successfully, or FALSE if not.
The purpose of the Paging procedure is to enable the MSC to request the RNC to contact that UE. Normally, the MSC will initiate the procedure by sending *pagingRequestMessage*. The paging message shall contain various IEs such as; IMSI,

TMSI, paging area, and others. Based on the message received, the RNC will send broadcast or uni-cast message to find the needed UE. This operation will receive the paging request, save some of the IEs that could be needed in the paging response message, and return "True" to the operation initiator to initiate the Paging response message. In this case, the initiator will be the *CallTerminatingController* object. The Paging request is a RANAP message, but the Paging response is an MM message.

RanapMessageProxy::performIuReleaseCommand

Operation Parameters: UserData; points to the RANAP actual message.
Possible return messages: *IuReleaseCompleteMessage*
For more details, see next operation.

RanapMessageProxy::performIuReleaseComplete

Operation Parameters: UserData; points to the RANAP actual message.
Possible return messages: *IuReleaseCompleteMessage*
The purpose of the Iu Release procedure is to enable the MSC to release an Iu connection and all RNC resources related only to that Iu connection. The Iu Release procedure will be initiated when the transaction between the UE and the MSC get completed. Also, the Iu Release procedure is initiated for other reasons, not important for our system. The *IuReleaseController* object will invoke the *performIuReleaseCommand* operation. This operation will carry the *IuReleaseCommandMessage* to the RANAP Message Proxy to release the Iu signaling connection for the corresponding UE. While invoking this operation, the *performIuReleaseComplete* will be invoked to perform the release and generate an appropriate response to send back to

the initiator. The *performIuReleaseCommand* will get the *IuReleaseCompleteMessage* and send it to its initiator in order to send that to the MSC.

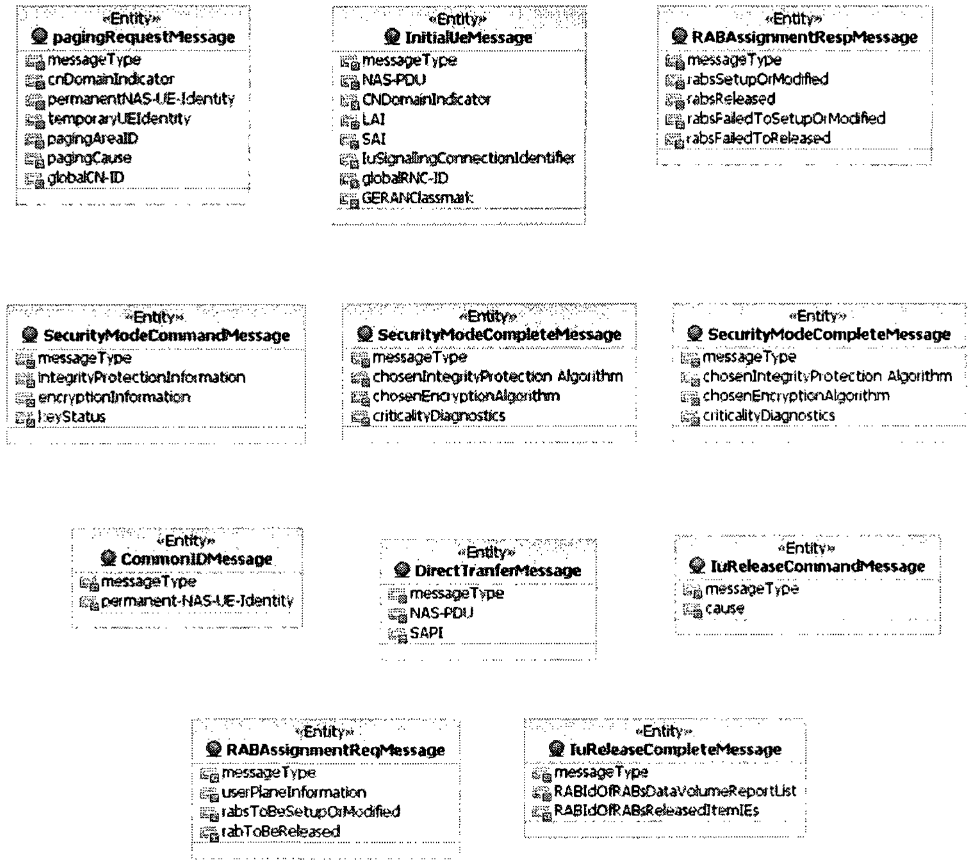


Figure A.2: Ranap Message Proxy's Entity Classes

Figure A.2 shows the Entity classes located in the *RanapMessageProxy* class.

MobilityManagmentMessageProxy

This class works as an agent or proxy for receiving, decoding, preparing, and forwarding various Mobility Management (MM) messages. Figure A.3 shows the UML representation for this class.

Class Data Members: T3230; this is a timer which is started by the Mobility Management Message Proxy when the CmServiceReqMessage is sent, and stopped when the CmServiceAcceptMessage or the CmServiceRejMessage is received.

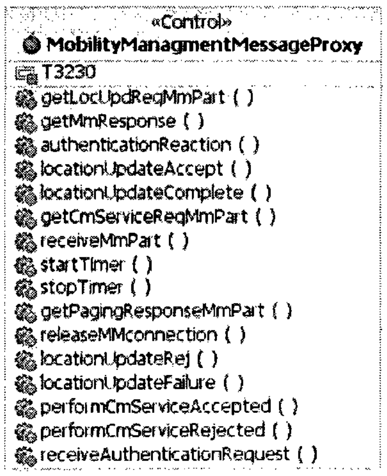

«Control»
MobilityManagmentMessageProxy
T3230
getLocUpdReqMmPart ( )
getMmResponse ( )
authenticationReaction ( )
locationUpdateAccept ( )
locationUpdateComplete ( )
getCmServiceReqMmPart ( )
receiveMmPart ( )
startTimer ( )
stopTimer ( )
getPagingResponseMmPart ( )
releaseMMconnection ( )
locationUpdateRej ( )
locationUpdateFailure ( )
performCmServiceAccepted ( )
performCmServiceRejected ( )
receiveAuthenticationRequest ( )

Figure A.3: Mobility Management Message Proxy Class

operations:

MobilityManagmentMessageProxy::getLocUpdReqMmPart

Operation Parameters: NAS-PDU; points to the Call Control or the Mobility Management actual message.

Possible return messages: LocationUpdateReqMessage

The RanapMessageProxy object will call this operation. The normal location updating procedure is used to update the registration of the actual Location Area of a mobile station in the network. The location updating type information element in the LocationUpdateReqMessage shall indicate the IMSI attach. This operation will be responsible for assigning values to the IEs in the LocationUpdateReqMessage such as; the locationUpdatingRequestMessageType, LAI, protocolDiscriminator, and others. The return message from this operation will be sent through the Initial UE Message to the MSC to start the location update procedure.

MobilityManagmentMessageProxy::getMmResponse

Operation Parameters: NAS-PDU; points to the Call Control or the Mobility Management actual message.

Possible return messages: Mobility Management message response

The RanapMessageProxy object will call this operation. This operation will carry the MM message (request), invoke an appropriate operation to get response, and return the response back to the initiator. In this case, the getMmResponse will carry the authenticationReqMessage -coming from the MSC- and will return back either the authenticationResMessage or the authenticationRejMessage.

MobilityManagmentMessageProxy::authenticationReaction

Operation Parameters: NAS-PDU; points to the Call Control or the Mobility Management actual message.

Possible return messages: authenticationResMessage or authenticationRejMessage.

The MM message proxy object will recognize the authenticationReqMessage by

checking the messageType, after that it will invoke the authenticationReaction to accept or reject the authentication procedure. The purpose of the authenticationReaction operation is to permit the network to check whether the identity provided by the mobile station is acceptable or not. Also, this will allow the network to provide parameters enabling the mobile station to calculate a new UMTS ciphering and integrity keys. Furthermore, this operation permits the mobile station to authenticate the network.
After calculating the ciphering integrity keys properly, the authenticationReaction will return back the authenticationResMessage to indicate the acceptance or the authenticationRejMessage to indicate the rejection. Since we are concentrating on traffic generation, we will indicate acceptance all the time unless the case requires rejection.

MobilityManagmentMessageProxy::receiveLocationUpdateAccept

Operation Parameters: NAS-PDU; points to the Call Control or the Mobility Management actual message.
Possible return messages: None

For more details, see next operation.

MobilityManagmentMessageProxy::receiveLocationUpdateRej

Operation Parameters: NAS-PDU; points to the Call Control or the Mobility Management actual message.
Possible return messages: None
The RanapMessageProxy will forward the MM message to the MobilityManagmentMessageProxy object. This object will check the value of the messageType as follows:

If the messageType = "xx00 0010"

This means that the MM message is LocationUpdateAcceptMessage. So, the receiveLocationUpdateAccept operation will be invoked to indicate that the IMSI is activated in the network and to store the received location area identification (LAI).

If the messageType = "xx00 0100"

This means that the MM message is LocationUpdateRejMessage. So, the receiveLocationUpdateRej operation will be invoked to indicate that the IMSI is not activated in the network and to store the received rejectCause.

MobilityManagmentMessageProxy::getCmServiceReqMmPart

Operation Parameters: None

Possible return messages: CmServiceReqMessage

The RanapMessageProxy object will call this operation. This operation will generate the CmServiceReqMessage and return it back to the initiator. This message will be carried by the Initial UE Message (RANAP Message) in order to establish MM connection in the MSC side. The CmServiceReqMessage is the first message in the call originating scenario. While the execution of this operation, the timers T3230 (MM timer) and T303 (CC timer) shall be started.

MobilityManagmentMessageProxy::performCmServiceAccepted

Operation Parameters: NAS-PDU; points to the Call Control or the Mobility Management actual message.

Possible return messages: None

For more details, see next operation.

MobilityManagmentMessageProxy::performCmServiceRejected

Operation Parameters: NAS-PDU; points to the Call Control or the Mobility Management actual message.

Possible return messages: None

The RanapMessageProxy will forward the MM message to the MobilityManagmentMessageProxy object. This object will check the value of the messageType as follows:

If the messageType = "xx10 0001"

This means that the MM message is CmServiceAcceptMessage. So, the performCmServiceAccepted operation will be invoked to indicate that the CM service request is accepted and the MM connection has been established. Timer T3230 shall be stopped.

If the messageType = "xx10 0010"

This means that the MM message is CmServiceRejMessage. So, the performCmServiceRejected operation will be invoked to indicate that the CM service request can not be accepted. Timer T3230 shall be stopped.

MobilityManagmentMessageProxy::receiveMmPart

Operation Parameters: NAS-PDU; points to the Call Control or the Mobility Management actual message.

Possible return messages: None

The RanapMessageProxy object will call this operation. This operation will carry the MM message, and invoke an appropriate operation to receive the MM message without returning any response back to the initiator. The receiveMmPart will carry various messages such as; CmServiceAcceptMessage, CmServiceRejMessage, LocationUpdateAcceptMessage, or LocationUpdateRejMessage.

MobilityManagmentMessageProxy::getPagingResponseMmPart

Operation Parameters: UserData; points to the RANAP actual message.
Possible return messages: PagingResponseMessage
At reception of a pagingRequestMessage (RANAP message) from the MSC, the RanapMessageProxy will invoke this operation from the MM message proxy. The getPagingResponseMmPart will create a PagingResponseMessage (MM message) as a response to the paging request message. The PagingResponseMessage shall be sent to the MSC to indicate that the required UE is located, the paging procedure is completed, and the MSC can start contact that UE. In our model, this UE represent the MS call terminating entity. The PagingResponseMessage will be the first message sent from the MS call terminating entity.

MobilityManagmentMessageProxy::releaseMMconnection

Operation Parameters: None
Possible return messages: None
When the CC proxy receives a ReleaseMessage, it will also ask the MM proxy -through releaseMMconnection operation- to release the Mobility management resources to complete the release procedure at the UE side.

MobilityManagmentMessageProxy::startTimer

Operation Parameters: timerId; indicates the timer identity
Possible return messages: Boolean
This operation shall be invoked when the MM message proxy requires to start a timer which belongs to the Mobility Management procedures.

MobilityManagmentMessageProxy::stopTimer

Operation Parameters: timerId; indicates the timer identity

Possible return messages: Boolean

This operation shall be invoked when the MM message proxy requires to stop a timer which belongs to the Mobility Management procedures.

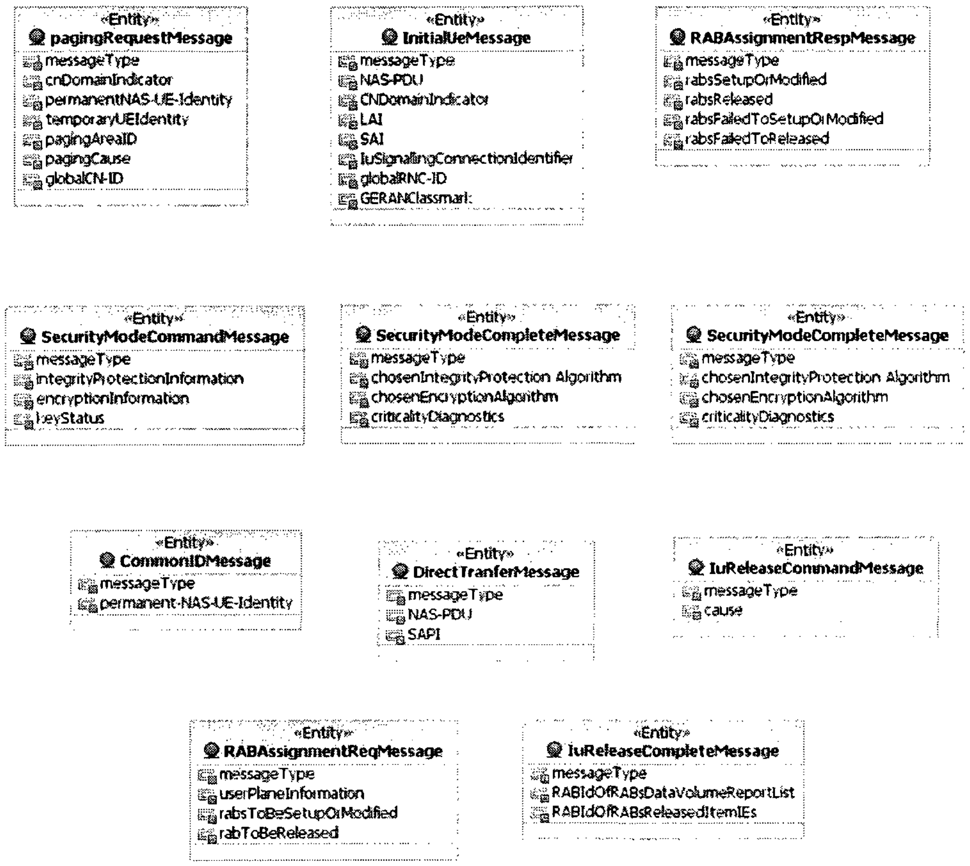


Figure A.4: Mobility Managment Message Proxy's Entity Classes

Figure A.4 shows the Entity classes located in the *MobilityManagmentMessageProxy* class.

