Collaboration-Oriented Modeling of an Offshore Group Communication System

DOAN THI HONG TAM

KTH Information and
Communication Technology

Master of Science Thesis
Stockholm, Sweden 2009

TRITA-ICT-EX-2009:59

Master Thesis

Collaboration-Oriented Modeling of an Offshore Group Communication System

Doan Thi Hong Tam

Master of Science in Network Security and Mobile Computing

Submission date:	June 2009
Supervisor:	Frank Alexander Kraemer
Professor - NTNU:	Rolv Bræk
Professor – KTH:	Gerald Maguire

Norwegian University of Science and Technology (NTNU)
Royal Institute of Technology (KTH)

Abstract

This thesis studies the SPACE method by creating building blocks for a Push to Talk (PTT) service in WLAN environment. The structure and behavior of a PTT service is analyzed and discussed. We have modeled the behavior of a PTT service with the GUI of the PTT client. As a result, several of building blocks for a PTT service have been proposed. They can be stored in a library for a later reuse. We consider that the SPACE method well suited for developing a PTT service.

Sammanfattning

Denna avhandling studerar SPACE metoden genom att skapa byggstenar för en PTT-tjänst i WLAN miljö. Struktur och beteende för en PTT-tjänst analyseras och diskuteras. Vi har utformat en model för beteendet av en PTT-tjänst med GUI av PTT klient. Som ett resultat har flera byggstenar för en PTT-tjänst föreslagits. Vi anser att SPACE metoden är väl lämpad för att utveckla ett PTT-tjänst.

Acknowledgements

This master thesis is the final project of a Master of Network Security and Mobile Computing taken place at Royal Institute (KTH) of Technology and Norwegian University of Science and Technology (NTNU). The thesis has been written at the Department of Telematics at NTNU during the spring semester of 2009. My thesis supervisors have been Professor Rolv Bræk at NTNU, Professor Gerald Maguire at KTH, and Frank Alexander Kraemer at NTNU.

First of all, I would like to use this opportunity to thank my academic supervisors, Professor Rolv Bræk at NTNU, Professor Gerald Maguire at KTH, as well as Frank Alexander Kraemer at NTNU. My supervisors always give me helpful supports, invaluable advices, and quick feedback on my work during the six month period of the thesis. I strongly believe that their invaluable advices have been significantly enhanced the quality of my work. Special thank to Frank Alexander Kraemer for his supportive discussion and guidance on my project. He was definitely a good navigator for my thesis work.

I would like to acknowledge the contribution of my friend Nattanond Savaphant for his comments and discussion during the project. Another acknowledgement would go to Frode Flægstad, TelCage/Telenor R&I for giving me an explanation of the problem domain.

I would like to show my appreciation to the NordSecMob consortium. The consortium gave me a great opportunity to study in this master program in a cross-culture environment.

Last but not least, I would like to thank my family and all of my friends for their supports, encouragements, and giving me motivations during my study. Special thank to my friend Dam Hoang Anh for his great source of inspiration to me.

Trondheim, June 16, 2009

Doan Thi Hong Tam

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

Abbreviations	Meaning
COOS	Connected Objects Operation System
ESM	External State Machine
GPRS	General Packet Radio Service
GUI	Graphical User Interface
IMS	IP Multimedia Subsystem
OMA	Open Mobile Alliance
PoC	Push to Talk over Cellular
PSTN	Public Switched Telephone Network
PTT	Push to Talk
RTCP	Real-time Transport Protocol
RTP	Real-time Transport Protocol
SCG	Sea Cage Gateway
SCTP	Stream Control Transmission Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
TB	Talk Burst
TBC	Talk Burst Control
TBCP	Talk Burst Control Protocol
TCP	Transmission Control Protocol
TLA	Temporal Logic of Actions
TLC	Temporal logic of causality
TLS	Transport Layer Security
UDP	User Datagram Protocol
UML	Unified Modeling Language
WLAN	Wireless Local Area Network

1 Introduction

There are many situations in which it is desirable for people working as a group to be able to easily communicate with each other. This thesis is concerned with the specific case of group communication in a Wireless Local Area Network (WLAN) environment installed in off-shore fish farms shown in Figure 1-1. In particular we study Push to Talk (PTT) communication. PTT provides a half-duplex communication for mobile phone users. The PTT service allows only one person to speak at the time and the others will listen to him or her in the communication session. Furthermore, due to the presence of the WLAN environment, PTT can utilize Voice over IP (VoIP) and Session Initiation Protocol (SIP) technology to set up a call. PTT can therefore be provided cheaply by avoiding the use of service providers in the Public Switched Telephone Network (PSTN).

The specification of PTT was defined by a consortium including Motorola, Nokia, Ericson, Siemens AG, and AT&T Mobility and approved as standards by the Open Mobile Alliance (OMA). There were 45.6 million PTT subscribers in 2006 and 64 million in mid 2007. It is expected that the number of PTT subscribers reaches 340 million by 2013 [1]. Therefore, the use of a PTT service for our off-shore group communication builds on a well proven technology for group communication.

Developing a PTT service can be very difficult since it has various participating components. A component has its behavior and interaction with other components. Furthermore, current specification of PTT service proposed by OMA is based on an IMS architecture. Hence, making the process of developing a PTT service easier is a great demand.

SPACE is an engineering method for rapid service development, developed at NTNU. In the SPACE method, a service is composed by building blocks using Arctis tool. A building block is a unit of specifications. It has participants, structure, and behavior. The participant is illustrated by collaboration roles. The structure is described by UML 2.0 collaborations. The internal behavior is illustrated by UML 2.0 activities diagrams. The external behavior is represented by a state machine. Then in SPACE method, the collaborations and activities of building blocks are transformed to state machines which generate executable code in a second step using another tool (Ramses).

The SPACE method is based on three principles to speed up development of services: collaborative building blocks, model transformation and code generation, and formal analysis of models [2]. Arctis and Ramses are tool suites in the SPACE method to support these concepts. In this project, we have built select building block for a PTT service in a WLAN environment. We have modeled the behavior of the service with the user interface blocks. Our building blocks for a PTT service can be stored in a library for later use.

1.1 Problem statement

In this thesis, we model a PTT system in a WLAN environment. Since the current specification for PTT proposed by Open Mobile Alliance (OMA) is based on an IMS architecture, we make some variations from the OMA specification for Push to talk over Cellular to develop a PTT system in a WLAN environment.

Figure 1-1 shows the Distributed Infrastructure of Connected Objects (DICO) network infrastructure of the offshore group communication system. As shown in the figure, the network infrastructure includes four main components: Sea Cage Gateway (SCG) Headquarters, SCG Operations Centre, several Feed barges, and multiple SCG Edges. A WLAN covers the working environment of each Feed barge. The communication link between the Operations Centre and the headquarters, third party service providers, and the management center is via WAN connections, while the other communication links use WLAN technology. Every fish famer is equipped with a mobile device. This device can use the WLAN for access through host spots at each location to communicate.

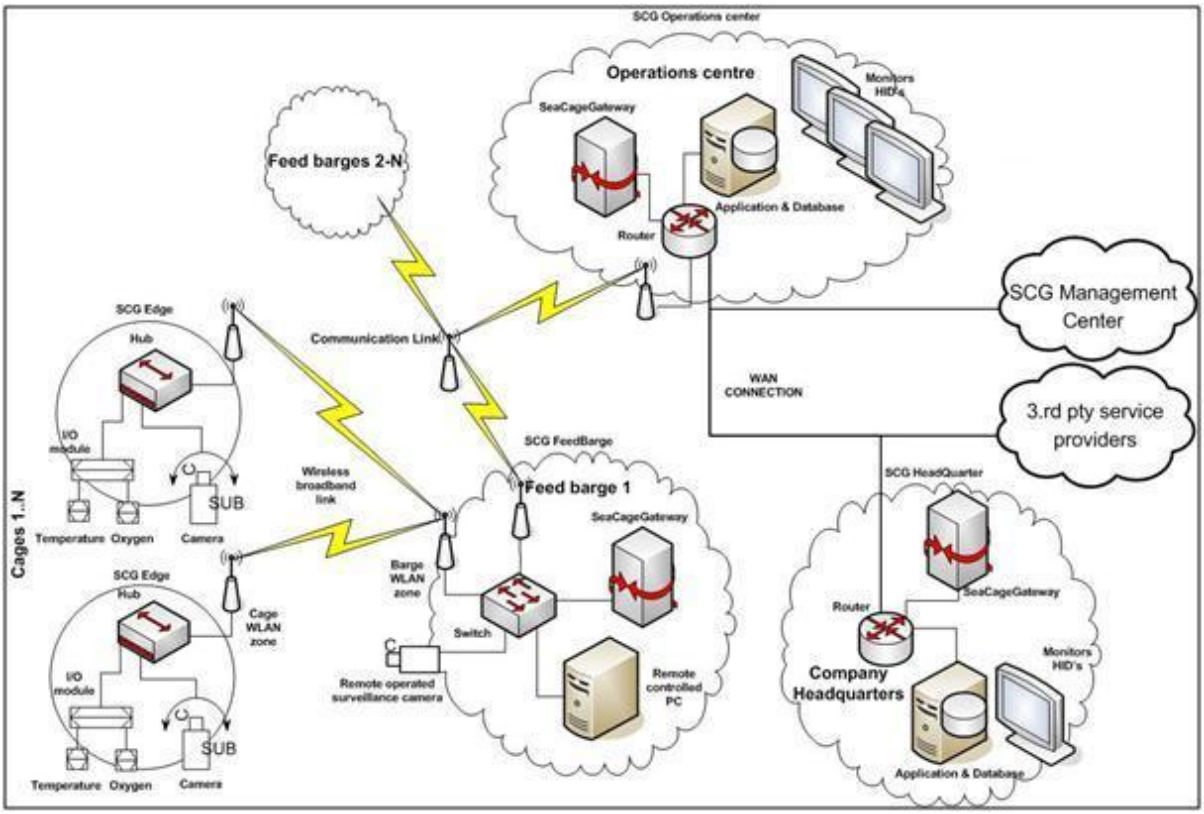


Figure 1-1: Network Infrastructure (taken from [3], appears with permission of Telcage/Telenor R&I)

We use a centralized PTT server which is responsible for managing the PTT service for the entire network. It is located on the Operations Centre. The PTT clients are located on the mobile devices of the fish farmer and the operators on the feed barges/boats and the Operations Centre.

In our design, the Session Initiation Protocol (SIP) is used to manage PTT sessions and the Real-time Transport Protocol (RTP) is used to transport the media stream. The Real-time

Transport Control Protocol (RTCP) is used in conjunction with RTP to provide feedback concerning the media session. The media exchanged among PTT clients in our project is only speech. Hence, other types of media are out of the scope of this project.

PTT clients can be in an instant personal talk, instant group talk, or *ad-hoc* instant group talk to communicate with other clients. In order to talk to others, a PTT user has to hold down a PTT button on the mobile device when talking via a PTT client. When the button is released, the talk session is terminated. The right for a PTT client to speak is arbitrated through a protocol called Talk Burst Control Protocol (TBCP) as defined in the OMA PoC specifications [4]. This protocol will be described in section 2.2.2 on page 13.

The main focus of the project is modeling the performance of message flow in PTT session initiation and termination. The scope of the project includes these tasks:

- Build a graphical user interface (GUI) for a PTT client using the Java Swing API [5] with the Eclipse integrated development environment (IDE) [6] .
- Use Arctis described in section 2.1.1 on page 8 to build selected blocks of a PTT service with support for instant personal talk, instant group talk, and *ad hoc* instant group talk.

The actual media delivery and presence functions are out of the project's scope. These features have been implemented in a number of existing SIP user agents, such as minisip [7], X-lite [8], Ekiga [9].

1.2 Structure of the report

The rest of this report is organized as follows:

Chapter 2 gives some background for the thesis project. Firstly, the idea of the SPACE method and its tools are described, then a system overview of the Push to Talk service is given.

Chapter 3 analyzes the system requirements in order to support the process of creating building blocks for Push to Talk using the SPACE method. In this chapter based upon the principal of the SIP protocol, its logic, and the specification of the Push to Talk over Cellular proposed by the OMA, we make some simplifications of message flows to create some use cases. At the end of this chapter, the specific use cases are selected and described in detail.

Chapter 4 analyzes the objects and the collaborations in a Push to Talk service. The modeling process is mainly based on the use cases specified in chapter 3.

Chapter 5 models the building blocks of a PTT service together with the GUI blocks.

Chapter 6 discusses the outcome of this thesis and other solutions for system design.