CallControlMessageProxy

This class works as an agent or proxy for receiving, decoding, preparing, and forwarding various Call Control (CC) messages. Figure A.5 shows the UML representation for this class.

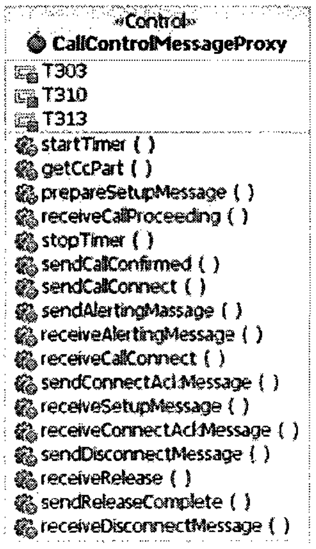


Figure A.5: Call Control Message Proxy Class

Class Data Members

T303: this is a timer which is started by the Call Control Message Proxy (originating side) when the CmServiceReqMessage is sent to establish a mobile originating MM connection, and stopped when the CallProceedingMessage is received.

T310: this is a timer which is started by the Call Control Message Proxy (originating side) when the CallProceedingMessage is received at the MS call originating entity, and stopped after the MS call originating entity receives a CallConnectMessage and send ConnectAckMessage.

T313: this is a timer which is started by the Call Control Message Proxy (terminating side) when the CallConnectMessage is sent from the MS call terminating entity, and stopped after the MS call terminating entity receives a ConnectAckMessage.

operations for CallControlMessageProxy:

CallControlMessageProxy::getCcPart

Operation Parameters: NAS-PDU; points to the Call Control or the Mobility Management actual message. typeOfCcService; indicates type of service that the CC proxy should provide.
Possible return messages: Call Control message response
The RanapMessageProxy object will call this operation. This operation carries -as a parameter- the CC message or/and typeOfCcService, to indicate the CC proxy which operation need to be performed, and return the response back to the initiator -if applicable-. Based on the parameters of this operation (CC message and typeOfCcService), and based on which entity this operation shall be performed (originating or terminating entity), the CC message proxy will perform the proper operation.

CallControlMessageProxy::prepareSetupMessage

Operation Parameters: None
Possible return messages: SetupMessage
The CallControlMessageProxy object (at the MS call originating entity) will call this operation. This operation will prepare the SetupMessage which shall contain all the information required by the MSC to process the call. In particular, the SetupMessage shall contain the calling and the called party addresses information.

Also, the SetupMessage initiates a mobile originating call establishment.

CallControlMessageProxy::receiveSetupMessage

Operation Parameters: NAS-PDU; points to the Call Control or the Mobility Management actual message.

Possible return messages: None

Upon completion of the MM connection (for the terminating side), the call control entity of the MSC shall send a SetupMessage to its peer entity (CC) at the MS call terminating entity. The CallControlMessageProxy object (at the MS call terminating entity) calls this operation. This operation will receive the SetupMessage and will indicate the successful compatibility checking. This message is sent by the MSC to the MS call terminating entity to initiate a mobile terminated call establishment.

CallControlMessageProxy::sendCallConfirmed

Operation Parameters: None

Possible return messages: CallConfirmedMessage

As an acknowledgment of successfully receiving the SetupMessage, the CallControlMessageProxy (at the MS call terminating entity) prepares and sends the CallConfirmedMessage by invoking this operation. All this happens to indicate that the incoming call request for the MSC has been confirmed.

CallControlMessageProxy::receiveCallProceeding

Operation Parameters: NAS-PDU; points to the Call Control or the Mobility Management actual message.

Possible return messages: None

Once the MSC receives a CallConfirmedMessage from the MS terminating entity, the call control entity of the MSC will send a CallProceedingMessage to the MS originating entity. The CallControlMessageProxy object (at the MS call originating entity) calls this operation. Through this operation, the CallProceedingMessage shall be received to indicate that the requested call establishment information has been received at the terminating side.

CallControlMessageProxy::sendAlertingMassage

Operation Parameters: None
Possible return messages: AlertingMessage
Upon completion of the RAB Assignment Procedure at the MS terminating entity, the CallControlMessageProxy object -in this entity- will prepare and send the AlertingMessage by invoking this operation. This message will be sent to the call control entity at the MSC to indicate that the alerting procedure has been initiated at the MS terminating entity.

CallControlMessageProxy::receiveAlertingMessage

Operation Parameters: NAS-PDU; points to the Call Control or the Mobility Management actual message.
Possible return messages: None
Once the MSC receives an AlertingMessage from the MS terminating entity, the call control entity of the MSC will send a corresponding AlertingMessage to the MS originating entity. The CallControlMessageProxy object (at the MS call originating entity) invokes this operation. Through this operation, the AlertingMessage shall be received to indicate that the alerting procedure has been initiated at the MS call terminating entity.

CallControlMessageProxy::sendCallConnect

Operation Parameters: None

Possible return messages: CallConnectMessage

The CallControlMessageProxy object -at the MS call terminating entity- prepares and sends the CallConnectMessage by invoking this operation. This message will be sent to the call control entity at the MSC to indicate that the call has been accepted at the called entity.

CallControlMessageProxy::receiveCallConnect

Operation Parameters: NAS-PDU; points to the Call Control or the Mobility Management actual message.

Possible return messages: None

Once the MSC receives a CallConnectMessage from the MS terminating entity, the call control entity of the MSC will send a corresponding CallConnectMessage to the MS originating entity. The CallControlMessageProxy object (at the MS call originating entity) invokes this operation. This operation receives the CallConnectMessage which indicates.

CallControlMessageProxy::sendConnectAckMessage

Operation Parameters: None

Possible return messages: ConnectAckMessage

The CallControlMessageProxy object -at the MS call originating entity- shall, upon receiving a CallConnectMessage, attach the user connection. Also it will prepare and send the ConnectAckMessage by invoking this operation. This message shall be sent to the call control entity at the MSC to acknowledge the offered connection.

CallControlMessageProxy::receiveConnectAckMessage

Operation Parameters: NAS-PDU; points to the Call Control or the Mobility Management actual message.
Possible return messages: None
Once the MSC receives a ConnectAckMessage from the MS originating entity, the call control entity of the MSC will send a corresponding ConnectAckMessage to the MS terminating entity. The CallControlMessageProxy object (at the MS call terminating entity) invokes this operation. This operation receives the ConnectAckMessage which indicates that the MS terminating entity has been awarded the call.

CallControlMessageProxy::sendDisconnectMessage

Operation Parameters: None
Possible return messages: DisconnectMessage
Upon receiving a Call Disconnect command from the system Manager, the CallControlMessageProxy object -at the originating or terminating entity- shall stop all the running timers in the corresponding entity and send DisconnectMessage to request the MSC to clear an end-to-end call connection by invoking this operation. The DisconnectMessage contains a "cause" information element (IE) which indicates the disconnection cause at the MSC side.

CallControlMessageProxy::receiveDisconnectMessage

Operation Parameters: NAS-PDU; points to the Call Control or the Mobility

Management actual message.

Possible return messages: None

Upon the MSC receive a DisconnectMessage from any entity in the end-to-end connection, it will send a corresponding DisconnectMessage to the other connected entity. The CallControlMessageProxy object (at this connected entity) will call this operation to receive the DisconnectMessage; this message indicates that the end-to-end connection has been cleared. The cause of the disconnection can be found in the "cause" IE.

CallControlMessageProxy::receiveRelease

Operation Parameters: NAS-PDU; points to the Call Control or the Mobility Management actual message.

Possible return messages: None

Once the MSC receives a DisconnectMessage from any entity in the end-to-end connection, it will send a ReleaseMessage to the connected entities. The CallControlMessageProxy object (at the originating and terminating entity) will call this operation to receive the ReleaseMessage which indicates that the MSC intends to release the transaction identifier and that the receiving entities shall release the transaction identifier, and stop all running timers. The transaction identifier is used for protocol error handling (see reference [4]).

CallControlMessageProxy::sendReleaseComplete

Operation Parameters: None

Possible return messages: ReleaseCompleteMessage

Upon a receipt of a DisconnectMessage from the MSC, the CallControlMessage-Proxy object -at the originating or terminating entity- shall stop all running timers

-which corresponds to a specific call- and send a ReleaseCompleteMessage to indicate that the originating or terminating entity has released the transaction identifier and that the MSC shall release the transaction identifier.

CallControlMessageProxy::startTimer

Operation Parameters: timerId; indicates the timer identity
Possible return messages: Boolean
This operation shall be invoked when the CC message proxy requires to start a timer which belongs to the Call Control procedures.

CallControlMessageProxy::stopTimer

Operation Parameters: timerId; indicates the timer identity. If the operation parameter is not provided, the operation will stop all running timers for the corresponding entity.
Possible return messages: Boolean
This operation shall be invoked when the CC message proxy requires stopping a timer belongs to the Call Control procedures.

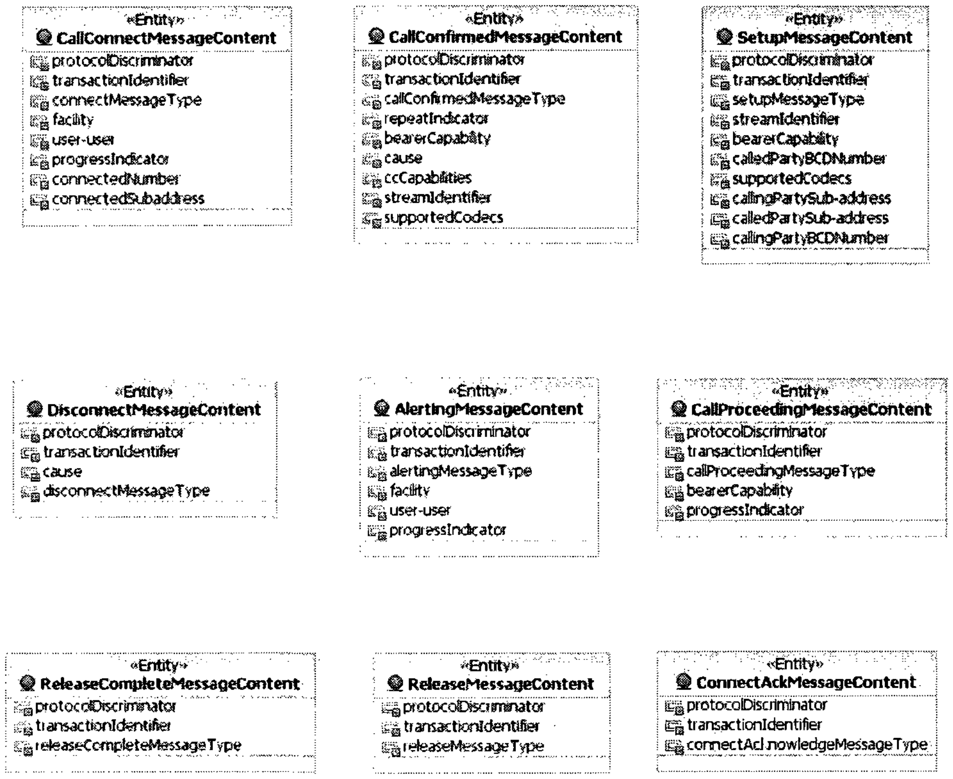


Figure A.6: Call Control Message Proxy's Entity Classes

Figure A.6 shows the Entity classes located in the *CallControlMessageProxy* class.

A.1.2 Traffic Handler Control Classes

The following classes illustrate the *control* Analysis classes for the *Traffic Handler Component*:

TrafficHandlingController

Figure A.7 shows the UML representation for this class.

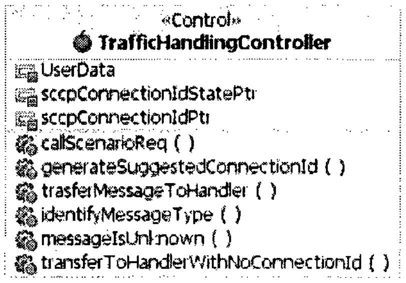

«Control»
TrafficHandlingController
UserData
sccpConnectionIdStatePtr
sccpConnectionIdPtr
callScenarioReq ( )
generateSuggestedConnectionId ( )
trasferMessageToHandler ( )
identifyMessageType ( )
messageIsUnknown ( )
transferToHandlerWithNoConnectionId ( )

Figure A.7: Traffic Handling Controller Class

Class Data Members: UserData; points to the RANAP actual message. sccpConnectionIdPtr; points to the CelloScci_ConnectionId data member in the SCCPInterfaceController class. sccpConnectionIdStatePtr; points to the sccpConnectionIdState enumeration data type in the SCCPInterfaceController class.

operations:

TrafficHandlingController::callScenarioReq

Operation Parameters: callScenario; indicates the required call scenario to be performed.

Possible return value: Boolean

This operation shall be invoked by the RanapIntController to forward the call scenario command requested by the manager to the Traffic Handler. Upon a receipt of this callScenario, the trafficHandlerController object shall go through the switch condition to forward the call scenario request to the proper call scenario entity.

TrafficHandlingController::generateSuggestedConnectionId

Operation Parameters: None
Possible return values: ConnectionIdPtr, sccpConnectionIdStatePtr
Once the switch condition evaluated by the trafficHandlerController object, it shall invoke this operation to generate suggested Connection ID to be used for the SCCP connection. Also, this operation saves the generated connection ID at the trafficHandlerController for a corresponding call scenario. For the corresponding saved Connection ID, a value of '2' will be assigned to the enumeration data type (sccpConnectionIdState); this value indicates that the Connection ID state is only "generated". This operation will be invoked when a location update, MS call originating, or MS call terminating scenario is recognized by the TrafficHandlerController.

TrafficHandlingController::trasferMessageToHandler

Operation Parameters: CelloScci_ConnectionId; holds the SCCP connection ID. CelloScci_UserData; holds the RANAP actual message. sccpConnectionIdState; holds the SCCP connection ID state.
Possible return values: None
This operation shall be invoked by the RanapIntController to forward the RANAP message, the SCCP connection ID, and the SCCP connection ID state to the Traffic Handler. All of these data elements will be carried by the Operation Parameters.

TrafficHandlingController::identifyMessageType

Operation Parameters: CelloScci_UserData; holds the RANAP actual message.
Possible return values: messageType
In the "Transfer to Handler" sequence diagram, the RANAP message will be received at the trafficHandlerController through the trasferMessageToHandler operation. Based on the connection ID, the switch condition will be evaluated to recognize the destination to forward the message. In this switch condition, if the message is received with undefined Connection ID, then there will be a high probability for this message to be a "Paging request" asking for the MS terminating entity. To perform that, this operation identifies the messageType data member of the message.
If the messageType = "0014", this means a Paging request. In this case, the generateSuggestedConnectionId operation will be invoked to generate a new connection ID to be attached with the message which will be forwarded to the MS terminating entity.
If not and the message is unknown, the messageIsUnknown operation will be invoked to indicate that.

TrafficHandlingController::messageIsUnknown

Operation Parameters: None
Possible return values: None
See the identifyMessageType operation for explanations.

LocationUpdateController

Figure A.8 shows the UML representation for this class.
Class Data Members: UserData; points to the RANAP actual message. NAS-PDU;

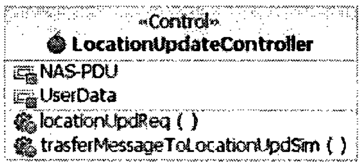


Figure A.8: Location Update Controller Class

points to the Call Control or the Mobility Management actual message.
operations:

LocationUpdateController::locationUpdReq

Operation Parameters: None
Possible return values: None

This operation shall be invoked by the trafficHandlerController to ask the LocationUpdateController to start the Location Update call scenario.

LocationUpdateController::trasferMessageToLocationUpdSim

Operation Parameters: CelloScci_ConnectionId; holds the SCCP connection ID. CelloScci_UserData; holds the RANAP actual message.
Possible return values: None
This operation shall be invoked by the trafficHandlerController to forward the RANAP message (CelloScci_UserData) along with the SCCP connection ID (CelloScci_ConnectionId) to the LocationUpdateController class. The RANAP message originally comes from the MSC through the CPP platform.

CallOriginatingController

Figure A.9 shows the UML representation for this class.

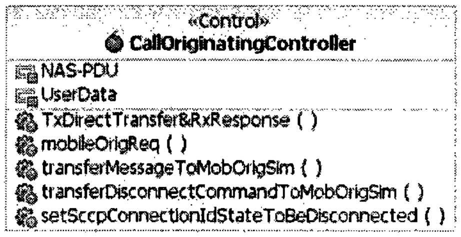

«Control»
CallOriginatingController
NAS-PDU
UserData
TxDirectTransfer&RxResponse ( )
mobileOrigReq ( )
transferMessageToMobOrigSim ( )
transferDisconnectCommandToMobOrigSim ( )
setSccpConnectionIdStateToBeDisconnected ( )

Figure A.9: Call Originating Controller Class

Class Data Members: UserData; points to the RANAP actual message. NAS-PDU; points to the Call Control or the Mobility Management actual message.

operations:

CallOriginatingController::TxDirectTransfer&RxResponse

Operation Parameters: UserData; points to the RANAP actual message.
Possible return messages: DirectTransferMessage
This operation will be invoked to transfer the RANAP message as a DirectTransferMessage. This operation will wait until it gets a response to return it back to the initiator as a DirectTransferMessage.

CallOriginatingController::mobileOrigReq

Operation Parameters: None
Possible return values: None

This operation shall be invoked by the trafficHandlerController to ask the CallOriginatingController to start the MS call originating call scenario.

CallOriginatingController::transferMessageToMobOrigSim

Operation Parameters: CelloScci_ConnectionId; holds the SCCP connection ID. CelloScci_UserData; holds the RANAP actual message.
Possible return values: None
This operation shall be invoked by the trafficHandlerController to forward the RANAP message (CelloScci_UserData) along with the SCCP connection ID (CelloScci_ConnectionId) to the CallOriginatingController class. The RANAP message originally comes from the MSC through the CPP platform.

CallOriginatingController::transferDisconnectCommandToMobOrigSim

Operation Parameters: None
Possible return value: None
This operation shall be invoked by the trafficHandlerController to forward the "Originating Call Disconnect" command to the CallOriginatingController class in order to initiate a call disconnect procedure by sending DisconnectMessage from the MS originating entity side to the MSC.

CallOriginatingController::setSccpConnectionIdStateToBeDisconnected

Operation Parameters: sccpConnectionIdState; holds the SCCP connection ID state. **Possible return value**: sccpConnectionIdState
This operation shall be invoked by the CallOriginatingController to change the state of the enumeration data type (sccpConnectionIdState) to "toBeDisconnected". This

means the value '3' will be assigned to the sccpConnectionIdState. This operation does not disconnect the SCCP connection; it just indicates the SCCP Interface Controller to disconnect the corresponding SCCP connection.

CallTerminatingController

Figure A.10 shows the UML representation for this class.

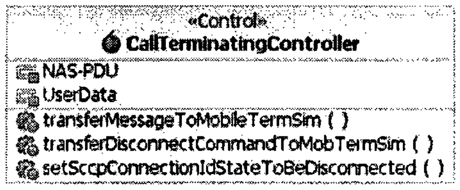

«Control»
CallTerminatingController
NAS-PDU
UserData
transferMessageToMobileTermSim ( )
transferDisconnectCommandToMobTermSim ( )
setSccpConnectionIdStateToBeDisconnected ( )

Figure A.10: Call Terminating Controller Class

Class Data Members:
NAS-PDU; points to the Call Control or the Mobility Management actual message.
UserData; points to the RANAP actual message.

operations:

CallTerminatingController::transferMessageToMobTermSim

Operation Parameters: CelloScci_ConnectionId; holds the SCCP connection ID. CelloScci_UserData; holds the RANAP actual message.
Possible return values: None
This operation shall be invoked by the trafficHandlerController to forward the RANAP message (CelloScci_UserData) along with SCCP connection ID (CelloScci_ConnectionId) to the CallTerminatingController class. The RANAP message originally comes from

the MSC through the CPP platform.

CallTerminatingController::transferDisconnectCommandToMobTermSim

Operation Parameters: None

Possible return value: None

This operation shall be invoked by the trafficHandlerController to forward the "Terminating Call Disconnect" command to the CallTerminatingController class in order to initiate a call disconnect procedure by sending a DisconnectMessage from the MS terminating entity side to the MSC.

CallTerminatingController::setSccpConnectionIdStateToBeDisconnected

Operation Parameters: sccpConnectionIdState; holds the SCCP connection ID state. **Possible return value**: sccpConnectionIdState

This operation shall be invoked by the CallTerminatingController to change the state of the enumeration data type (sccpConnectionIdState) to "toBeDisconnected". This means the value '3' will be assigned to the sccpConnectionIdState. This operation does not disconnect the SCCP connection, it just indicates the SCCP Interface Controller to disconnect the corresponding SCCP connection.

InitialUEMessageController

Figure A.11 shows the UML representation for this class.
operations:

InitialUEMessageController::startLocationUpdate

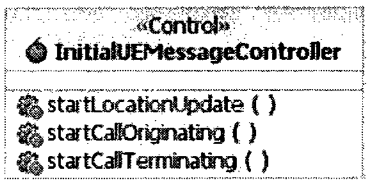


Figure A.11: Initial UE Message Controller Class

Operation Parameters: None **Possible return messages**: InitialUEMessage This operation will be invoked by the LocationUpdateController to start the location update call scenario by asking the InitialUEMessageController class to prepare the first message of this procedure. This message shall contain the LocationUpdateReqMessage carried as NAS-PDU by the InitialUEMessage.

InitialUEMessageController::startCallOriginating

Operation Parameters: None **Possible return messages**: InitialUEMessage This operation will be invoked by the CallOriginatingController to start the MS call originating scenario by asking the InitialUEMessageController class to prepare the first message of this procedure. This message shall contain the CmServiceReqMessage carried as NAS-PDU by the InitialUEMessage.

InitialUEMessageController::startCallTerminating

Operation Parameters: None **Possible return messages**: InitialUEMessage After receiving a Paging request from the MSC, this operation will be invoked by the CallTerminatingController to start the MS call terminating scenario by asking the InitialUEMessageController class to prepare the PagingResponseMessage. This message shall be carried as NAS-PDU by the InitialUEMessage.

DirectTranferController

Figure A.12 shows the UML representation for this class.

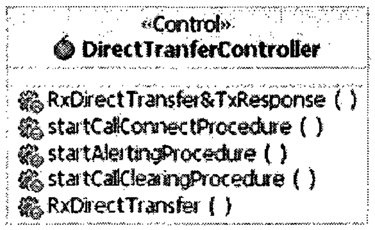

«Control»
DirectTranferController
RxDirectTransfer&TxResponse ( )
startCallConnectProcedure ( )
startAlertingProcedure ( )
startCallClearingProcedure ( )
RxDirectTransfer ( )

Figure A.12: Direct Transfer Controller Class

operations:

DirectTranferController::RxDirectTransfer&TxResponse

Operation Parameters: UserData; points to the RANAP actual message.
Possible return messages: DirectTransferMessage
This operation will be invoked to receive the RANAP message as a DirectTransferMessage. The DirectTranferController forwards the received message to the RanapMessageProxy and waits until it gets the response. The response message (DirectTransferMessage) will be returned back to this operation initiator.

DirectTranferController::RxDirectTransfer

Operation Parameters: UserData; points to the DirectTransferMessage.
Possible return messages: None

This operation will be invoked to receive the RANAP message as a DirectTransferMessage. The DirectTranferController forwards the received message to the RanapMessageProxy. No response message is expected from this operation.

DirectTranferController::startCallConnectProcedure

Operation Parameters: None **Possible return messages**: DirectTransferMessage

This operation will be invoked by the CallTerminatingController to start the call connect procedure in the MS call terminating scenario, which means to prepare the CallConnectMessage and send it to the MSC. This message shall contain the CallConnectMessage carried as NAS-PDU by the DirectTransferMessage.

DirectTranferController::startAlertingProcedure

Operation Parameters: None **Possible return messages**: DirectTransferMessage

This operation will be invoked by the CallTerminatingController to start the call alerting procedure in the MS call terminating scenario, which means to prepare the AlertingMessage and send it to the MSC. This message shall contain the AlertingMessage carried as NAS-PDU by the DirectTransferMessage.

DirectTranferController::startCallClearingProcedure

Operation Parameters: None **Possible return messages**: DirectTransferMessage

This operation will be invoked by the CallOriginatingController to start the call setup procedure in the MS call originating scenario, which means to prepare the SetupMessage and send it to the MSC. This message shall contain the SetupMessage carried as NAS-PDU by the DirectTransferMessage.

SecurityModeController

Figure A.13 shows the UML representation for this class.

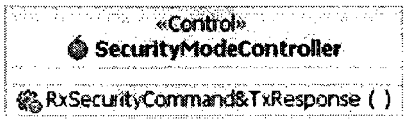


Figure A.13: Security Mode Controller Class

operations:

SecurityModeController::RxSecurityCommand&TxResponse

Operation Parameters: UserData; points to the SecurityModeCommandMessage.
Possible return messages: SecurityModeCompleteMessage
This operation will be invoked to receive the RANAP message as a SecurityModeCommandMessage. The SecurityModeController will forward the received message to the RanapMessageProxy and waits until it gets the response. The response message (SecurityModeCompleteMessage) will be returned back to this operation initiator to indicate the completion of the security mode procedure.

CommonIDController

Figure A.14 shows the UML representation for this class.

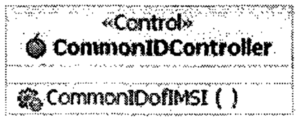


Figure A.14: Common Id Controller Class

operations:

CommonIDController::CommonIDofIMSI

Operation Parameters: UserData; points to the CommonIDMessage.
Possible return messages: None
This operation will be invoked to receive the RANAP message (CommonIDMessage) which contains the permanent-NAS-UE-Identity or the International Mobile Subscriber Identity (IMSI). The CommonIDController will forward the received message to the RanapMessageProxy.

RABAssignmentController

Figure A.15 shows the UML representation for this class.