Chapter 7 concludes the work and suggests future work.

2 Background

In this section, we present the background of this thesis project. Firstly, we describe the idea of the SPACE method and explain how to use its tools to support the creation of services in reactive systems. Following this a system overview of group communication, specifically Push to Talk, which is focus in our project, is given.

2.1 SPACE method

The SPACE[1] method is an engineering approach for creating reactive systems by mean of building blocks [2]. The SPACE method utilizes UML 2.0 collaborations and activities to describe systems as compositions of building blocks. Some of these building blocks are taken from a library [10]. When the system specification is created in the form of building blocks containing UML collaborations and activities, it can be transformed automatically into executable state machines by the Arctis tool, then from these state machines into executable Java code by another tool (Ramses). Arctis and Ramses are two tool suites in the SPACE method that support creation of services. An overview of how to use these tools is given in section 2.1.1.

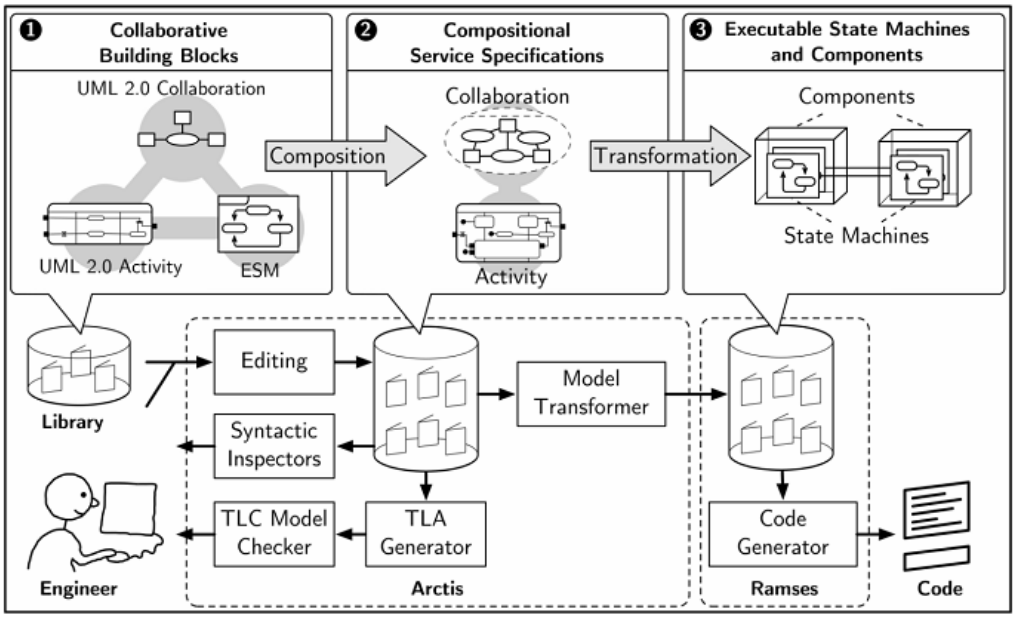


Figure 2-1: Model Transformation using the SPACE method (taken from [2])

Figure 2-1 illustrates the SPACE method. Services are composed from building blocks which may be retrieved from a library. The resulting set of blocks is transformed into executable state machines and components using the Arctis tool. The roles of Arctis and Ramses tools in each process are depicted clearly in the figure.

[1] SPACE stands for specification by activities, collaborations, and external state machines.

Compositional Temporal Logic of Actions (cTLA) is used to check the correctness of the model and the transformation [11]. This method has two variants: cTLA/c and cTLA/e (see Figure 2-2). The cTLA/c formalizes the collaborative service specification by UML 2.0 activities. It transforms UML collaborations and activities to UML state machines. The cTLA/e formalizes the behavior of the UML state machines and transforms these state machines into executable code.

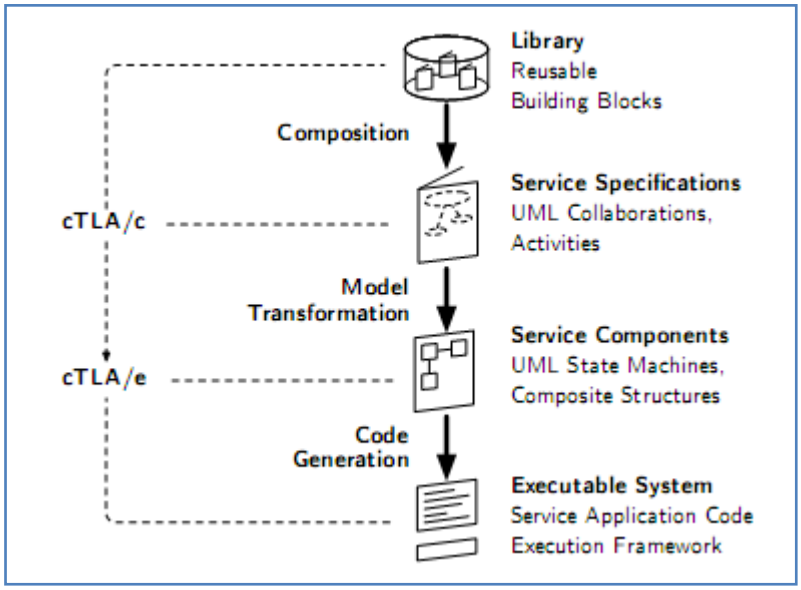


Figure 2-2: The SAPCE method (taken from [11])

The result of Temporal Logic of Actions (TLA) generator is used as input for the model checking Temporal Logic of Causality (TLC). This is an external model checker invoked by Arctis from the command line. The model checker generates a number of theorems which are used to verify the specification [11].

In the SPACE method, a system specification is composed of several building blocks which are modeled in UML as collaboration, activities, and External State Machines (ESMs). UML collaborations describes the structure of the system, for example the participants in services. Collaborations specify a relationship between participants in services. These collaborations describe how the participants relate to each other. Collaboration consists of collaboration *roles* of the participants of the collaboration and collaboration *uses* which describe the functionality between the participants of the collaboration [12]. In a UML diagram, the collaboration role is rectangular and the collaboration use is elliptical. Figure 2-3 shows an example of a collaboration login. The rectangular client and server are collaboration roles. The elliptical login is a collaboration use.

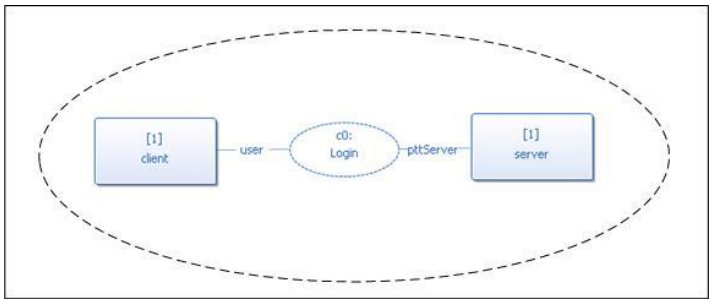


Figure 2-3: Collaboration Login

UML activities illustrate the behavior of participants in a service and the interaction between them. UML activities define a flow graph of nodes connected by edges. The control and data flow along the edges are operated by other nodes or represent information routed to them [12]. The summary of activity nodes in Arctis is given in Table 2-1.

Table 2-1: Activity node [13] [12]

Activity Node	
Initial node	A system activity starts in the initial node when it is activated
Fork node	A fork node has one incoming flow and at least two outgoing flows. All the incoming flow and outgoing flows maybe either all object flows or all control flows. However, if the incoming flow is an object flow, the outgoing flows may include both object flows or control flows.
Join node	A join node synchronizes several incoming flows. When all incoming flows arrive at a join node, an outgoing flow is produced.
Decision node	When an incoming flow arrives at a decision node, one of its outgoing flows is chosen.
Merge node	A merge node has at least two incoming flows and one outgoing flow. The outgoing flow is produced when at least one incoming flow arrives at the merge node.
Timer	When an incoming flow arrives at the timer node, it will be sent out when the timer expires. The timer is usually used after a fork node since all outgoing flows of a fork node are executed in parallel. Hence, the timer is used to send a flow after the other flows.
Flow final node	A flow final node simply ends a flow. It does not terminate the activity.
Activity final node	An activity final node terminates an activity.

Call operation action	A call operation action performs a local method call in an activity. It is assigned to one activity partition.
Call behavior action	A call behavior action is an instantiation of an activity. It has input and output pins corresponding to the parameter nodes of the activity.
Activity parameter node	There are three types of activity parameter nodes: starting parameter nodes, terminating parameter nodes, and streaming parameter nodes. • The activity starts/terminates in a starting/terminating parameter node. When the activity starts, it is active. When the activity terminates, it is inactive. • When the activity is active it can send or receive data via streaming parameter nodes.
Pin	When an activity is instantiated as a call behavior action, its activity parameter nodes are turned to be corresponding pins in call behavior action. There are two types of pin: input pin and output pin.
Send/Receive signal action	Send/receive signal action is used to send/receive a signal to/from the environment. The send signal action may have an input pin. The receive signal action may have an output pin.

Figure 2-4 illustrates the deference between activities and call behavior actions, activity parameter nodes and pins. On the left hand side, the figure shows the activity of the *Login* service with a starting parameter node *start* and two alternative terminating parameter nodes *loginFail* and *loginSuccess*. The activity of the Login service can be terminated via either *loginFail* parameter node or *loginSuccess* parameter node. On the right hand side, the figure shows the call behavior action which is an instantiation of the *Login* service and its input/output pins corresponding to the starting/terminating parameter nodes of the *Login* service.

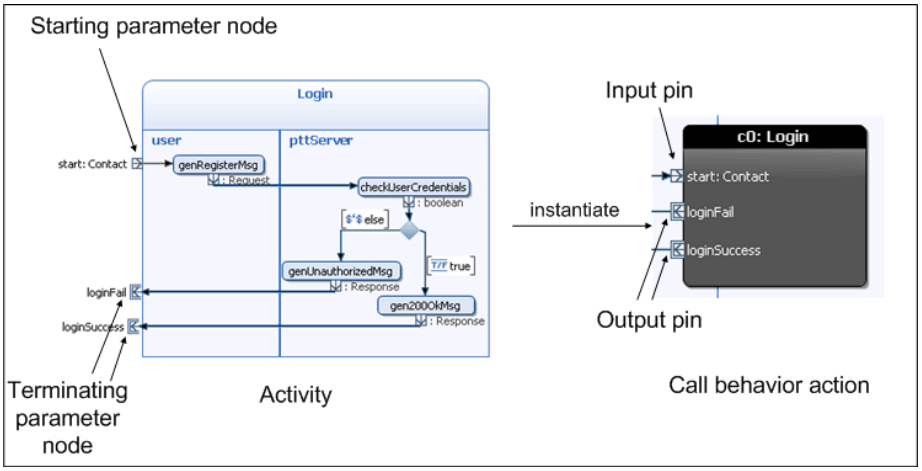


Figure 2-4: Activity and corresponding call behavior action example [13]

The last component of a building block is an External State Machine (ESM) which defines the external behavior of the building block. An ESM transition is the activity parameter node separated by a "/". The parameter node before the slash is called the *trigger* parameter node. The parameter node after the slash is called the *effect* parameter node. The ESM is triggered by the trigger parameter node. The effect parameter node is the consequence of the transition being triggered. States in ESM do not contain entry or exit actions [13].

2.1.1 Arctis and Ramses

Arctis and Ramses are tool suites implementing the SPACE method. Arctis [2] [10] is a plug-in to the Eclipse [6] development platform to provide an editor for a service specification by editing UML 2.0 collaborations and activities. Subsequently these building blocks are transformed into state machines by an algorithm [14] used by Arctis. When the state machines of the service are ready, Ramses [2] is used to generate executable Java code from them. The focus of our project is using Arctis to model a Push to Talk group communication in terms of building blocks which can be stored in the library for later use in related applications.

Building blocks are the major units in Arctis. They are the units of the service specification. Building blocks include participants, structure, and behavior. The significant feature of Arctis is that it supports the reusability of building blocks; because completed blocks are stored in the library for later use. Therefore, Arctis can increase the productivity of software developers. Arctis provides four types of building block:

Service collaboration or **Service** is the most common building block. It has more than one participant. The service collaboration is modeled as a UML 2.0 collaboration. The building block has an internal behavior and an external behavior [13].

Figure 2-5 shows an example of login service collaboration in a PTT system. It includes two participants: a user (using a PTT client) and a PTT server. The internal behavior of the login building block is the interaction between the user's client and the PTT server. Firstly, in order to login to the system the user's client executes the *genRegisterMsg* call operation action to generate a SIP REGISTER request and sends it to the PTT server. After the PTT server receives the message, it checks the user's credentials by executing the *checkUserCredentials* call operation action. The checking result of the operation then arrives at a decision node with two branches: true and false (or *else* branch). The true branch leads the PTT server sends a SIP 200 OK response to the user's client since the user is authenticated. The building block terminates via the output pin *loginSuccess*. On the contrary, the else branch leads the PTT server sends a SIP 401 Unauthorized response to the user's client. The building block then terminates via the output pin *loginFail*. The output pins *loginSuccess* and *loginFail* are alternative terminating parameter nodes.

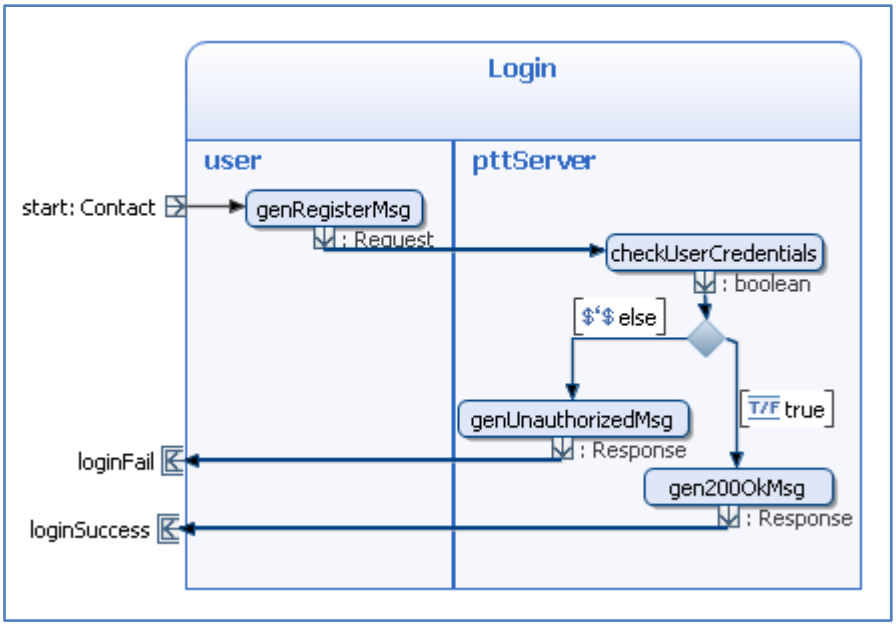


Figure 2-5: Login service example

Activity block or **block** is similar to a service, but it has only one participant with one activity and an ESM. Hence, a block does not have any UML collaboration [13]. Figure 2-6 shows an example of an activity building block named "Compare". It has two integer variables a and b and a call operation action named *compare* to compare the values of these variables. The compare method will return true or false depending on the values of the variables. Figure 2-7 shows the corresponding Java code of the Compare block. Each participant in a block is realized as a Java class. The declarations of variables and methods are all automatically generated based upon the composition of the block.

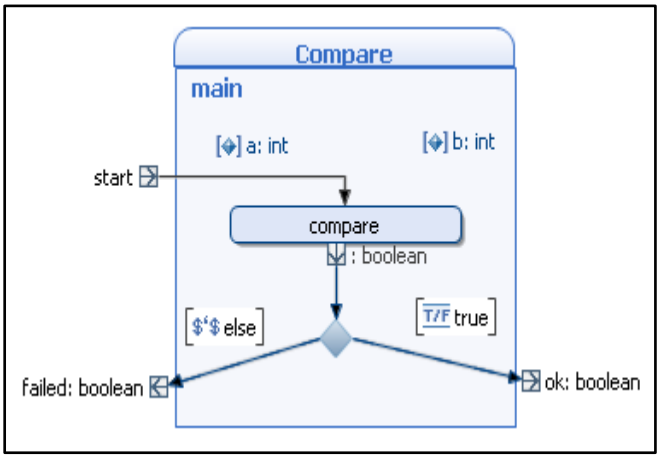


Figure 2-6: Activity block example

```
public class Compare {

    public int a;
    public int b;

    public boolean compare() {
    if (a==b)return true;
    return false;
    }
}
```

Figure 2-7: Java code of the Compare block

System collaboration or **system** is a special service. It is used to generate code for a service to execute it. A system includes some services and executes them, thus it does not have any input pin or output pin. Hence, a system does not have an ESM. Furthermore a system starts with an initial node and ends with an activity final node; while a service is triggered by an event and terminated by an output event.

Figure 2-8 shows an example of a system block to test the Compare block (described above) by generating code to invoke it. It starts with an initial node, then the *init* method is called to input the values for the two variables. The Compare block is placed after the init method. Depending on the result of the Compare block, one of two call operation actions is called to print out the conclusion; i.e., printing out whether the variables were equal or different in value. The Java code of the *TestCompare* system block is shown in Figure 2-9.

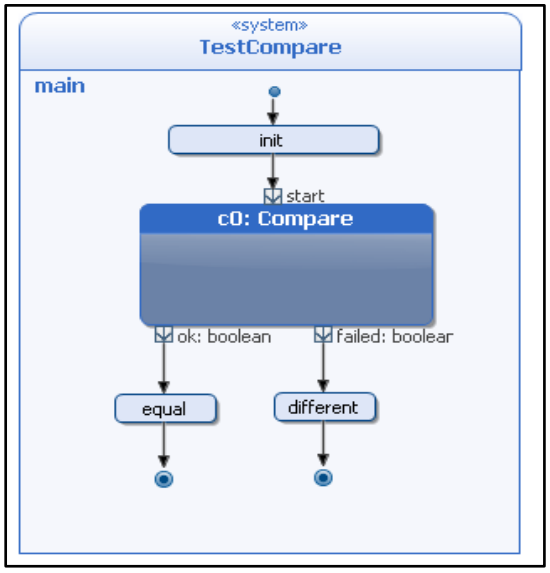


Figure 2-8: TestCompare system block

```
public class Main {

    public void init() {
        int a=3;
        int b=4;
    }

    public void equal() {
        System.out.println("a is equal to b");
    }

    public void different() {
        System.out.println("a is different to b");
    }
}
```

Figure 2-9: Java code of the TestCompare system block

Shallow block is a building block that is completely described by its own ESM, hence it does not have any UML activity. A shallow block contains only of parameter nodes and an ESM [13].

2.2 Push to Talk

Unlike regular phone calls which are full-duplex, Push-to-Talk (PTT) is a half-duplex service - providing an one-way voice group communication mechanism for mobile phone users. That means that once a group communication session is created, the service allows only one member of the group to speak and the others will listen to him or her until the user gives up the floor. When the user who has the floor speaks, his or her voice traffic is sent as packet stream. These packets are duplicated and sent to all recipients participating in this PTT session. In addition to group communication, PTT also supports one-to-one half-duplex communication between two users. In this case, a user who wishes to set up a PTT session with another user; simply selects an entry from his contact list and pushes the PTT button. In general, before speaking, a potential talker has to request the floor – this is done by pressing the PTT button on the mobile device. The button will be held down when this user speaks and released to give the floor to another participant in the session.

A PTT service was first introduced in the USA by the operator Nextel in August 1993 in their iDen network. The service has steadily grown since that time and now is deployed in 2.5G and 3G packet switched cellular networks, but it is called Push-to-Talk over Cellular (PoC).

The specification of PoC was defined by an industry consortium including Motorola, Nokia, Ericson, Siemens AG, and AT&T Mobility. The result of this collaboration is the set of PoC specifications approved as standards by the Open Mobile Alliance (OMA) [15]. All of these specifications are based on using an IMS architecture to implement PTT. In fact, PTT is the first IMS based service for mobile networks. The PTT service based on the OMA specifications utilizes a General Packet Radio Service (GPRS) network to stream media data [16].

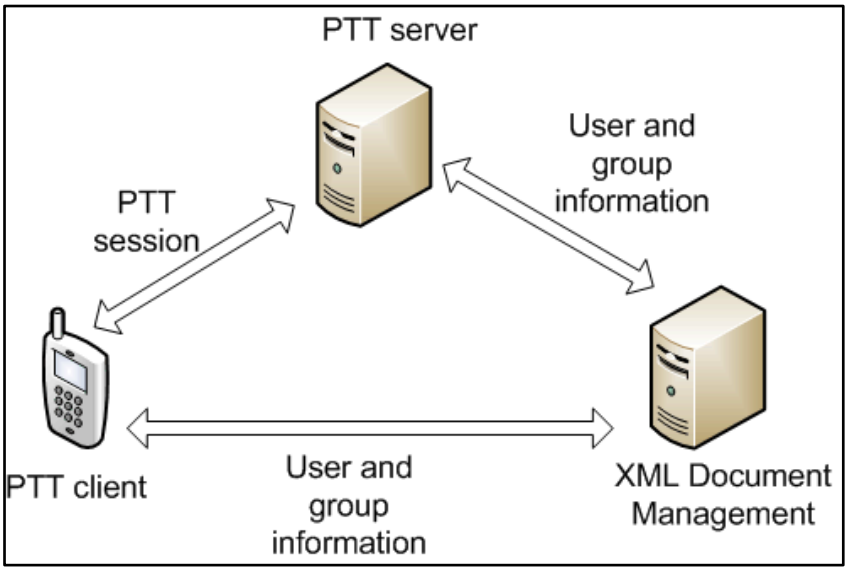


Figure 2-10: General PTT architecture

According to the OMA specification [17], the architecture of the PTT relies on PTT clients, PTT servers, and an XML Document Management server (see Figure 2-10). The PTT client is usually implemented as software in mobile phones. The PTT server controls one or more PTT sessions and transfers media data among the PTT clients. However, in an instant personal talk session, the sender may send the media data directly to the receiver since he or she knows the IP address of the receiver in the talk session initiation. In this case, the PTT server has the role of forwarding the talk request of the sender to the receiver. In a session the media data provided by the PTT service may be audio (e.g. speech, music), video, images, text, and files. The PTT server controls PTT sessions - from initiation of a new session, media transfer during a session, and termination of the session.

PTT session initiation and termination is controlled by using the Session Initiation Protocol (SIP). SIP is described in [18]. SIP does not convey media data. The information concerning the media capabilities of the participating SIP user agents is carried in a SIP message body using the Session Description Protocol (SDP) [19]. This SDP information is included in SIP messages during the session initiation and for modifications to an existing session. The Real-time Transport Protocol (RTP) [20] is used to transport media data. A companion protocol, the Real-time Transport Control Protocol (RTCP) is used to monitor the quality of an associated RTP session.

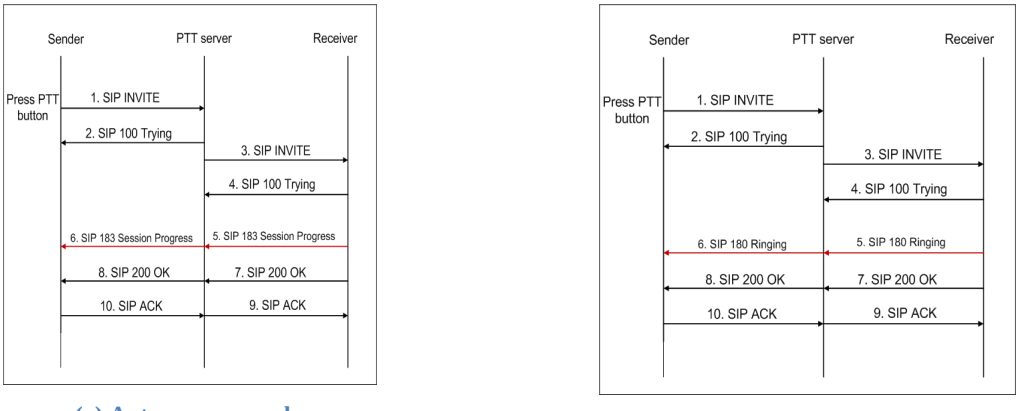
In figure 2-10, the XML Document Management (XDM) server implements a group of four functions: Shared List XDM Server (XDMS), Shared Group XDMS, Shared Policy XDMS,

and Aggregation Proxy. The Shared List XDMS manages user information. The Shared Group XDMS manages group member list and group information. The Policy XDMS manages the group's policy. The Aggregation proxy connects the PTT server and one or more XDMSs.