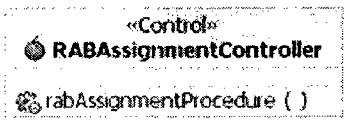


Figure A.15: RAB Assignment Controller Class

operations:

RABAssignmentController::rabAssignmentProcedure

Operation Parameters: UserData; points to the RABAssignmentReqMessage.
Possible return messages: RABAssignmentRespMessage
This operation will be invoked to receive the RANAP message as a RABAssignmentReqMessage. The RABAssignmentController forwards the received message to the RanapMessageProxy and waits until it gets the response. The response message (RABAssignmentRespMessage) will be returned back to the operation initiator to complete the RAB Assignment procedure, which is initiated by the MSC.

PagingController

Figure A.16 shows the UML representation for this class.

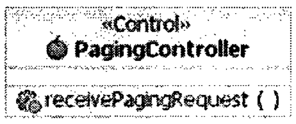


Figure A.16: Paging Controller Class

operations:

PagingController::receivePagingRequest

Operation Parameters: UserData; points to the pagingRequestMessage. **Possible return values**: Boolean
This operation will be invoked to receive the RANAP message (pagingRequestMessage). The return value indicates the CallTerminatingController (the initiator) that the paging request was received properly or not. If the paging request was received,

the CallTerminatingController will initiate an operation to respond by paging a response message.

IuReleaseController

Figure A.17 shows the UML representation for this class.

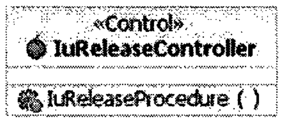


Figure A.17: Iu Release Controller Class

operations:

IuReleaseController::IuReleaseProcedure

Operation Parameters: UserData; points to the IuReleaseCommandMessage.
Possible return messages: IuReleaseCompleteMessage
This operation will be invoked to receive the RANAP message as an IuReleaseCommandMessage. The IuReleaseController forwards the received message to the RanapMessageProxy and waits until it gets the response. The response message (IuReleaseCompleteMessage) will be returned back to the operation initiator to complete the Iu Release procedure, which is initiated by the MSC.

A.1.3 RANAP and SCCP Controllers Classes

The following classes illustrate the Analysis classes for the *RANAP Controller* Component:

RanapInterfaceController

Figure A.18 shows the UML representation for this class.

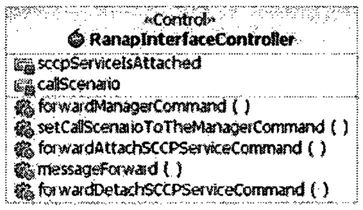

«Control»
RanapInterfaceController
sccpServiceIsAttached
callScenario
forwardManagerCommand ( )
setCallScenarioToTheManagerCommand ( )
forwardAttachSCCPServiceCommand ( )
messageForward ( )
forwardDetachSCCPServiceCommand ( )

Figure A.18: RANAP Interface Controller Class

Class Data Members: sccpServiceIsAttached; a flag indicates if the SCCP service is attached or not. callScenario; indicates the call scenario required by the Manager command.

Class operations:

RanapInterfaceController::forwardManagerCommand

Operation Parameters: managerCommand; holds the Manager command.
Possible return messages: None
This operation will be invoked by the RanapInterfaceForm to forward the Manager command to the RanapInterfaceController.

RanapInterfaceController::setCallScenarioToTheManagerCommand

Operation Parameters: managerCommand; holds the Manager command.
Possible return messages: callScenario
This operation interprets the managerCommand into a callScenario.

RanapInterfaceController::forwardAttachSCCPServiceCommand

Operation Parameters: managerCommand; holds the Manager command.
Possible return messages: None
This operation will be invoked by the RanapInterfaceForm to forward the SCCP Attach command to the RanapInterfaceController.

RanapInterfaceController::messageForward

Operation Parameters: CelloScci_ConnectionId; holds the SCCP connection ID. CelloScci_UserData; holds the RANAP actual message. sccpConnectionIdState; holds the SCCP connection ID state.
Possible return messages: None
This operation will be invoked by the trafficHandlerController and the SCCPInterfaceController to transfer the RANAP message to the RanapInterfaceController. Along with the message, this operation transfers the CelloScci_ConnectionId and the sccpConnectionIdState.

RanapInterfaceController::forwardDetachSCCPServiceCommand

Operation Parameters: managerCommand; holds the Manager command.
Possible return messages: None
This operation will be invoked by the RanapInterfaceForm to forward the SCCP Detach command to the RanapInterfaceController.

RanapInterfaceForm

Figure A.19 shows the UML representation for this class.

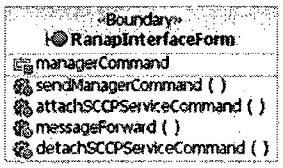


Figure A.19: RANAP Interface Form Class

Class Data Members: managerCommand; Implies the Manager command
Class operations:

RanapInterfaceForm::sendManagerCommand

Operation Parameters: managerCommand; holds the Manager command.
Possible return messages: None
This operation will initiate the Manager command towards the RanapInterfaceForm class.

RanapInterfaceForm::attachSCCPServiceCommand

Operation Parameters: clientID, attachRef; they are configuration parameters used to identify the attached service by the SccpApfi Interface.
Possible return messages: clientID, attachRef
This operation is the SccpApfi Interface attachment command, which is initiated by the Manager.

RanapInterfaceForm::messageForward

Operation Parameters: CelloScci_ConnectionId; holds the SCCP connection ID. CelloScci_UserData; holds the RANAP actual message. sccpConnectionIdState; holds the SCCP connection ID state.
Possible return messages: None
This operation will be invoked by the RanapInterfaceController to transfer the RANAP message to the RanapInterfaceForm in order to forward it to the manager. Along with the message, this operation transfers the CelloScci_ConnectionId and the sccpConnectionIdState.

RanapInterfaceForm::detachSCCPServiceCommand

Operation Parameters: clientID, attachRef; they are configuration parameters used to identify the attached service by the SccpApfi Interface.
Possible return messages: clientID, attachRef
This operation is the SccpApfi Interface detachment command initiated by the Manager.

The following classes illustrate the Analysis classes for the *SCCP Interface Controller Component*:

SCCPInterfaceController

Figure A.20 shows the UML representation for this class.

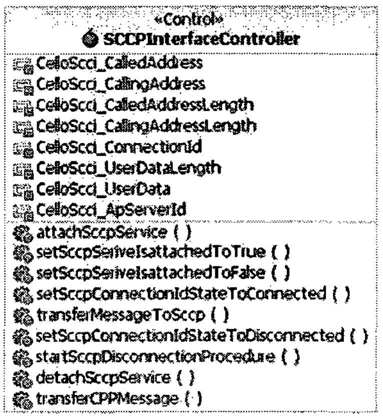

«Control»
SCCPInterfaceController
CelloScci_CalledAddress
CelloScci_CallingAddress
CelloScci_CalledAddressLength
CelloScci_CallingAddressLength
CelloScci_ConnectionId
CelloScci_UserDataLength
CelloScci_UserData
CelloScci_ApServerId
attachSccpService ( )
setSccpSeriveIsattachedToTrue ( )
setSccpSeriveIsattachedToFalse ( )
setSccpConnectionIdStateToConnected ( )
transferMessageToSccp ( )
setSccpConnectionIdStateToDisconnected ( )
startSccpDisconnectionProcedure ( )
detachSccpService ( )
transferCPPMessage ( )

Figure A.20: SCCP Interface Controller Class

Class Data Members:

CelloScci_CalledAddress; implies the address of destination SCCP node.

CelloScci_CallingAddress; implies the address of originating SCCP node.

CelloScci_CalledAddressLength; implies the length in octets of address of destination SCCP node.

CelloScci_CallingAddressLength; implies the length in octets of address of originating SCCP node.

CelloScci_ConnectionId; Identifies the SCCP connection between Data Transfer Applications.

CelloScci_UserDataLength; implies the length in bytes of the user data.

CelloScci_UserData; implies the data to be transferred to a remote SCCP user. (RANAP Message)

CelloScci_ApServerId; implies a SCCP AP server identity (configuration information).
sccpConnectionIdState;
The *sccpConnectionIdState* is an enumeration data element declared in this class. This enumeration is transferred between various components along with the CelloScci_ConnectionId. The following are the *sccpConnectionIdState* possible values and their implications about the corresponding SCCP connection:

1 "*connected*": the SCCP connection is connected.

2 "*disconnected*": the SCCP connection is disconnected.

3 "*generated*": the SCCP connection is only generated.

4 "*toBeDisconnected*": the SCCP connection is intended to be disconnected.

Class operations:

SCCPInterfaceController::transferCPPMessage

Operation Parameters: CelloScci_ConnectioId, CelloScci_UserData, and sccpConnectionIdState.
Possible return messages:
This operation forwards the CPP message -sent through the CPPSystem boundary class- to the SCCPInterfaceController class. This message contains all the parameters stated above.

SCCPInterfaceController::transferMessageToSccp

Operation Parameters: CelloScci_ConnectioId, CelloScci_UserData, and sccpConnectionIdState

Possible return messages:

This operation forwards the CelloScci_ConnectioId, CelloScci_UserData, and the sccpConnectionIdState to the CPPSystem boundary class in order to forward it to the CPP platform.

SCCPInterfaceController::attachSccpService

Operation Parameters: clientID and attachRef; they are configuration parameters used to identify the attached service by the SccpApfi Interface.
Possible return messages: clientID and attachRef
This operation is invoked by the RanapInterfaceController to forward the SCCP service attachment request.

SCCPInterfaceController::detachSccpService

Operation Parameters: clientID and attachRef; they are configuration parameters used to identify the attached service by the SccpApfi Interface.
Possible return messages: clientID and attachRef
This operation is invoked by the RanapInterfaceController to forward the SCCP service detachment request.

SCCPInterfaceController::setSccpSeriveIsattachedToTrue

Operation Parameters: None
Possible return messages: None
This operation sets the sccpServiceIsAttached flag to True; this indicates that the SCCP service is attached.

SCCPInterfaceController::setSccpSeriveIsattachedToFalse

Operation Parameters: None

Possible return messages: None

This operation sets the sccpServiceIsAttached flag to False; this indicates that the SCCP service is detached.

SCCPInterfaceController::setSccpConnectionIdStateToConnected

Operation Parameters: None

Possible return messages: None

This operation changes the sccpConnectionIdState to "connected" state. This means that the value '1' will be assigned to the sccpConnectionIdState.

SCCPInterfaceController::setSccpConnectionIdStateToDisconnected

Operation Parameters: None

Possible return messages: None

This operation changes the sccpConnectionIdState to "disconnected" state. This means that the value '2' will be assigned to the sccpConnectionIdState.

SCCPInterfaceController::startSccpDisconnectionProcedure

Operation Parameters: clientID and attachRef; they are configuration parameters used to identify the attached service by the SccpApfi Interface.

Possible return messages:

This operation initiates the SCCP disconnection procedure. This operation will be invoked if the sccpConnectionIdState is "toBeDisconnected" only.

CPPSystem

Figure A.21 shows the UML representation for this class.

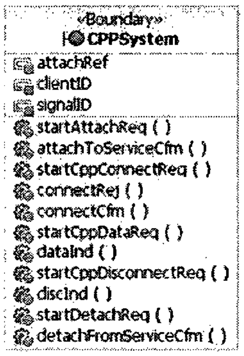


Figure A.21: CPPSystem Class

Class Data Members: clientID, attachRef, and signalID; they are configuration parameters used by SccpApfi Interface to identify the attached service.
Class operations:

CPPSystem::startAttachReq

Operation Parameters: clientID and attachRef; they are configuration parameters used to identify the attached service by the SccpApfi Interface.
Possible return messages: clientID and attachRef (confirmation message).
This operation is performed on the CPPSystem to invoke an appropriate operation from the CpxScciApfiProxy::CpxSccpApfiAttachP library (CPP platform library) to attach the SCCP service from the SccpApfi interface.

CPPSystem::attachToServiceCfm

Operation Parameters: clientID, attachRef, and SignalID; they are configuration

parameters used to identify the attached service by the SccpApfi Interface.
Possible return messages: clientID, attachRef, SignalID (confirmation message).
This operation is performed to forward the SCCP service attach confirmation to the CPPSystem.

CPPSystem::startDetachReq

Operation Parameters: clientID and attachRef; they are configuration parameters used to identify the attached service by the SccpApfi Interface.
Possible return messages: clientID, attachRef (confirmation message).
This operation is performed on the CPPSystem to invoke an appropriate operation from the CpxScciApfiProxy::CpxSccpApfiAttachP library (CPP platform library) to detach the SCCP service from the SccpApfi interface.

CPPSystem::detachFromServiceCfm

Operation Parameters: clientID and attachRef; they are configuration parameters used to identify the attached service by the SccpApfi Interface.
Possible return messages: clientID, attachRef, and SignalID (confirmation message).
This operation is performed to forward the SCCP service detach confirmation to the CPPSystem.

CPPSystem::startCppConnectReq

Operation Parameters: CalledAddress, CalledAddressLength, CallingAddress, CallingAddressLength, ConnectionId, UserData, and UserDataLength.
Possible return messages:

This operation will be invoked by the SCCPInterfaceController to ask the boundary class (CPPSystem) to initiate the SCCP connect procedure with the MSC through the CPP platform for a corresponding call scenario. This call scenario is identified by the CalledAddress and the CallingAddress.
The CPPSystem will initiate the connect procedure by invoking a connectReq operation -in the CpxScciProxy library-. Upon invoking this operation, the CPPSystem will attach the generated ConnectionId. After the SCCP signaling connection is established, the confirmation message shall contain the ConnectionId that was generated before. The CPPSystem identifies messages for corresponding call scenario using the ConnectionId. After the connection establishment is confirmed, the sccpConnectionIdState becomes "generated".

CPPSystem::connectCfm

Operation Parameters: ConnectionId, ClientID, SignalID, UserData, UserDataLength, and ApServerId.
This operation informs the CPPSystem that the SCCP connection is established successfully with the MSC.