2.2.1 Signaling Protocol

PTT uses the Session Initiation Protocol (SIP) to initiate, terminate, and modify PTT sessions. SIP is a text based application layer protocol. SIP messages are generally wrapped in UDP packets for transmission over IP network (however, SIP can utilize TCP, SCTP, or TLS). These messages can be divided into requests and responses. Requests are sent by a client to the server, for example INVITE, REGISTER, REFER, NOTIFY, ACK, and BYE messages; whereas responses are sent by a server in response to requests. Examples of SIP response messages are 200 OK, 100 TRYING, 180 RINGING, and so on. Figure 2-5 illustrates the SIP message sequence to initiate a talk session. When the sender presses the PTT button, an INVITE message is sent to the incoming PTT server for the potential receiver in order to create a talk session. This distinction between client and server is important because it allows the server to provide interesting services: while the sender does not actually need to know the IP address of a potential receiver.

Depending on the configuration of PTT clients, there are two kinds of answer modes for incoming session invitation: automatic and manual answer mode. In the automatic mode, the receiver hears the voice from the sender without any notification or performing any action. In contrast, in the manual mode the receiver receives a notification and must decide to accept the call before hearing the voice of the caller. The core difference between these modes is the SIP messages exchanged during the establishment of the session. Figure 2-11 shows the details of the messages exchanged in these two modes. In this figure, when the receiver gets the SIP INVITE message, it sends a SIP 100 Trying response back to the sender through the PTT server. Depending on the answer mode configuration, the receiver either generates SIP 183 Session Progress (figure 2-11a) for auto answer mode or SIP 180 Ringing (figure 2-11b) while generating a tone to notify the receiver of an incoming call in manual mode.



(a) Auto answer mode

(b) Manual answer mode

Figure 2-11: Talk Session Initiation, with two modes [21]

The SIP message flow in a PTT session termination is shown in Figure 2-12. When the sender finishes talking, he or she releases the PTT button terminating the session. A SIP BYE message is sent to the receiver and a SIP 200 OK is the expected response to confirm session termination. After this exchange the media stream terminates.

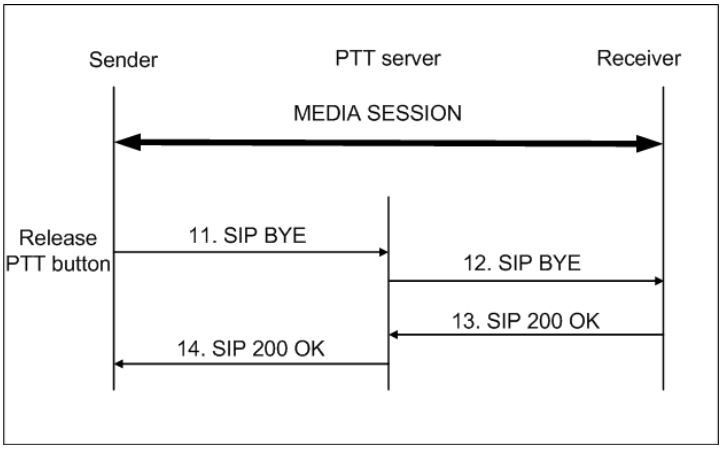


Figure 2-12: Talk session termination

There are several changes to the SIP message format when it is used in a PTT service in order to differentiate between a normal SIP call and a PTT call. When a PTT client registers for PTT service, the REGISTER message has a contact header containing a feature tag of the form [21]:

```
Contact:        sip:john.smith@ntnu.no;+g.ptt.talkburst="TRUE";
```

Figure 2-13: Contact Header of a SIP REGISTER request in PTT

The Contact header in all SIP responses includes the feature tag "+g.ptt.talkburst". When a PTT client generates a SIP INVITE message, the message has an Accept-Contact header of the form [21]:

```
Accept-Contact:      *;+g.ptt.talkburst="TRUE";require;explicit
```

Figure 2-14: Accept-Contact Header of a SIP INVITE request in PTT

When the SIP INVITE message is sent to the PTT server, the PTT server checks to see if the invited user is registered or not. If the invited user is registered with the feature tag true for PTT, then the PTT server continues processing the INVITE request; otherwise it rejects the request with a "480 Temporarily not available" response [21]. In addition to being sent in the SIP INVITE message, the Accept-Contact header shown in figure 2-14 is included in SIP all other SIP requests except the BYE request.

2.2.2 Talk Burst Control

After a PTT session has been established, the PTT server decides which participant is allowed to talk. Because PTT is a half-duplex service, only one participant can talk at a time. The process of controlling which participant has the right to talk is referred as talk burst

control[2] [4]. The right to speak is normally requested by pressing a PTT button. A talk burst is a flow of PTT speech from one PTT client to the other PTT clients in a group. When the PTT server receives a talk burst, it distributes it to all of other members of the session. The talk burst control message is carried in RTCP packets while the actual talk burst is carried in RTP packets.

The Talk Burst Control Protocol (TBCP) is used to request and grant the right to send a talk burst. Table 2-2 shows eight types of talk burst (TB) messages.

Table 2-2: Talk burst control messages

TB message	Sent by	Description
TB IDLE (TB_IDLE)	PTT server	Notify all PTT clients that no one is sending a talk burst at the moment, thus a TB_REQUEST may be granted
TB RELEASE (TB_RELEASE)	PTT client	Notify the PTT server that it has finished sending a talk burst.
TB REQUEST (TB_REQUEST)	PTT client	Request the PTT server to grant the right to send a talk burst.
TB GRANTED (TB_GRANTED)	PTT server	Notify the PTT client that it has been granted the right to send a talk burst.
TB TAKEN (TB_TAKEN)	PTT server	Notify all PTT clients excluding the PTT client that has the right to send a talk burst that another PTT client has been granted the right to send a talk burst.
TB DENY (TB_DENY)	PTT server	Notify the PTT client that its talk burst request has been denied.
TB REVOKE (TB_REVOKE)	PTT server	Revoke the use of the media resource for a long time from a PTT client to guarantee fairness. The PTT client replies an SIP 200 OK message
TB ACKNOWLEDGEMENT (TB_ACKNOWLEDGEMENT)	PTT client	Send an acknowledgement as required when it receives a TB message.

2.2.3 Transport Protocol

User Datagram Protocol (UDP) is used to transport RTP and RTCP packets. RTP is used to transport media data from the sender to the receivers [20]. Talk burst control messages between the PTT client and the PTT server are carried by RTCP [4].

[2] Talk burst control is a specific form of "floor control". It is a control mechanism to arbitrate the right to send speech in PTT. In addition to talk burst control, there is a media burst control which is a control mechanism to arbitrate the right to send media and multimedia.

RTCP is used for control information associated with an RTP flow or to provide feedback of the quality of service being provided by RTP, such as bytes sent, lost packets, jitter, round trip delay, and feedback. There are several types of RTCP packets: Sender Report (SR) packet, Receiver Report (RR) packet, Source Description (SD) RTCP packet, Goodbye (BYE) RTCP packet, and Application (APP) Specific RTCP packets. In a PTT application, RTCP APP is used for talk burst control while RCTP RR and SR are used for quality feedback [4] [23].

2.2.4 Push to talk communication

PTT can be used for various types of communication in order to meet the communication demands for a group of users. The main difference between these different types of communication is their group policy. A PTT group can be a pre-defined group with a restricted list of participating members or an open group that anyone can join. Therefore we can divide PTT into two types of communication: one-to-one or group communication.

One-to-one communication or Instant Personal Talk provides one-to-one voice communication between two users where only one user talks at a time. The caller controls the communication by pressing the PTT button and starting to talk. Depending on the configuration of the PTT client, the recipient may immediately hear the caller's voice without manually answering or the callee may manually answer the call as they would answer a normal phone call.

Group communication provides one-to-many communication in a group of PTT users. In a group communication session, there is only one user speaking and the rest of the group listens. When a PTT user receives a group invitation he or she can either accept or reject the invitation, depending on the user's preference. Group communication can be divided into many sub-types, including the following [16]:

- **Chat group talk** includes two sub-sub-types:
  - **Open chat group** to which any other users can be invited at any time. It allows anyone to join; and
  - **Restricted chat group** is for members only. The members can join this group at any time they wish during a talk session.
- **Instant group talk** is a PTT session for a pre-defined group. All the members of the group are immediately invited to join the session at the time of the initial call establishment. This behavior is different from chat group talk because users in chat group talk can join the group at anytime they wish, while an instant group talk allows members to join only *before* the start of a session.
- ***Ad hoc* instant group talk** is the same as an open chat group. The only difference is that a unique identifier for an open chat group has been created. The unique identifier of an *ad hoc* instant group talk is transient. *Ad hoc* instant group talks are created on the fly. A user invites other users to an *ad hoc* group talk by selecting them from a contact list or by typing their SIP-URI.

- **Instant personal alert** is used to alert the recipient when he or she misses a call in an instant personal talk session and requests the callee to call the caller back. An Instant personal alert is really useful because sometimes the recipient is unreachable, for example he or she is busy with other call or is offline. An instant personal alert enables the callee to know that they have missed a call and to call back at a more suitable time.

A PTT group is uniquely identified by a SIP URI which is generated by the XML Document Management server when the group is created. Hence, every member can connect to the group by using the group SIP URI. Group members are maintained in a group member list. Additionally, every user is also identified by an address which is either a SIP URI, for example sip:john.smith@ntnu.no or E.164 telephone numbers, for example +4791009171. This address is used to set up communication with this user via a SIP proxy for this address [24].

In a pre-defined group, all of the members are selected by the group owner/leader and only the pre-defined members can join the group, therefore use of the group is restricted to its members. A group communication session starts when the first member of the pre-defined group accepts a session invitation and the right to talk is granted to the initiator of the session.

In contrast, an open group is open for anyone to join and the group members can invite any other user to join the group at any time (even) during a talk session.

A PTT user can be a member of multiple groups at the same time. This is support by the simultaneous session feature of PTT. For example, Alice is a member of three groups: Boats, Operators, and Colleagues. She can receive voice messages from all of these groups and whenever she wants to talk to a particular group, she simply selects the group and presses her PTT button to start talking. However, Alice can only talk to one group at a time.

2.2.5 Group and List management

A Group and List Management Server (GLMS) is provided in a PTT system so that PTT users can manage group and contact lists. A user can create, update, retrieve, and delete a group or a contact list in his or her contact list through a graphic user interface [24].

The group identity, group member identity, and contact identity in the contact list of the user are stored and managed in the GLMS. The group identity is generated by the GLMS when the user creates the group. The group identity includes an indication of the group domain where the group has been created and the group name. The GLMS generates a unique SIP URI for the group when it is created and returns this group identity to the group member who created the group.

A contact list is used by the PTT client. This contact list includes the identities of the other PTT clients, groups, and contact lists. A contact list is allocated a unique SIP URI that is generated by the GLMS when the user creates the contact list.

3 System Requirement Analysis

In this chapter, some analysis results of a PTT system are presented in order to support the process of building blocks for PTT using the SPACE method. At the end of this chapter, some specific use cases are selected a description of model methodology of these cases is given. The modeling process in the following will be mainly based on these use cases.

3.1 User agents

In this section, we specify the functions of PTT user agents including PTT client and PTT server.

3.1.1 Push to Talk client

The PTT client is implemented by software running on mobile device used by each of the fish farmers. The features of PTT client to be modeled in our project are shown in Figure 3-1. The Graphical User Interface (GUI) helps the client interact with other PTT clients and offers three main functions: instant personal talk, instant group talk, and *ad hoc* instant group talk. The initiation and the termination of these types of sessions are implemented using the SIP protocol. The actual media data control is out of the project's scope.