CPPSystem::startCppDataReq

Operation Parameters: ApServerId, ConnectionId, UserData, and UserDataLength
Possible return messages: None
This operation is performed on the CPPSystem to initiate a transfer data to the MSC through the SCCI interface in the CPP platform.

CPPSystem::dataInd

Operation Parameters: ConnectionId, UserData, UserDataLength, ClientID, and SignalID

Possible return messages: None

This operation forwards the MSC data to the CPPSystem. This data is transferred through the SCCI interface to the CPP platform.

CPPSystem::startCppDisconnectReq

Operation Parameters: ConnectionId, UserData, UserDataLength, and ApServerId

Possible return messages: None

This operation will be invoked by the SCCPInterfaceController to ask the boundary class (CPPSystem) to initiate the SCCP disconnect procedure with the MSC through the CPP platform for a corresponding ConnectionId.

CPPSystem::discInd

Operation Parameters: ConnectionId, ClientID, SignalID, UserData, and UserDataLength.

This operation informs the CPPSystem that the SCCP disconnection request has been approved.

A.2 UML Use-Case Realization

The UML Use-case Realization for the RanapSim model are illustrated in this section. Some of the figures have been intentionally omitted from this section because they are already part of the main text of this thesis in Chapters 3, 4, and 5. The sequence diagrams that will be presented for each use case in the model as follows:

Handle Traffic Use-Case

Figure A.22 shows the *Transfer To RANAP Controller* Sequence Diagram.

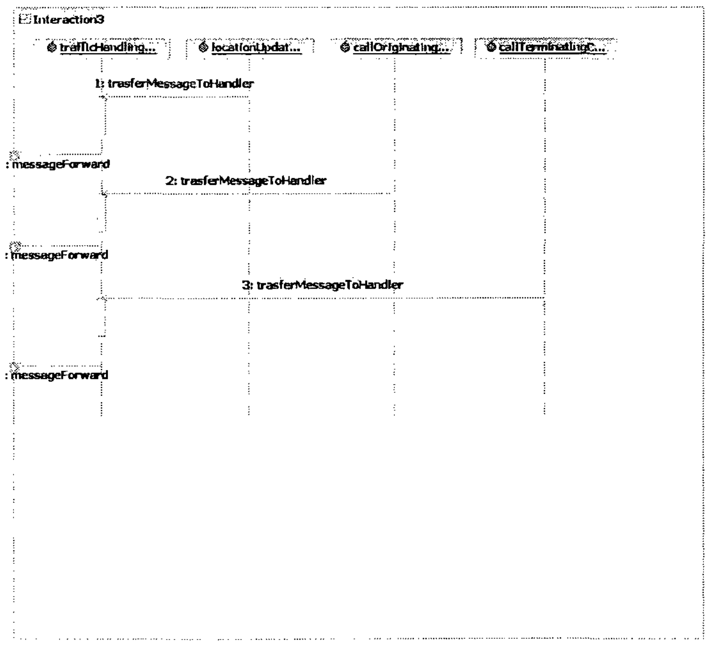


Figure A.22: Transfer To RANAP Controller Sequence Diagram

Figure A.23 shows the *Transfer To Handler* Sequence Diagram.

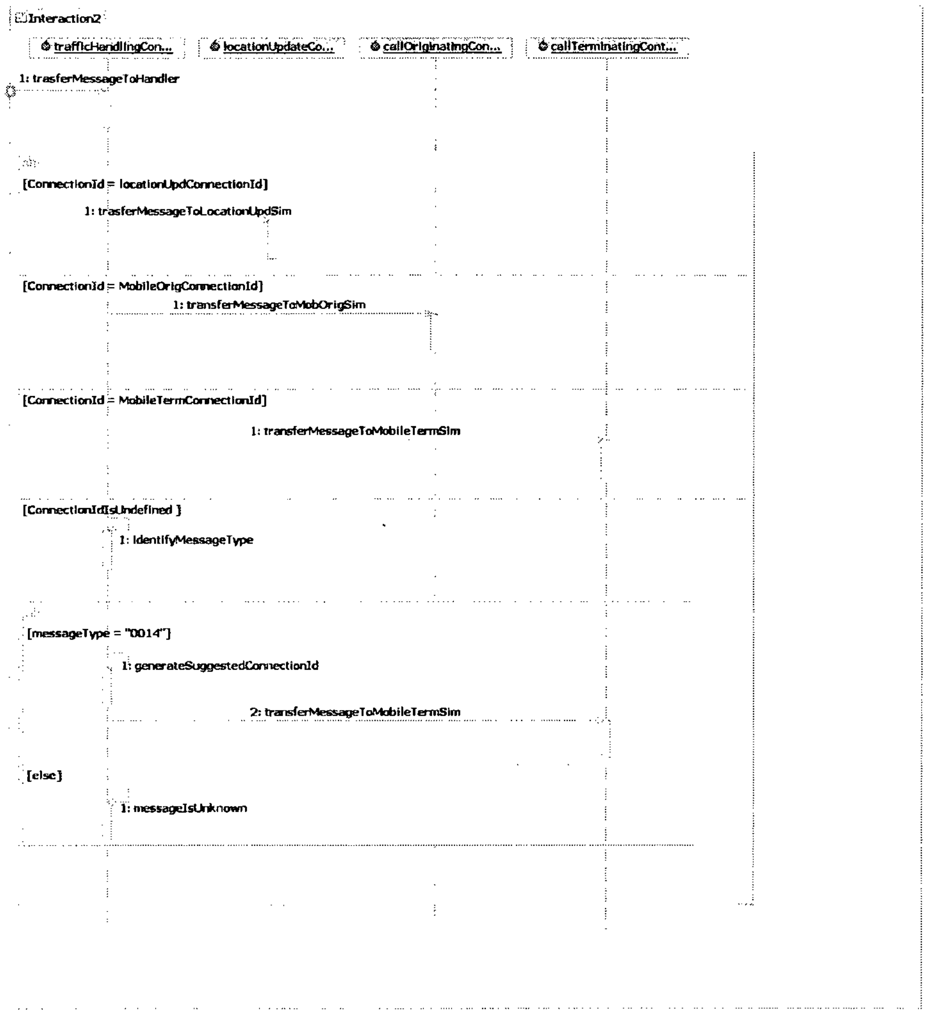

Interaction2
trafficHandlingCon...
locationUpdateCo...
callOriginatingCon...
callTerminatingCont...
1: trasferMessageToHandler
[ConnectionId = locationUpdConnectionId]
1: trasferMessageToLocationUpdSim
[ConnectionId = MobileOrigConnectionId]
1: transferMessageToMobOrigSim
[ConnectionId = MobileTermConnectionId]
1: transferMessageToMobileTermSim
[ConnectionIdIsUndefined }
1: IdentifyMessageType
[messageType = "0014"]
1: generateSuggestedConnectionId
2: transferMessageToMobileTermSim
[else}
1: messageIsUnknown

Figure A.23: Transfer To Handler Sequence Diagram

Disconnect Originating Call Use-Case

Figure A.24 shows the *Disconnect Command to the Mobile Originating* Sequence Diagram.

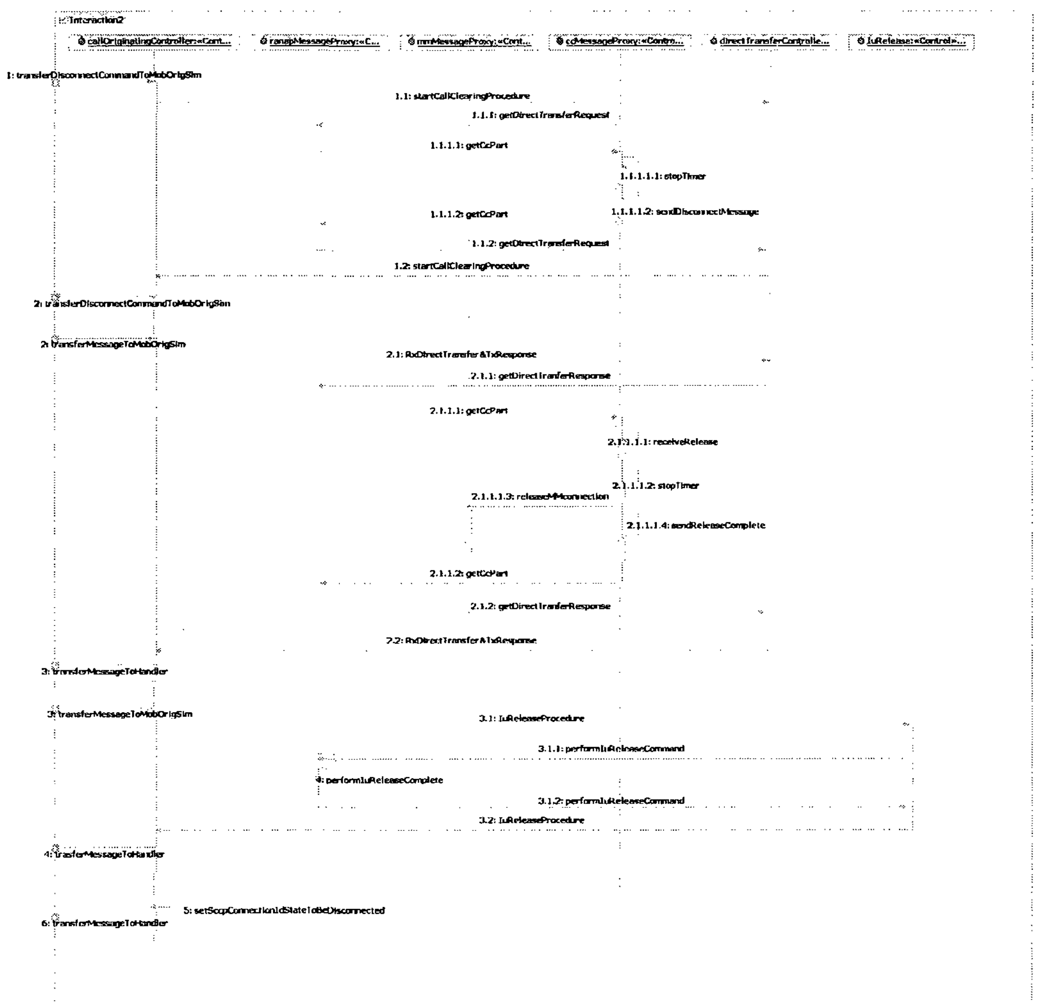

Interaction2
callOriginatingController:«Cont...
ranapMessageProxy:«C...
mmMessageProxy:«Cont...
ccMessageProxy:«Contro...
directTransferControlle...
IuRelease:«Control»...
1: transferDisconnectCommandToMobOrigSim
1.1: startCallClearingProcedure
1.1.1: getDirectTransferRequest
1.1.1.1: getCcPart
1.1.1.1.1: stopTimer
1.1.1.2: getCcPart
1.1.1.1.2: sendDisconnectMessage
1.1.2: getDirectTransferRequest
1.2: startCallClearingProcedure
2: transferDisconnectCommandToMobOrigSim
2: transferMessageToMobOrigSim
2.1: RxDirectTransfer&TxResponse
2.1.1: getDirectTranferResponse
2.1.1.1: getCcPart
2.1.1.1.1: receiveRelease
2.1.1.1.2: stopTimer
2.1.1.1.3: releaseMMconnection
2.1.1.1.4: sendReleaseComplete
2.1.1.2: getCcPart
2.1.2: getDirectTranferResponse
2.2: RxDirectTransfer&TxResponse
3: transferMessageToHandler
3: transferMessageToMobOrigSim
3.1: IuReleaseProcedure
3.1.1: performIuReleaseCommand
4: performIuReleaseComplete
3.1.2: performIuReleaseCommand
3.2: IuReleaseProcedure
4: trasferMessageToHandler
5: setSccpConnectionIdStateToBeDisconnected
6: transferMessageToHandler

Figure A.24: Disconnect Command to the Mobile Originating Sequence Diagram

Figure A.25 shows the *Disconnect Message from the MSC to the Mobile Originating* Sequence Diagram.