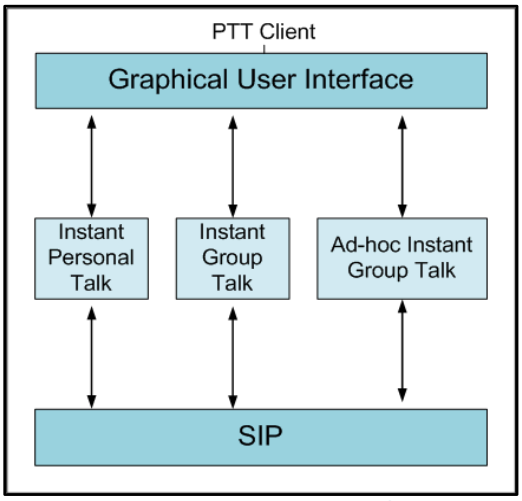


Figure 3-1: PTT client and desire functions

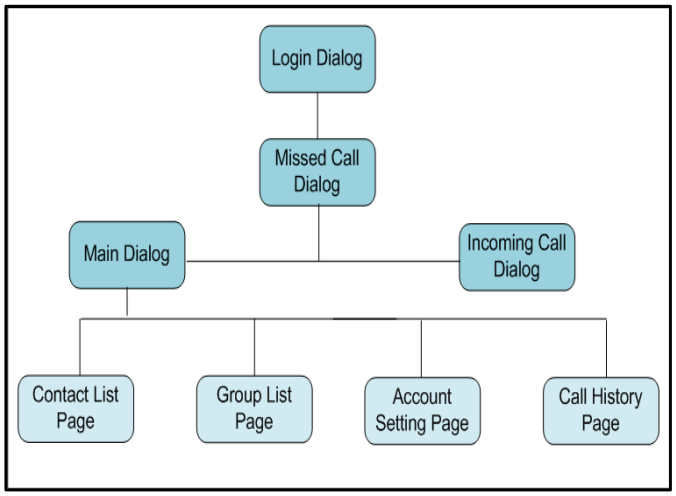


Figure 3-2: PTT client's GUI functions

The GUI consists of four main window dialogs: login dialog, missed call dialog, main dialog, and incoming call dialog (see Figure 3-2). We use the Java Swing API [5] to implement all of these dialogs in the Eclipse IDE [6]. The login dialog (Figure 3-3) is popped up after the PTT client program is executed. After the user enters the correct user name and password, the missed call dialog (Figure 3-6) is shown if the user has missed a call, otherwise the main dialog is initiated. In the missed call dialog, all the contacts who had given a missed call to the user are listed. The user can select a contact to call him or her. The incoming call dialog (Figure 3-4) is shown when the user receives an incoming call. When the incoming call dialog is shown a short ring tone is played. Then the user hears the voice of the talker

immediately. If the user can not handle the call, he or she can press the Cancel button in the dialog to deny the call. The incoming call dialog is closed when the call is finished or the user presses the Cancel button in the dialog.

The main dialog contains four pages: contact list page, group list page, account setting page, and call history page. The functionalities of these pages are described in Table 3-1.

Table 3-1: Main dialog pages

Main Dialog	Description
Contact list page (see Figure 3-7)	The contact list page shows all the contacts of the PTT user. In the contact list page, the PTT user can: • either select a user in the contact list or type his or her SIP URI address in a text box and press the Call button to make a call. • add a new contact by pressing New button. After that a new page is open to ask for the new contact information input. • delete a selected contact in the contact list by pressing the Delete button.
Group list page (see Figure 3-8)	The group list page shows all the groups that the PTT user currently participates in. In the group list page, the PTT user can: • Join a group by typing the group SIP URI in a text box and press the Join button • invite other user to join a selected group, the PTT user type the user SIP URI in a text box and press the INIVTE button. ; or enter a group SIP URI in a text box to join. • leave a group by selecting the group and then pressing the Leave button • call a group by selecting the group in the group list and press the Call button
Account setting page (see Figure 3-5)	The PTT user can set a PTT account in the account setting page.
Call history page (see Figure 3-9)	The call history page shows the most ten recent dialed calls and received calls of the PTT user.

Figure 3-3: Login dialog

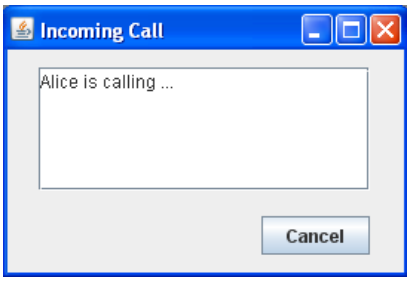


Figure 3-4: Incoming Call dialog

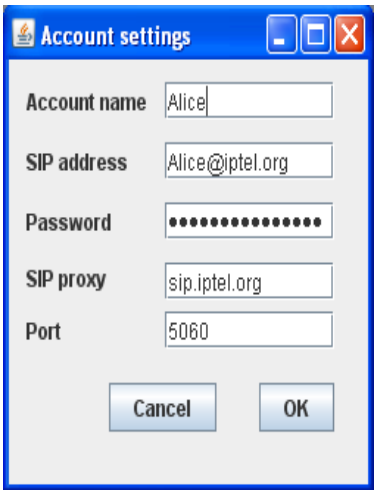


Figure 3-5: Account setting page

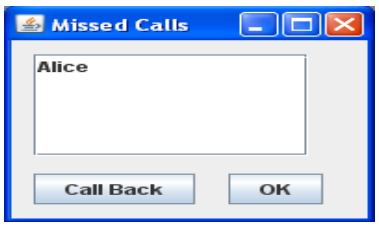


Figure 3-6: Missed Call dialog

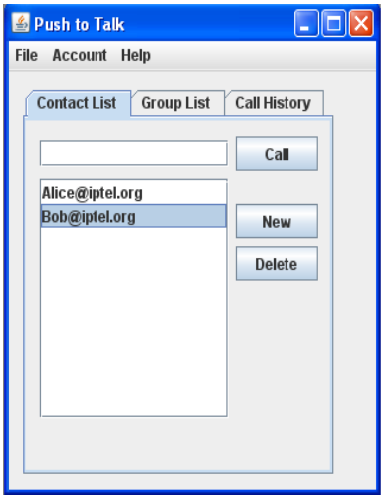


Figure 3-7: Contact list page

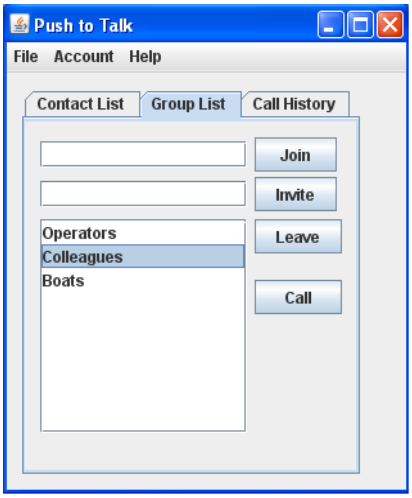


Figure 3-8: Group list page

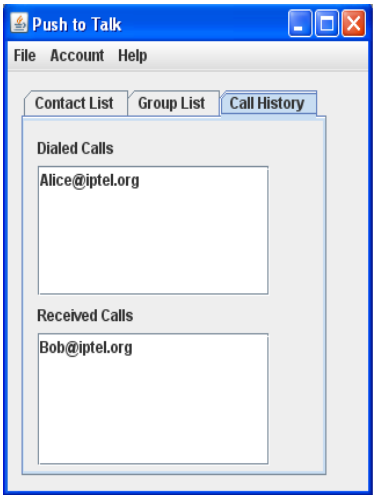


Figure 3-9: Call history page

A PTT client can support both automatic answer and manual answer mode. In this project we use manual answer mode as the default mode when modeling. When the sender pushes the PTT button to talk, the receiver is notified by a short ring tone which the user must manually answer before hearing the sender's voice.

3.1.2 Push to Talk server

In general, there are two approaches for deploying a PTT service including centralized and decentralize approach. In a centralized approach, each PTT client connects to a central PTT server that provides a PTT service. The PTT client sends media only once time to the PTT server, then the PTT server distributes the media to individual receivers. In the decentralized approach, every user agent is both a PTT client and a PTT server managing the media to the receiver. However, in a centralized approach, the PTT server acts as a central communication; it controls all SIP messages and talk burst control messages in a PTT session. This is an advantage of the centralized approach compared to the decentralized

approach [23]. Hence, in this project, we propose to use a centralized approach for a PTT service with a single central PTT server to handle all PTT sessions in the network.

As specified by OMA [17], the PTT server has two types of functions: participating and controlling functions. The participating functionality handles SIP signaling from the PTT client to create and terminate a PTT session; while the controlling functionality is responsible for manipulating the media stream in a PTT session. Based upon this study, we illustrate the idea of the PTT server functions in Figure 3-10.

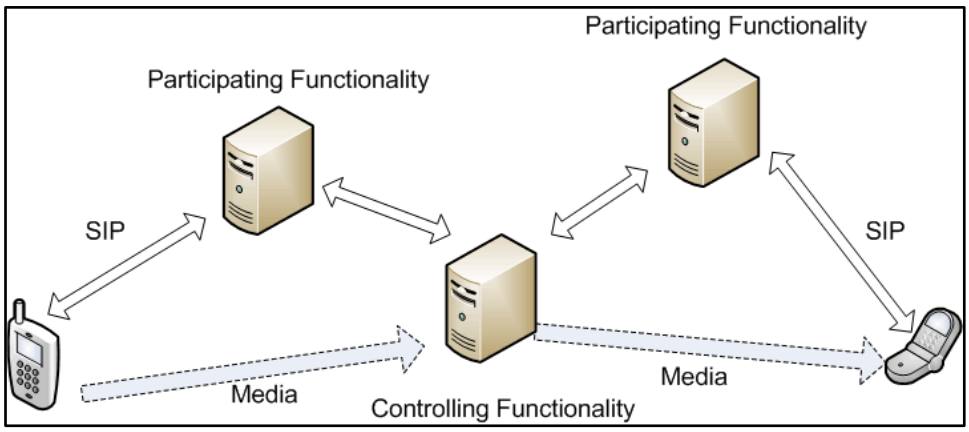


Figure 3-10: PTT server functions

In our project, the main function of the PTT server is to handle SIP messages (to initiate and terminate a PTT session) and talk burst control messages. We summarize the functionality of the PTT server to process SIP requests and responses in Figure 3-11.

- When the PTT server receives a SIP REGISTER request, it will check the user credentials. If the user is authenticated, it binds the user's identity with the corresponding IP address of the user's client and sends a SIP 200 OK response to the user client. If the user is not authenticated, the PTT server sends a SIP 401 Unauthorized response to the user client.
- When the PTT server receives a SIP request from a sender's client, such as an INVITE request, REFER request, and so on, it lookups the address of the receiver's client to forward the request to the receiver's client.
- When the PTT server receives a SIP response, it will forward the response to the destination.

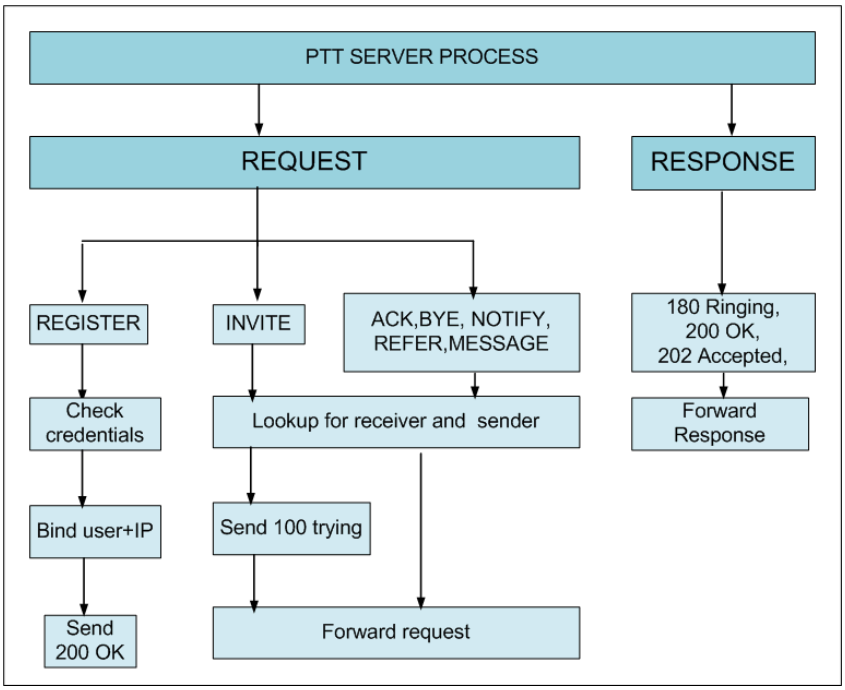


Figure 3-11: PTT server processes SIP requests and Responses

The process that the PTT server handles talk burst requests and responses are described in a specific case in the following section.

3.2 Use cases

While based on the OMA PoC specifications [22] [24], the principals of the SIP protocol [18], and document [22] our project focuses on three typical features of a push to talk system: instant personal talk, instant group talk, and *ad hoc* instant group talk. Various use cases regarding these three types of PTT communication are specified and analyzed in the following subsections. These cases include:

- User login
- Instant personal talk session initiation
- Joining a group talk session
- Group talk session initiation
- Adding users to a group talk session
- Leaving a talk session
- Talk termination

The idea presented in all of the message flow diagrams in the following subsections is inspired by the OMA PoC specifications [22] [24], the principals of the SIP protocol [18] , document [25], and document [21].