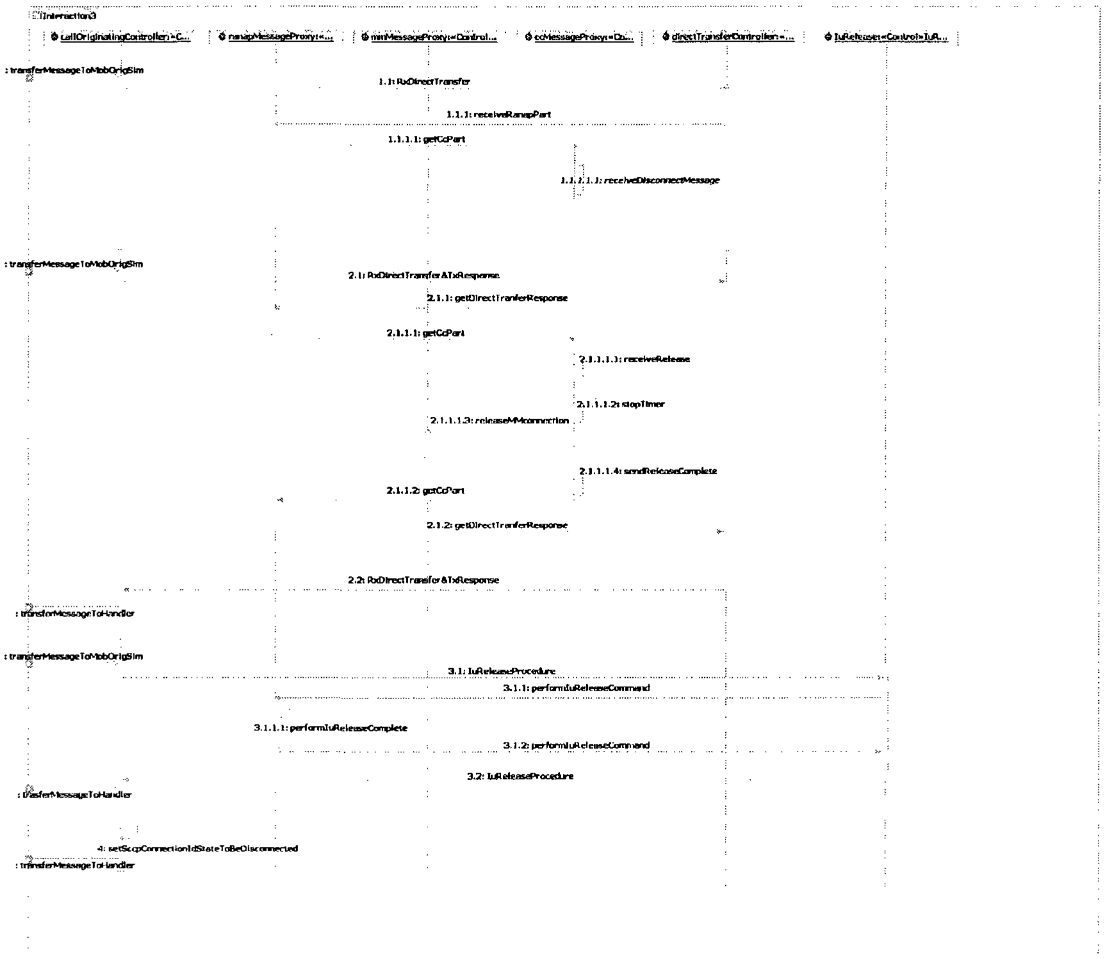

Interaction3
callOriginatingController:«C...
ranapMessageProxy:«...
mmMessageProxy:«Control...
ccMessageProxy:«Co...
directTransferController:«...
IuReleaser:«Control»IuR...
: transferMessageToMobOrigSim
1.1: RxDirectTransfer
1.1.1: receiveRanapPart
1.1.1.1: getCcPart
1.1.1.1.1: receiveDisconnectMessage
: transferMessageToMobOrigSim
2.1: RxDirectTransfer&TxResponse
2.1.1: getDirectTranferResponse
2.1.1.1: getCcPart
2.1.1.1.1: receiveRelease
2.1.1.1.2: stopTimer
2.1.1.1.3: releaseMMconnection
2.1.1.1.4: sendReleaseComplete
2.1.1.2: getCcPart
2.1.2: getDirectTranferResponse
2.2: RxDirectTransfer&TxResponse
: transferMessageToHandler
: transferMessageToMobOrigSim
3.1: IuReleaseProcedure
3.1.1: performIuReleaseCommand
3.1.1.1: performIuReleaseComplete
3.1.2: performIuReleaseCommand
3.2: IuReleaseProcedure
: trasferMessageToHandler
4: setSccpConnectionIdStateToBeDisconnected
: transferMessageToHandler

Figure A.25: Disconnect Message from the MSC to the Mobile Originating Sequence Diagram

Disconnect Terminating Call Use-Case

Figure A.26 shows the *Disconnect Command to the Mobile Terminating* Sequence Diagram.

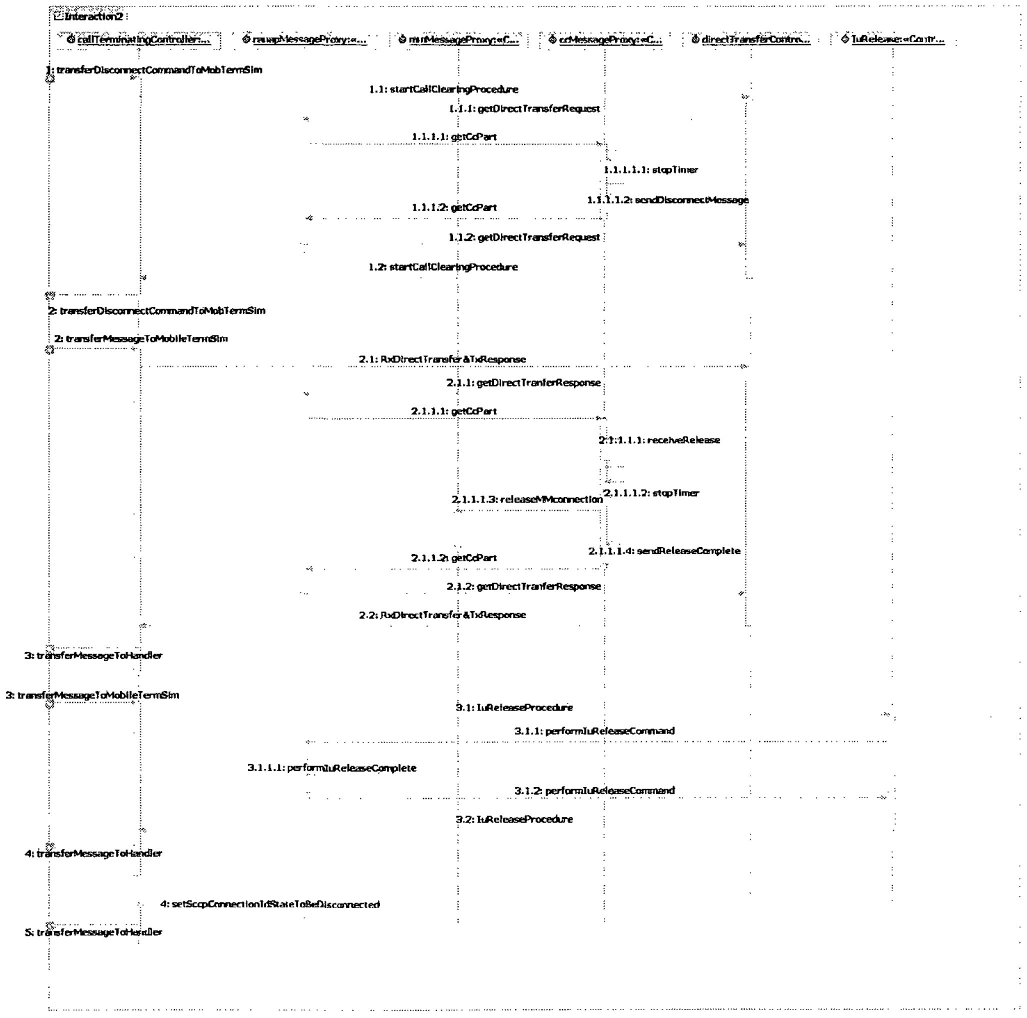

Interaction2 :
callTerminatingController:...
ranapMessageProxy:«...
mmMessageProxy:«C...
ccMessageProxy:«C...
directTransferContro...
IuRelease:«Contr...
1: transferDisconnectCommandToMobTermSim
1.1: startCallClearingProcedure
1.1.1: getDirectTransferRequest
1.1.1.1: getCcPart
1.1.1.1.1: stopTimer
1.1.1.1.2: sendDisconnectMessage
1.1.1.2: getCcPart
1.1.2: getDirectTransferRequest
1.2: startCallClearingProcedure
2: transferDisconnectCommandToMobTermSim
2: transferMessageToMobileTermSim
2.1: RxDirectTransfer&TxResponse
2.1.1: getDirectTranferResponse
2.1.1.1: getCcPart
2.1.1.1.1: receiveRelease
2.1.1.1.2: stopTimer
2.1.1.1.3: releaseMMconnection
2.1.1.1.4: sendReleaseComplete
2.1.1.2: getCcPart
2.1.2: getDirectTranferResponse
2.2: RxDirectTransfer&TxResponse
3: transferMessageToHandler
3: transferMessageToMobileTermSim
3.1: IuReleaseProcedure
3.1.1: performIuReleaseCommand
3.1.1.1: performIuReleaseComplete
3.1.2: performIuReleaseCommand
3.2: IuReleaseProcedure
4: transferMessageToHandler
4: setSccpConnectionIdStateToBeDisconnected
5: transferMessageToHandler

Figure A.26: Disconnect Command to the Mobile Terminating Sequence Diagram

Figure A.27 shows the *Disconnect Message from the MSC to the Mobile Terminating* Sequence Diagram.

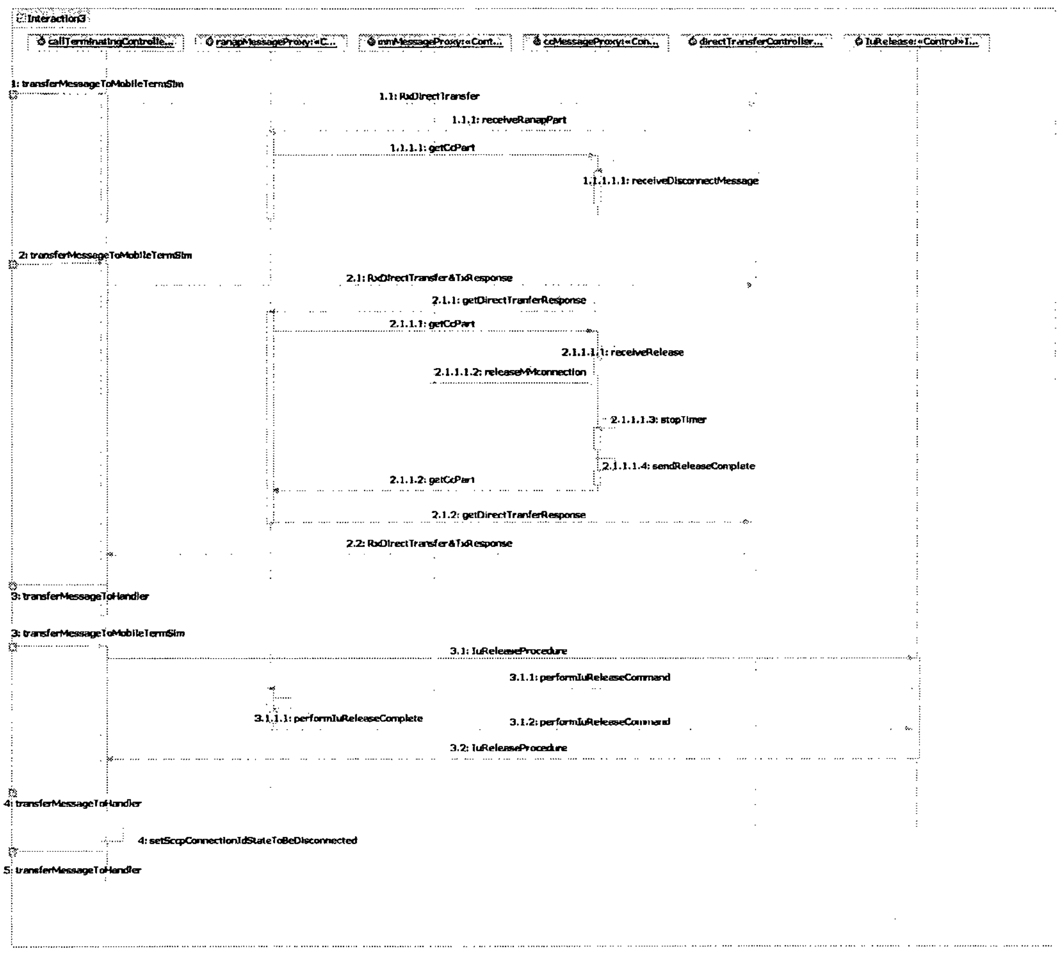

Interaction3
callTerminatingControlle...
ranapMessageProxy:«C...
mmMessageProxy:«Cont...
ccMessageProxy:«Con...
directTransferController...
IuRelease:«Control»I...
1: transferMessageToMobileTermSim
1.1: RxDirectTransfer
1.1.1: receiveRanapPart
1.1.1.1: getCcPart
1.1.1.1.1: receiveDisconnectMessage
2: transferMessageToMobileTermSim
2.1: RxDirectTransfer&TxResponse
2.1.1: getDirectTranferResponse
2.1.1.1: getCcPart
2.1.1.1.1: receiveRelease
2.1.1.1.2: releaseMMconnection
2.1.1.1.3: stopTimer
2.1.1.1.4: sendReleaseComplete
2.1.1.2: getCcPart
2.1.2: getDirectTranferResponse
2.2: RxDirectTransfer&TxResponse
3: transferMessageToHandler
3: transferMessageToMobileTermSim
3.1: IuReleaseProcedure
3.1.1: performIuReleaseCommand
3.1.1.1: performIuReleaseComplete
3.1.2: performIuReleaseCommand
3.2: IuReleaseProcedure
4: transferMessageToHandler
4: setSccpConnectionIdStateToBeDisconnected
5: transferMessageToHandler

Figure A.27: Disconnect Message from the MSC to the Mobile Terminating Sequence Diagram

Disconnect SCCP Connection Use-Case

Figure A.28 shows the *Disconnect SCCP Connection* Sequence Diagram.

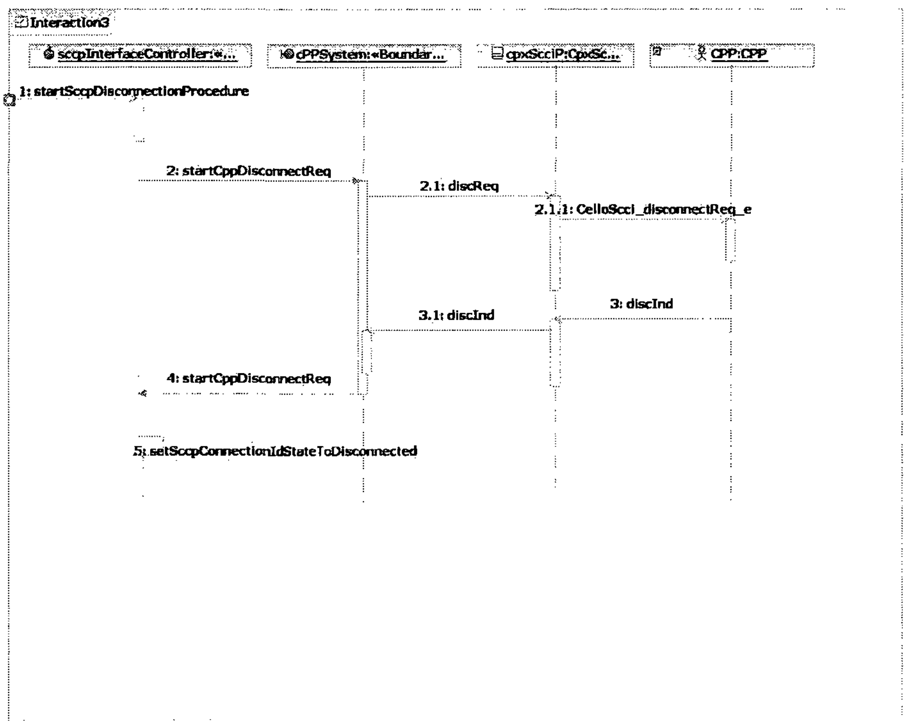


Figure A.28: Disconnect SCCP Connection Sequence Diagram

Detach SCCPApfi Service Use-Case

Figure A.29 shows the *Detach SCCPApfi Service* Sequence Diagram.

Attach SCCP Use-Case

Figure A.30 shows the *Attach SCCP* Sequence Diagram.

Figure A.31 shows the *Call Scenario Commands* Sequence Diagram.

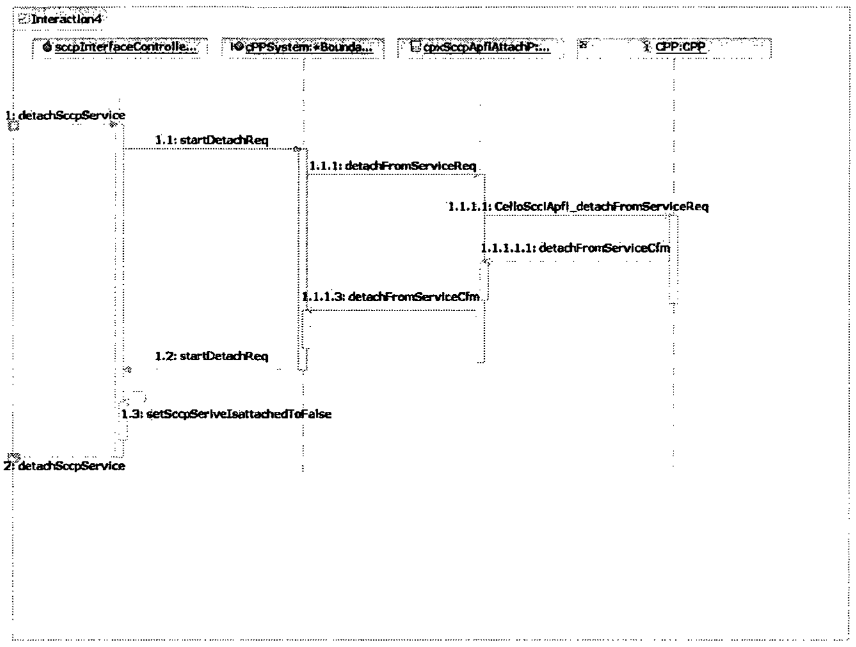

Interaction4
sccpInterfaceControlle...
cPPSystem:«Bounda...
cpxSccpApflAttachP:...
CPP:CPP
1: detachSccpService
1.1: startDetachReq
1.1.1: detachFromServiceReq
1.1.1.1: CelloScclApfl_detachFromServiceReq
1.1.1.1.1: detachFromServiceCfm
1.1.1.3: detachFromServiceCfm
1.2: startDetachReq
1.3: setSccpSerlveIsattachedToFalse
2: detachSccpService

Figure A.29: Detach SCCPApfi Service Sequence Diagram

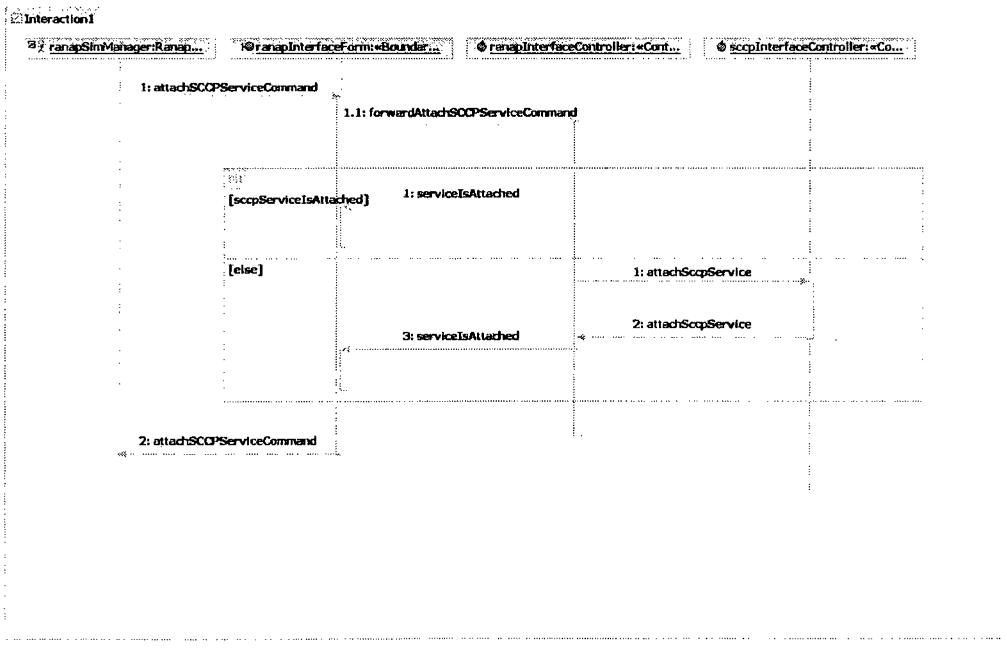

Interaction1
ranapSimManager:Ranap...
ranapInterfaceForm:«Bounder...
ranapInterfaceController:«Cont...
sccpInterfaceController:«Co...
1: attachSCCPServiceCommand
1.1: forwardAttachSCCPServiceCommand
[sccpServiceIsAttached]
1: serviceIsAttached
[else]
1: attachSccpService
2: attachSccpService
3: serviceIsAttached
2: attachSCCPServiceCommand

Figure A.30: Attach SCCP Sequence Diagram

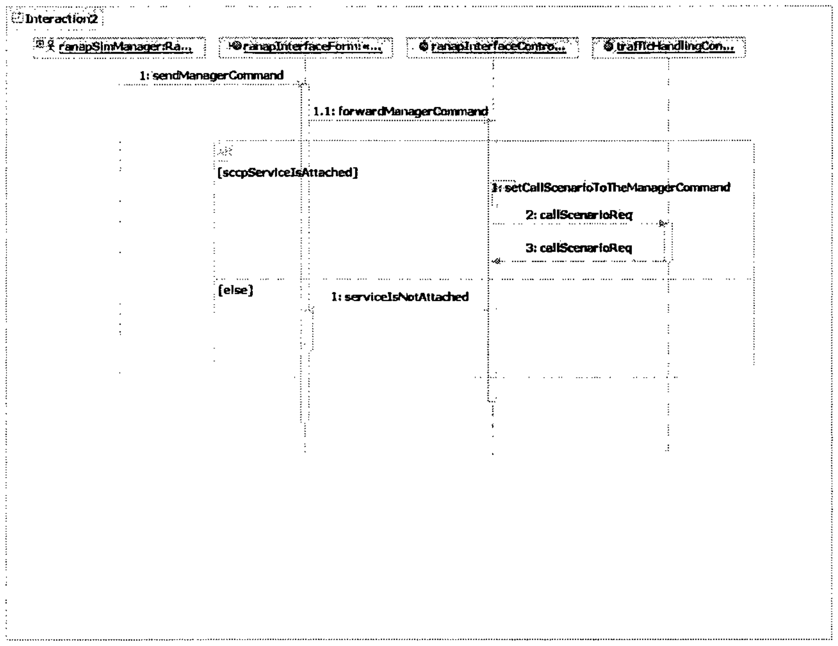


Figure A.31: Call Scenario Commands Sequence Diagram

Detach SCCP Use-Case

Figure A.32 shows the *Detach SCCP* Sequence Diagram.

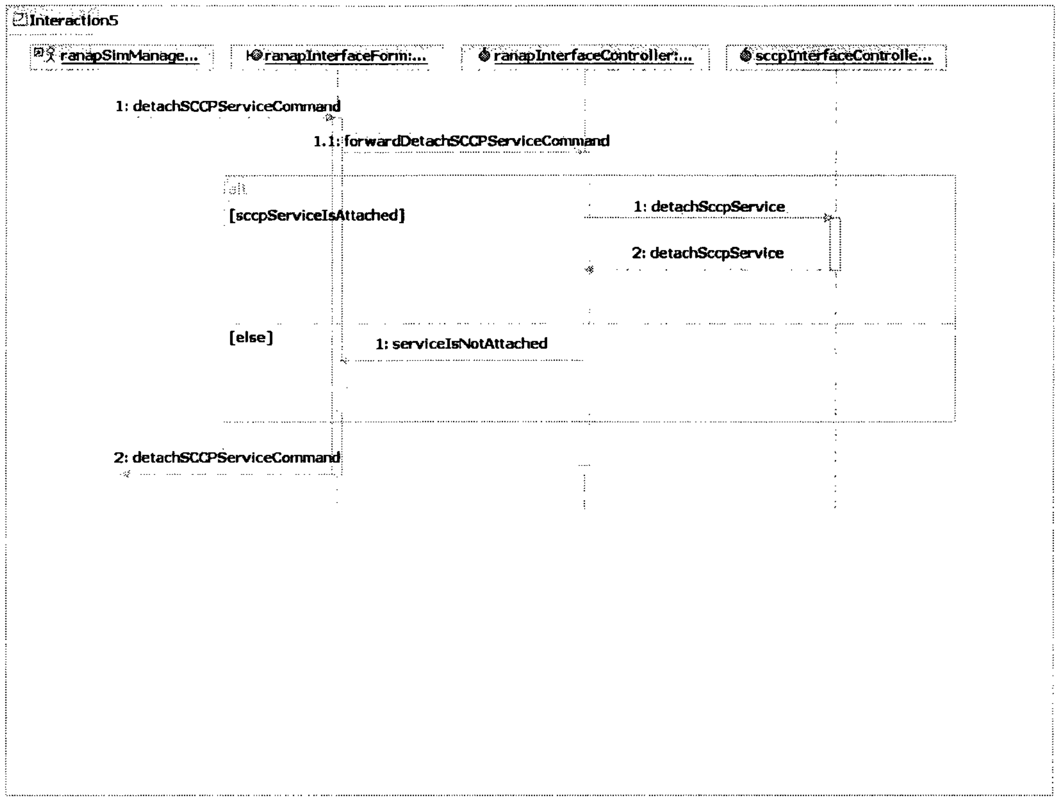


Figure A.32: Detach SCCP Sequence Diagram

A.3 UML Interfaces

UML Interfaces are model elements that define sets of operations that other classes must implement. It is possible to use the interfaces in class diagrams to specify a contract between the interface and the class that realizes the interface. Each interface specifies a well-defined set of operations that have public visibility. Those operations will be provided to another class through a "*use*" relationship.

In UML, we call the relationship between the interface and its implementing class "*interface realization*" relationship.

This section shows the UML representation for all interfaces that have been used to build the class diagram for the *RanapSim* Model, these interfaces are shown

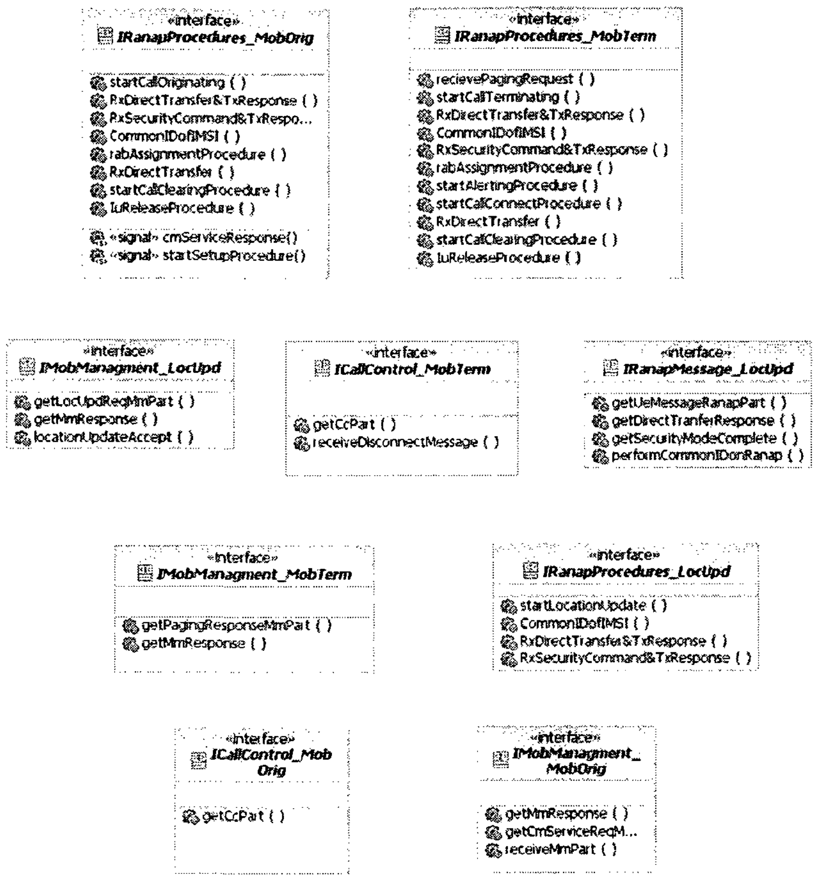


Figure A.33: UML Interface set 1

in Figure A.33 and A.34 as follows:

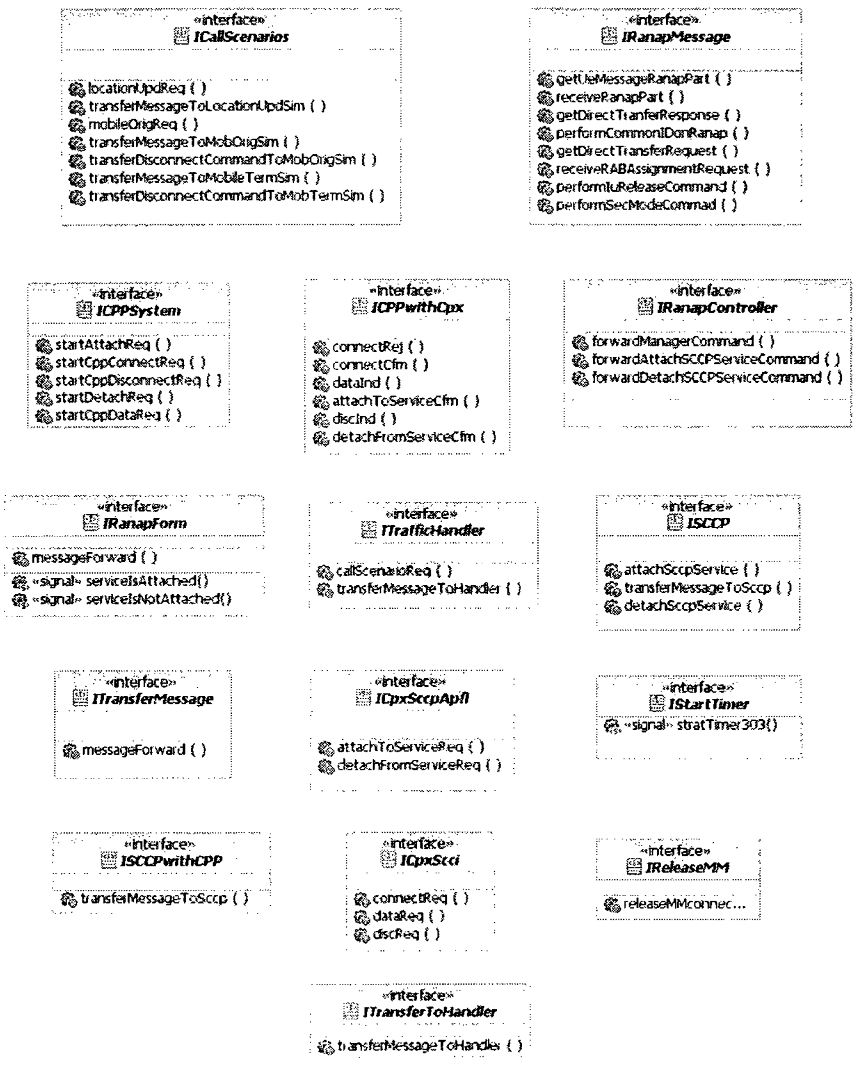


Figure A.34: UML Interface set 2

A.4 CPP Platform Classes

This section illustrates two main classes that are provided by the CPP to communicate with the SCCI and SccpApfi Interfaces, these classes illustrated as follows:

CpxScciApfiProxy::CpxSccpApfiAttachP

Figure A.35 shows the UML representation for this class.

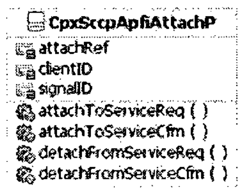


Figure A.35: CpxScciApfiProxy::CpxSccpApfiAttachP Class

Class Data Members: attachRef, clientID, and signalId.

Class operations:

CpxScciApfiProxy::CpxSccpApfiAttachP::attachToServiceReq

CpxScciApfiProxy::CpxSccpApfiAttachP::attachToServiceCfm

CpxScciApfiProxy::CpxSccpApfiAttachP::detachFromServiceReq

CpxScciApfiProxy::CpxSccpApfiAttachP::detachFromServiceCfm

CpxScciProxy::CpxScciP

Figure A.36 shows the UML representation for this class.

Class operations:

CpxScciProxy::CpxScciP::connectReq

CpxScciProxy::CpxScciP::connectCfm

CxScciProxy::CpxScciP::dataReq

CpxScciProxy::CpxScciP::dataInd

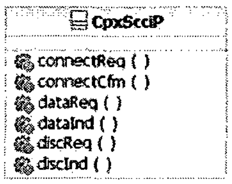


Figure A.36: CpxScciProxy::CpxScciP Class

CpxScciProxy::CpxScciP::discReq

Bibliography

[1] R. Miles and K. Hamilton. *Learning UML 2.0.* O'Reilly, 2006.

[2] J. Schmuller. *Sams Teach Yourself UML in 24 Hours.* Sams Publishing, 2004.

[3] IBM Corporation. Rational Systems Developer Tool. http://www.ibm.com, 2007.

[4] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley, 2005.

[5] Ericsson Inc. CPP Platform Specification, SCCP Signaling Service Using UML, Ericsson confidentainal information, 2007.

[6] 3GPP TS 25.413: Universal Mobile Telecommunications System (UMTS); UTRAN Iu Interface Radio Access Network Application Part (RANAP) Signaling, version 5.12.0 Release 5, 2005.

[7] 3GPP TS 24.008: Digital cellular telecommunications system; Universal Mobile Telecommunications System (UMTS); Mobile radio interface Layer 3 specification; Core network protocols; Stage 3, version 5.16.0 Release 5, 2006.

[8] H. Holma and A. Toskala. *WCDMA for UMTS: Radio Access for Third Generation Mobile Communications.* John Wiley & Sons, 2004.

[9] R. Kreher and T. Ruedebusch. *UMTS Signaling: UMTS Interfaces, Protocols, Message Flows and Procedures Analyzed and Explained.* John Wiley & Sons, 2005.

[10] 3GPP TS 24.007: Digital cellular telecommunications system; Universal Mobile Telecommunications System (UMTS); Mobile radio interface signaling layer 3; General Aspects; version 5.4.0 Release 5, 2005.

[11] 3GPP TS 25.410: Universal Mobile Telecommunications System (UMTS); UTRAN Iu Interface: General Aspects and Principles; version 5.4.0 Release 5, 2004.

[12] 3GPP TS 25.412: Universal Mobile Telecommunications System (UMTS); UTRAN Iu Interface signaling transport; version 5.2.0 Release 5, 2004.

[13] 3GPP TS 44.018: Digital cellular telecommunications system; Mobile radio interface layer 3 specification; Radio Resource Control (RRC) Protocol; version 5.22.0 Release 5, 2006.

[14] ITU-T Recommendation Q.711: Specifications of Signaling System No. 7 - Signaling connection control part, functional description of the Signaling Connection Control Part, 1996.

[15] ITU-T Recommendation Q.714: Specifications of Signaling System No. 7 - Signaling connection control part, Signaling connection control part procedures, 1996.

[16] Ericsson Inc. MGwSim Simulator Architecture, Ericsson confidentainal information, 2005.

[17] X. Li, S. Li, C. Gorg, and A. Timm-Giel. Traffic Modeling and Characterization for UTRAN. In *Wired/Wireless Internet Communications*, LNCS 3970, pp. 190-201, Springer-Verlag, 2006.

[18] D. Staehle, K. Leibnitz, and P. Tran-Gia. Source Traffic Modeling of Wireless Applications. *International Journal of Electronics and Communications*, 55 (1): 27-36, 2001.

[19] A. Garcia, E. Garcia, M. Alvarez-Campana, J. Berrocal and E. Vazquez. A Simulation Tool for Dimensioning and Performance Evaluation of the UMTS Terrestrial Radio Access Network. In *Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems: Protocols and Systems for Interactive Distributed Multimedia*, LNCS 2515, pp. 49-60, Springer-Verlag, 2002.

[20] S. Tabbane. Modelling the MSC/VLR processing load due to mobility management. In Proceedings *International Conference of Universal Personal Communications*, volume 1, pp. 741-744, Florence, Italy, 1998.

[21] ITU-T Union. Worldwide mobile cellular subscribers to reach 4 billion mark late 2008. http://www.itu.int/newsroom/press_releases/2008/29.html.

[22] B. Reed. Global telecom revenue to hit $2 trillion in '08, Network World, http://www.networkworld.com/news/2008/091708-global-telecom.html.

[23] K. Prasad and K. Poon. Design of a protocol controller for wireless information networks. In Proceedings *Local Computer Networks*, pp. 519-524, Minneapolis, Minnesota, USA, 1992.

[24] T. Suzuki, S. Shatz, and T. Murata. A Protocol Modeling and Verification Approach Based on a Specification Language and Petri Nets. *IEEE Transactions on Software Engineering*, 16 (5): 523-536, 1990.

[25] J. Lee and P. Hsu. Design and Implementation of the SNMP Agents for Remote Monitoring and Control via UML and Petri Nets. *IEEE Transactions On Control Systems Technology.* 12 (2): 293-302, 2004.

[26] V. Garousi, L. Briand, and Y. Labiche. Traffic-aware Stress Testing of Distributed Systems Based on UML Models. In Proceedings *International Conference on Software Engineering*, pp. 391-400, Shanghai, China, 2006.

[27] Catapult Communications. MGTS System Reference Manual. http://www.catapult.com/products/mgts.htm, 2007.

[28] Catapult Communications. DCT2000 System Reference Manual. http://www.catapult.com/products/dct2000.htm, 2007.

[29] Polystar Inc. SOLVER System Information. http://www.polystar.com/SOLVER system/Downloads/Post.aspx, 2007.

[30] Ericsson Inc. 3Gsim Solution, Confidential Information.

[31] Ericsson Inc. UTMS Solution, Confidential Information.

[32] D. Loukatos, L. Sarakis,K. Kontovasilis, C. Skianis, and G. Kormentzas. Tools and Practices for Measurement-based Network Performance Evaluation. In Proceedings *Personal, Indoor and Mobile Radio Communications*, pp. 1-5, Athens, Greece, 2007.

[33] D. Loukatos, L. Sarakis, K. Kontovasilis, and N. Mitrou. An Efficient ATM Traffic Generator for the Real-Time Production of a Large Class of Complex Traffic Profiles. *Journal of Communication and Networks.* 7 (1): 54-64, 2005.

[34] O. Kone and R. Castanet. Test generation for interworking systems. *Computer Communications*, 23 (7): 642-652, 2000.

[35] N. Celandroni, E. Ferro, and F. Potorti. A Traffic Generator for Testing Communication Systems: Presentation, Implementation and Performance. *Real-Time Systems*, 13 (1): 5-24, 1997.

[36] V. Frost, B. Melamed. Traffic modeling for telecommunications networks. *IEEE Communications Magazine*, 32 (3): 70-81, 1994.

[37] C. Barrett, M. Drozda, M. Marathe, S. Ravi, and J. Smit. A Mobility and Traffic Generation Framework for Modeling and Simulating Ad hoc Communication Networks. *Scientific Programming*, 12 (1): 1-23, 2004.

[38] F. Sandu, S. Cserey, I. Szekely, D. Robu, and T. Balan. Simulation of an advanced mobile communication network. In Proceedings *Optimization of Electrical and Electronic Equipment*, pp. 223-230, Brasov, Romania, 2008.

[39] A. Varga. OMNeT++ Discrete Event Simulation System Version 3.2 User Manual. www.omnetpp.org, 2005.

[40] 3GPP TS 23.205: Universal Mobile Telecommunications System (UMTS); Bearer-independent circuit-switched core network; Stage 2, version 5.16.0 Release 5, 2008.

[41] ITU-T Recommendation Q.711: Language and General Doftware Aspects for Telecommunication Systems, Specification and Description Language (SDL), 1992.

[42] K. Konishi, K. Maeda, K. Sato, A. Yamasaki, H. Yamaguchi, T. Higashino, and K. Yasumoto. MobiREAL Simulator - Evaluating MANET Applications in Real Environments. In Proceedings *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 499-502, Washington, DC, USA, 2005.

[43] M. Mackaya, O. Kone, and R. Castanet, Modeling Location Operations in UMTS Networks. In Proceedings *Modeling Analysis and Simulation of Wireless and Mobile Systems*, pp. 69-73, Atlanta, Georgia, USA, 2002.

[44] M. Mackaya and R. Castanet. Modeling and testing location based application in UMTS networks. In Proceedings *International Conference on Telecommunications*, volume 1, pp. 189-195, Zagreb, Croatia, 2003.

[45] S. Rios. Location Based Services: Interfacing to a mobile Positioning Center. www.wirelessdevnet.com/channels/lbs.

[46] 3GPP TS 23.171: Universal Mobile Telecommunications System (UMTS); Functional description of location services; Stage 2, version 3.11.0, 2004.

[47] ITU-T Recommendation Z.120: Language and General Software Aspects for Telecommunication Systems, Message Sequence Charts (MSC), 1993.

[48] R. Thomas, H. Gilbert, and G. Mazziotto. Influence of the Moving of the Mobile Stations on the Performance of a Radio Mobile Cellular Network. In Proceedings *Digital Land Mobile Radio Communications*, Copenhagen, Denmark, 1988.

[49] T. Kim, Q. Yang, S. Park, and Y. Shin. SDL Design and Performance Evaluation of a Mobility Management Technique for 3GPP LTE Systems. In *SDL 2007: Design for Dependable Systems*, LNCS 4745, pp. 272-288, Springer, 2007.

[50] Ericsson Inc. Formal Check Process, Ericsson confidentainal information, 2005.