3.2.1 User login

A user needs to login to the system before making or receiving any calls from other users through the network. This is done when the user executes the PTT application in their mobile device and enters his or her user name and password via a Login Dialog. This causes the PTT client to generate a SIP REGISTER request that includes the feature tag "+g.ptt.talkburst" in the contact header indicating that the client is capable of supporting the PTT service. If the user enters an incorrect username or password, the PTT server sends SIP 401 Unauthorized message back to the client. On the contrary, if the user enters authentication credentials, he or she successfully register the user's client in the system (see Figure 3-12). To perform the registration the PTT server binds the client's identity to the device's IP address.

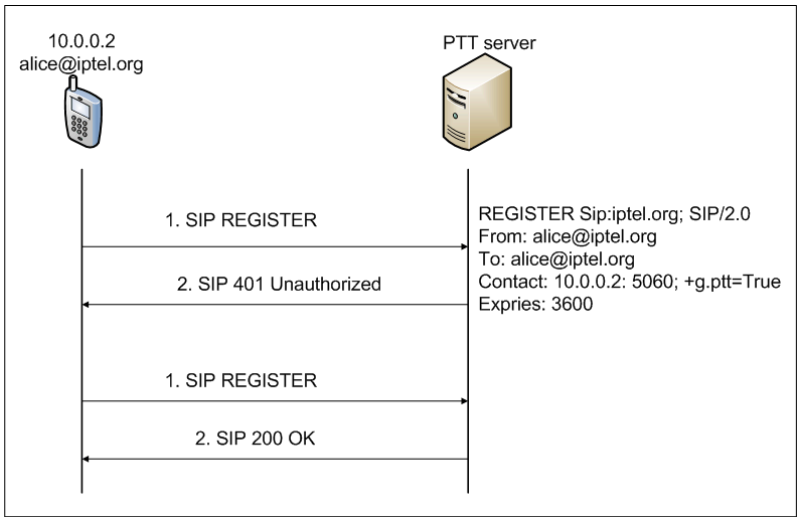


Figure 3-12: REGISTER message flow

3.2.2 Instant personal talk session initiation

This case begins when a user selects another user in his or her contact list in order to initiate an instant personal talk session. After the session is created, the initiator's speech is transported to the called party.

Figure 3-13 shows the INVITE procedure of an instant personal talk session. After the sender selects a receiver from his or her contact list, he or she presses a PTT button to initiate the talk session. At this time, a SIP INVITE message is generated and sent to the receiver. The SIP INVITE message in this case is considered as an implicit TB_REQUEST [4]. The Accept-contact header of the SIP INVITE message includes feature tag "+g.ptt.talkburst".

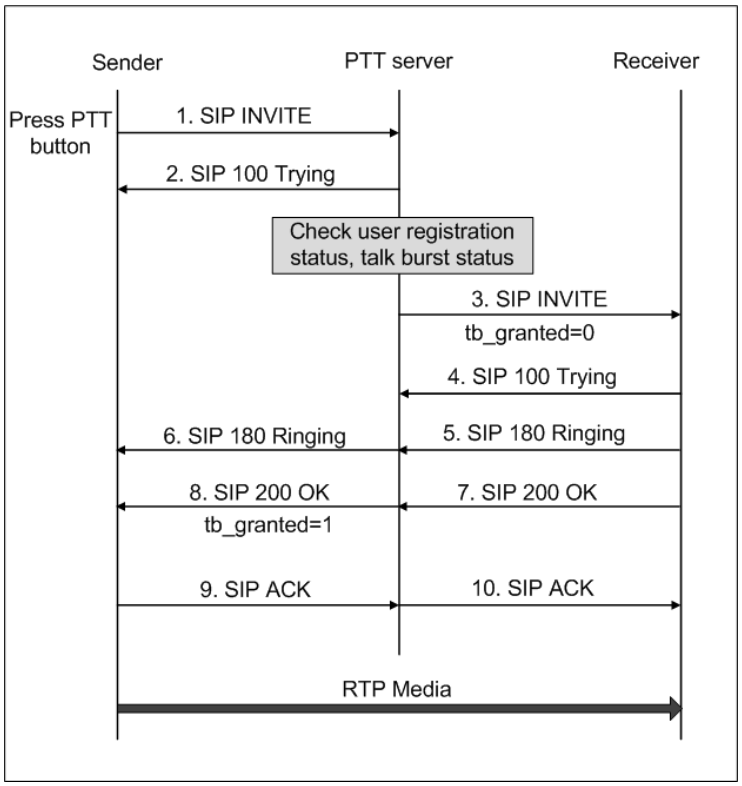


Figure 3-13: Instant personal talk session initiation

When the PTT server receives the SIP INVITE message, it checks the talk burst status of the target receiver. If this receiver's status is idle, then the PTT server forwards the INVITE message to the receiver with the parameter "tb_granted=0" in the SDP parameters to announce that the receiver is not allowed to speak. When the PTT server gets a SIP 200 OK message from the invited client it replies with a SIP 200 OK message with "tb_granted=1" in the SDP parameters to the sender to grant the sender permission to talk in this session [24] [26]. After this step, the speech is delivered via an RTP media stream.

If the invited user is unreachable, the sender sends an instant personal alert to the receiver asking for him or her to initiate an instant personal talk session between them in a more suitable time. The instant personal alert is not modeled in this project.

3.2.3 Joining a group talk session

This case describes the message flow when a member of an existing instant group talk wants to join/rejoin an instant group talk session or a PTT user wants to join an existing *ad hoc* instant group talk session. The message flows of this process are shown in Figure 3-14.

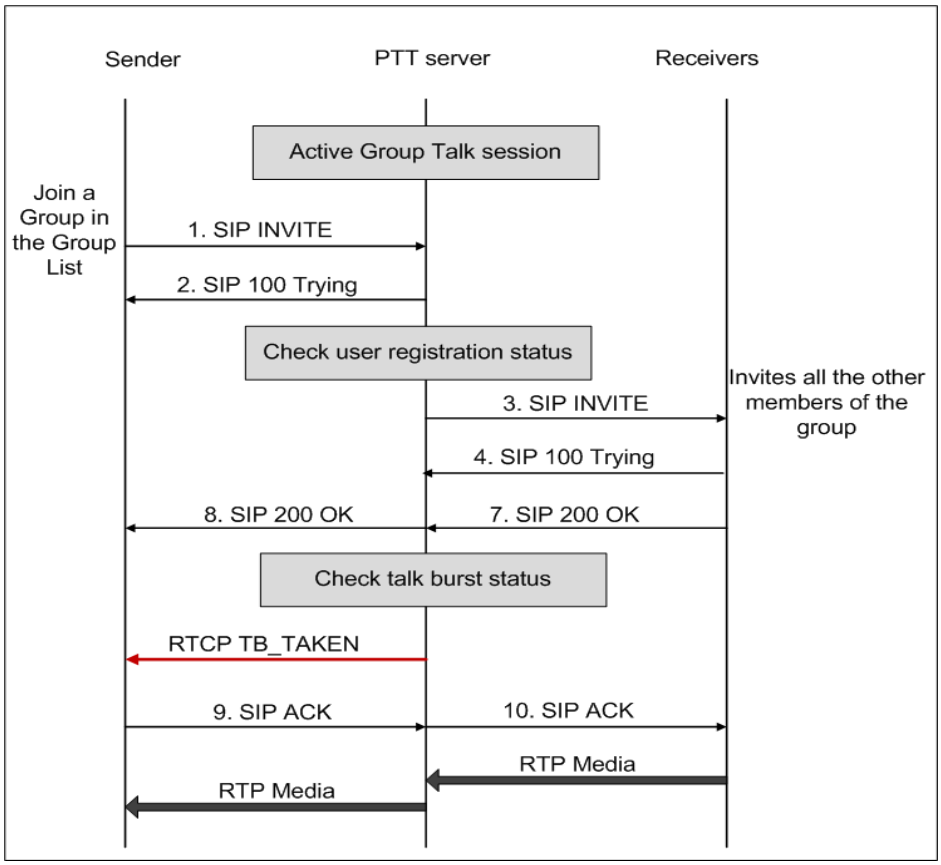


Figure 3-14: Joining a group talk

In order to join a group talk, the PTT user either selects the desired group in the group list or enters the group SIP URI in the text box in the group list page and presses the join button. After this process, a SIP INVITE message is generated and sent to all the members of the group. When the sender's client receives a SIP 200 OK message from the first recipient, the PTT server starts checking the talk burst status of this group talk session.

- If nobody is talking in the session, then the PTT server sends a TB_IDLE message to the sender's client.
- In other cases, the PTT server sends a TB_TAKEN message to the sender's client.

After receiving the TBC message, the PTT client replies with a SIP ACK to the PTT server to indicate that it is ready to transfer media. After the sender successfully joins the group, he or she starts receiving the media from the group if one member in the group is talking.

3.2.4 Group talk session initiation

This case describes the message flow when a member of an instant group talk or *ad hoc* instant group talk starts talking in his or her group. This process is shown in Figure 3-15. The PTT user initiates a group talk session by pressing a PTT button. The SIP INVITE request in this case is considered as an implicit TB_REQUEST [4]. Therefore, when the PTT server receives the SIP INVITE message, it checks the talk burst status of this group. If nobody is

talking in this session, the PTT forwards the SIP INVITE message to all the members of this group. In other cases, it informs the initiator that the group talk session initiation is fail.

After the SIP session is initiated in a group talk session, the PTT server sends a TB_GRANTED message to the PTT client and a TB_TAKEN message to all the other members of the group. Then the PTT client can start talking in this group session.

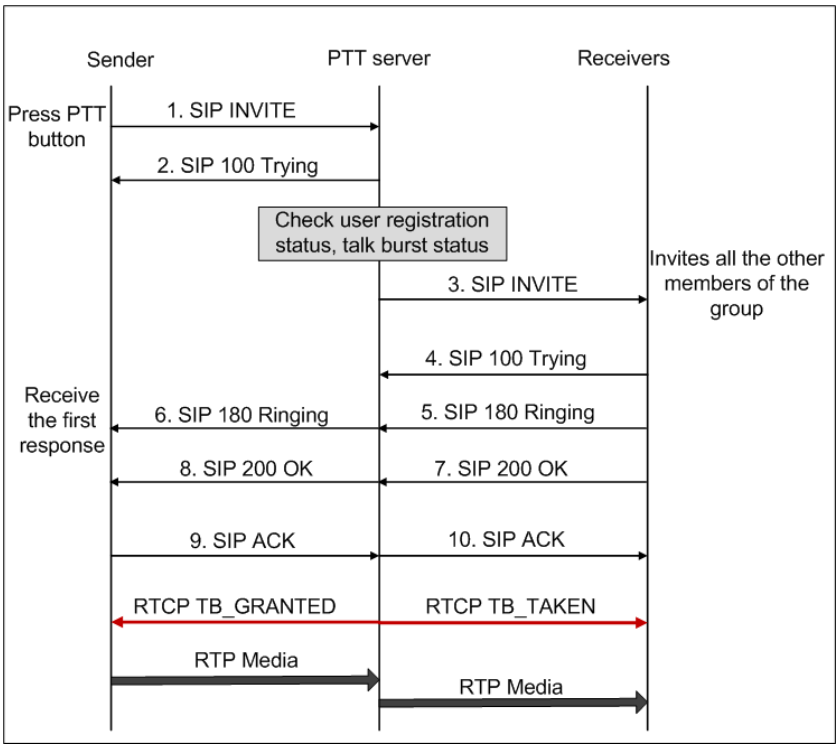


Figure 3-15: Group talk session initiation

3.2.5 Adding users to a group talk

This case describes the message flow when a member of an active *ad hoc* group invites another PTT user in his contact list to join the group. Alternatively, the owner of the instant group talk invites the group members to join an instant group talk session (see Figure 3-16).

If the invited user is already in the group list, then the PTT server returns a SIP NOTIFY message indicating that the invitation was successful. In this case, in figure 3-17, messages 5, 6, 7, and 8 are not necessary.

If the invited user rejects the invitation, he or she replies with an SIP reject response to the invitation. The message flow of this case is illustrated in figure 3-17. In order to simplify the model results, both of these two above exceptions in this use case are out of the project's scope.

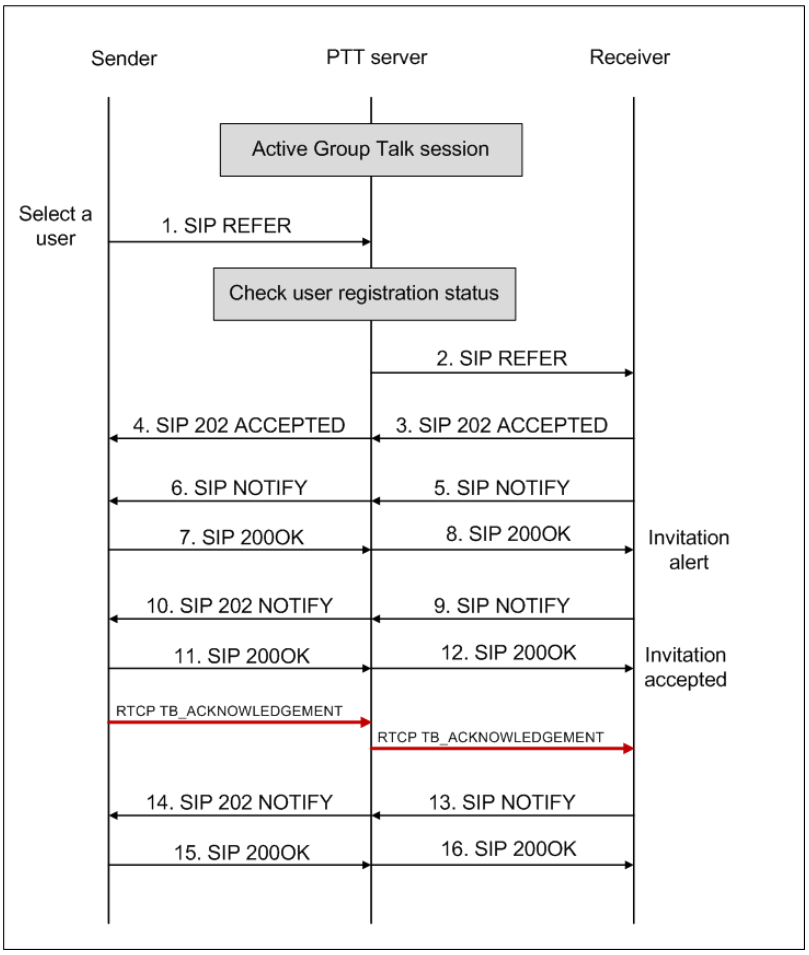


Figure 3-16: Add a user to an ad hoc instant group talk

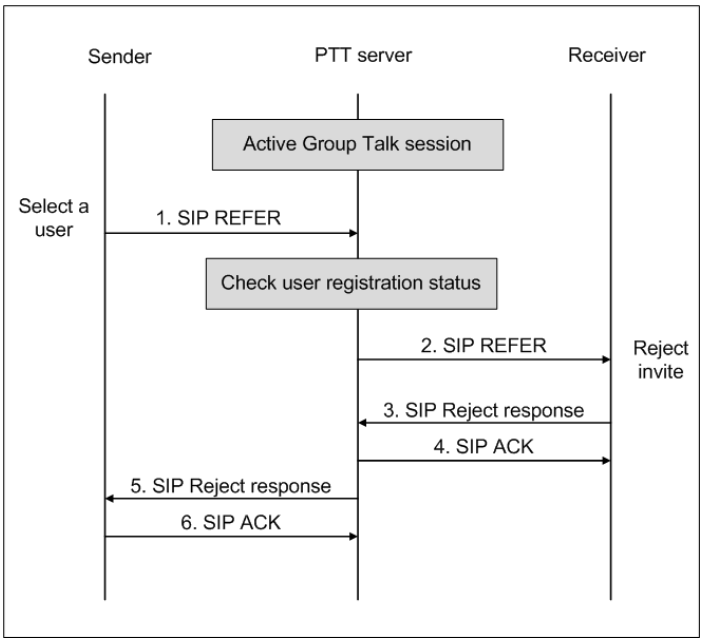


Figure 3-17: The receiver rejects the invitation

3.2.6 Leaving a talk session

This case describes the message flow when a user participating in an instant groups talk, or an *ad hoc* instant group talk wants to leave the talk session. The remaining group members continue their session. The message flow for this case is illustrated in Figure 3-18. The PTT client sends a SIP BYE message to the PTT server in order to leave the group. After the PTT server replies with a SIP 200 OK message to the PTT client, then the PTT client successfully leaves the talk session. There are no TBC messages exchanged in this case, as there is no continuing session – hence no need for talk burst control for the client leaving the talk session.

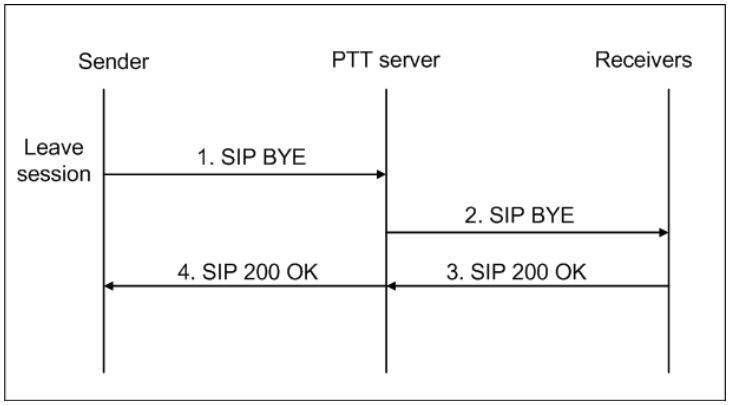


Figure 3-18: Leave a talk session

3.2.7 Talk session termination

This case describes the message flow when a user finishes talking and releases the PTT button in an instant personal talk, instant group talk, or *ad hoc* group talk (see Figure 3-19). After the PTT user releases the PTT button, the PTT client sends a TB_RELEASE message to the PTT server to indicate that he or she has finished talking. Then the PTT server sends TB_IDLE messages to all the group members to announce that nobody is currently talking in the group talk session. Therefore, any member of the session can ask for the floor to talk.

When the PTT client receives the TB_IDLE message, it generates a SIP BYE request and sends it the PTT server. The PTT server then forwards the request to all the remaining group members who reply with a SIP 200 OK response in a next step.

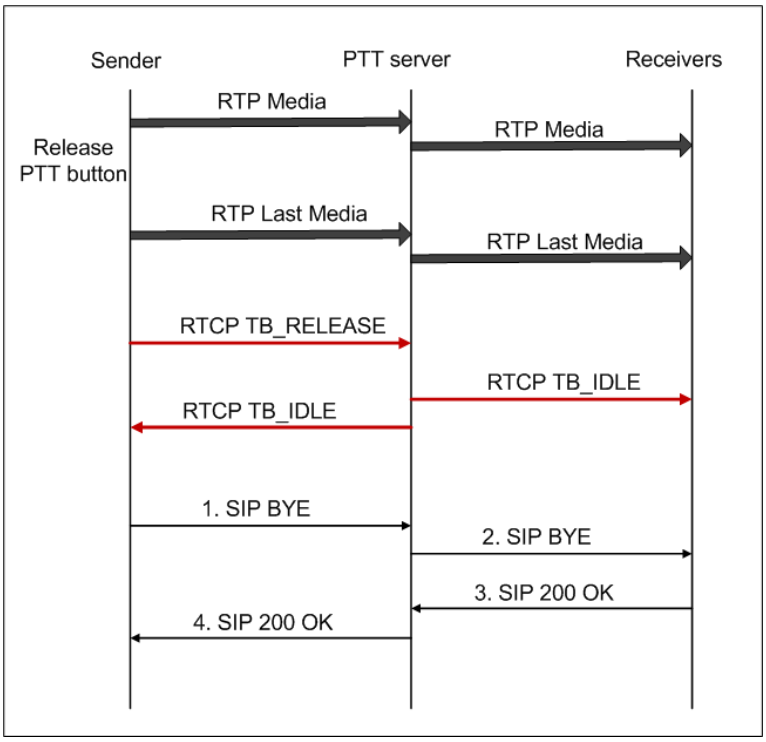


Figure 3-19: Talk session termination

3.3 Integration of message flows in Arctis

The methodology of creating building blocks for a PTT service in our project follows these steps:

- Identify the activity participants of each building block.
- Analyze the collaborations between these participants.
- Specify the behavior of each building block.

Based upon the use cases analysis (described above) we define that the participants of the building blocks includes: PTT client and PTT server. A PTT client can be in role of a sender or a receiver. The detail analysis of this step is given in section 4.1 in chapter 4. In our project, the collaborations between these two participants consist of four services: login, instant personal talk, instant group talk, and *ad hoc* instant group talk. Except the collaboration login the remaining collaborations include several sub-collaborations defined in section 3.2 of this chapter. The collaboration analysis is given in section 4.2 in chapter 4. The last step of the project's methodology is done in chapter 5.

In the third step, according to the above use cases the behavior of each building block is the exchanging messages between participants. Before exchanging a message, the corresponding participant has to generate the message. The participant generates the message by executing a call operation action in Arctis. The call operation action is a Java method generating the message. Then the generated message is passed along an object flow to the destination

participant. The message reaches the destination participant by arriving at an activity node in the destination participant.

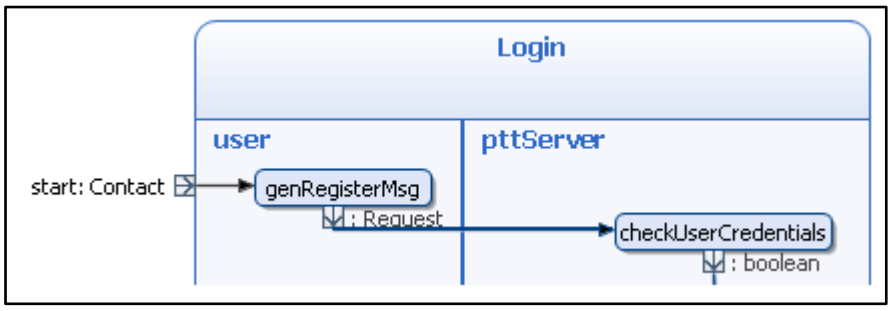


Figure 3-20: Exchanging a REGISTER message between a user and a PTT server

Figure 3-20 shows an example of exchanging a REGISTER message between a PTT user and a PTT server in a login service. In order to login into the system, the user has to send an INVITE message to the PTT server. An example of a SIP REGISTER message format is described in Figure 3-21.

REGISTER	sip:129.241.209.224;lr SIP/2.0
Call-ID	7556f2f2222de1365b8c9c0f0a583541@129.241.128.250
CSeq	1 REGISTER
From	<sip:alice@129.241.209.224> ;tag=4fa3
To	<sip:alice@129.241.209.224>
Via	SIP/2.0/UDP 129.241.128.250:6740;branch=z9hG4bKnashds7
Max-Forwards	70
Contact:	<sip:129.241.128.250:6740>; +g.ptt.talkburst;
Content-Length	0

Figure 3-21: SIP REGISTER header [22]

```
public Request genINVITE(){
        Address destinationAddress = myAddressFactory.createAddress
                        ("sip:"+destination+";lr");
        javax.sip.address.URI myRequestURI=destinationAddress.getURI();
        Address contactAddress=myAddressFactory.createAddress
                        ("sip:"+myIP+":"+myPort);
        ArrayList viaHeaders = new ArrayList();
        ViaHeader myViaHeader = yHeaderFactory.createViaHeader
                         (myIP,myPort,"udp","z9hG4bKnashds7");
        viaHeaders.add(myViaHeader);
        MaxForwardsHeader myMaxForwardsHeader =
                        myHeaderFactory.createMaxForwardsHeader(70);
        CallIdHeader myCallIdHeader = mySipProvider.getNewCallId();
        CSeqHeader myCSeqHeader = myHeaderFactory.createCSeqHeader
                                (1L, "REGISTER");
```

```
        SipURI fromURI = myAddressFactory.createSipURI(myID, myProxy);
        Address fromAddress = myAddressFactory.createAddress(fromURI);
        FromHeader myFromHeader = myHeaderFactory.createFromHeader
                                (fromAddress, "456248");
        ToHeader myToHeader = myHeaderFactory.createToHeader
                               (fromAddress, null);
        Request myRequest = myMessageFactory.createRequest
                    (myRequestURI,"REGISTER",myCallIdHeader, myCSeqHeader,
               myFromHeader, myToHeader, viaHeaders, myMaxForwardsHeader);

        return myRequest;
}
```

Figure 3-22: Java method of the call operation action *genRegisterMsg*

The call operation action *genRegisterMsg* in the participant user is responsible for generating a SIP REGISTER message. The Java method of the call operation action *genRegisterMsg* is shown in Figure 3-22. The call operation action *genRegisterMsg* returns the REGISTER message and the message is carried along an object flow (colored in blue). The message reaches the PTT server by arriving at the call operation action *checkUserCredentials*.

The proposed PTT service is run on the Connected Objects Operation System (COOS) platform [27]. The platform provides an overlay network for communication between components in the network infrastructure shown in Figure 1-1 in chapter 1. Components can send and receive data as *messages* to/from other components on the COOS platform instead of using TCP/IP transport directly. The current implementation of the COOS platform uses a linkstate routing protocol on each COOS instances to find the shortest path for routing message to the destination [28].

The messages using in our project's implementation are SIP messages and RTCP messages. The SIP messages are for the SIP protocol used in the project. The RTCP messages are for the talk burst control protocol. Currently, there are no APIs and frameworks provided by the COOS platform for a SIP communication and RTCP streams. Therefore, for the modeling process in our project's implementation, we assume to use Java JAIN SIP API [29] to build the SIP messages and Java Multimedia Framework (JMF) [30] to build the RTCP messages. When the messages are generated, they are transported using the COOS message bus to the destination.

When the executable state machines generated from the building blocks, they can run on a normal Java platform.

4 Object and collaboration analysis

In our approach, the PTT service will be composed using building blocks in Arctis. Then Arctis will generate state machines that can be transformed into executable Java code by Ramses. This section describes the process of building blocks for a PTT service. Firstly, an analysis to define participants in a PTT service is given in section 4.1. Secondly, based upon the system requirements specified in chapter 3, several collaborations for a PTT service are defined in section 4.2.

4.1 Object-oriented analysis

The purpose of our object-oriented analysis is to define a number of participants/objects in a PTT service. Obviously PTT clients and a PTT server are objects in a PTT. Both of these types of objects are active because they have their behaviors. In a group communication session, a PTT client object can be either a sender object or a receiver object or a receiver object with a multiplicity attribute from 0 to many. The attributes of these objects are described in Table 4-1.

Table 4-1: Object attributes

Attribute	Type	Description
PTT client		
myURI	SipURI	The SIP URI of the PTT client
myName	String	The name of the PTT user
myIP	String	The IP address of the PTT client
myServer	String	The IP address of the PTT server
myPort	int	The port of the PTT server providing the service
status	int	The login status of the PTT client
PTT server		
myIP	String	The IP address of the PTT server
myName	String	The name of the PTT server
myPort	int	The port of the PTT server providing the service

4.2 Collaborations of Push to Talk

We depict in Figure 4-1 the system collaborations used to model a PTT service. The collaboration role is rectangular; whereas the collaboration use is elliptical. It describes the structure of the whole service including participants and the relations between them. The PTT service includes two participants: PTT clients and a PTT server. A PTT client can be either sender or a receiver in a group communication session. The sender has a multiplicity of one, while receiver has a multiplicity of many because in PTT communication only one person is allowed to speak and others can only listen to him or her. PTT server has a multiplicity of one since there is a single central PTT server responsible for handling all communication in our system design.

The proposed PTT service provides three types of group communication: instant personal talk, instant group talk, and *ad hoc* instant group talk. In addition, before starting communication, each PTT client is required to login into the system. All of these behaviors are illustrated in corresponding collaborations with the collaboration roles bound to the corresponding participants.

In Figure 4-1 the collaboration c0 models the instant personal talk session between the sender and the receiver. Likewise, the collaborations c1and c2 describe the instant group talk and *ad hoc* instant group talk, respectively. Collaborations c3 and c4 are instantiations of the *login* collaboration.

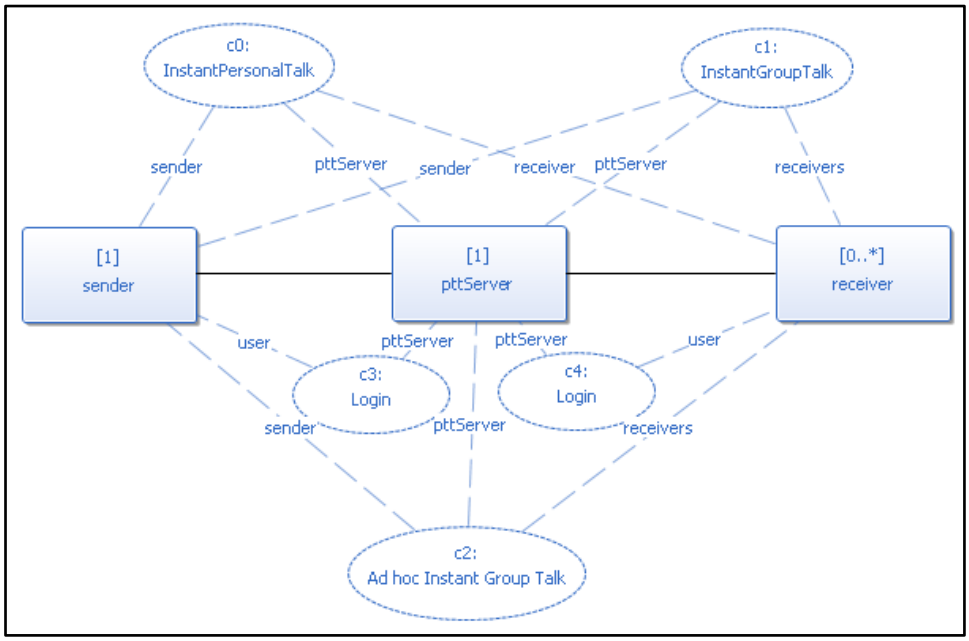


Figure 4-1: System collaboration

The **instant personal talk collaboration** c0 in Figure 4-1 is divided into two sub services expressed in two collaborations: *instant personal talk session initiation c0* and *talk termination c1* (see Figure 4-2). The former collaboration initiates an instant personal talk session. The later collaboration terminates the instant personal talk session.

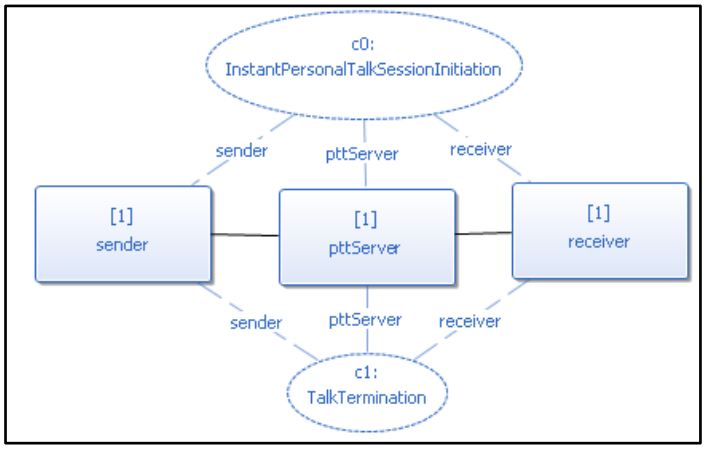


Figure 4-2: Instant personal talk collaboration

The **instant group talk collaboration** c1 in Figure 4-1 has four sub-collaborations, as shown in Figure 4-3. It includes: *Group talk initiation c0* collaboration, *talk leaving c1* collaboration, *talk termination c2* collaboration, and *group joining c3* collaboration. The initiation and the termination of an instant group talk are done in the *group talk initiation c0* collaboration and the *talk termination c2* collaboration, respectively. During the talk session, a PTT member of the instant group talk can join or leave the session. These behaviors are encapsulated in the *group joining c3* collaboration *and talk leaving c1* collaboration, respectively.

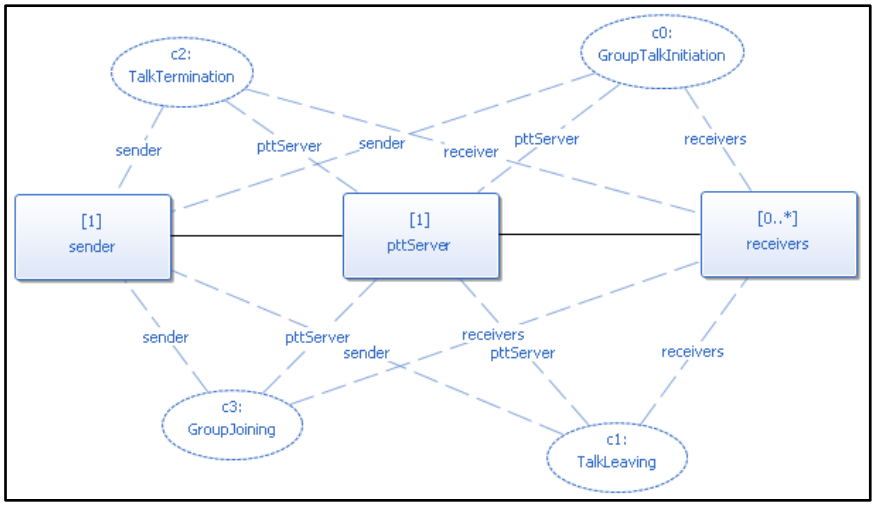


Figure 4-3: Instant group talk collaboration

Likewise, the *ad hoc* instant group talk collaboration in Figure 4-1 consists of those sub-collaborations in the instant group talk collaboration. In addition to those sub-collaborations, the *ad hoc* instant group talk collaboration has one sub-collaboration more which is *user add c4*. The sub-collaboration *user add c4* allows a PTT user to add another PTT user in an *ad hoc* instant group talk session. All sub-collaborations of an *ad hoc* instant group talk are depicted in Figure 4-4.

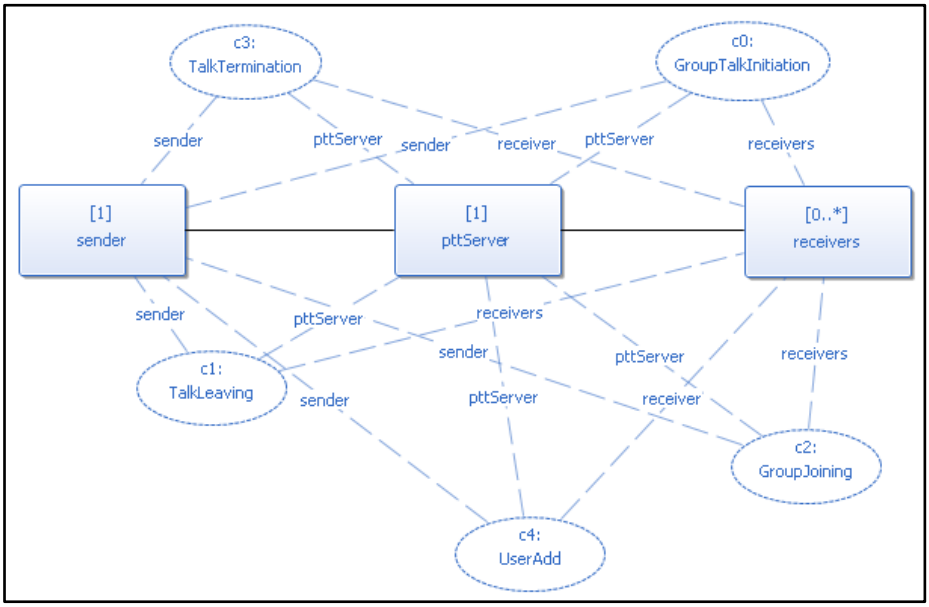


Figure 4-4: Ad-hoc instant group talk collaboration

Since the four common sub-collaborations between the instant group talk collaboration and the *ad hoc* instant group talk collaboration can be instantiated in the super-collaborations and the *ad hoc* instant group talk collaboration has one sub-collaboration more than the instant group talk we analyze the *ad hoc* instant group talk collaboration in the modeling process presented in chapter 5.

Each PTT communication session always includes at least a sender and a receiver. If the system is centralized, then a PTT server is needed. In addition, PTT can support several types of group communication (discussed in section 2.2.4) which can be specified in collaborations as described above. We consider the collaboration-oriented specification in the SPACE method is really helpful to develop a PTT service because of the following reasons. Firstly, the process of composing collaborations following the system requirements is very easy in Arctis. Secondly, the divisions of a PTT system into participants and collaborations enable a software developer to quickly understand of the system (at least at an overview level). Thirdly, the model's structure supports a high reusability of the individual collaborations. A software developer can create a system specification by selecting related collaborations from a library, instantiating them and composing them according to a system description [31]. Lastly, when the behavior of the collaboration is specified, it can be automatically transformed into executable state machines from which executable code is generated. The specification of the system's behavior is presented in the following chapter.

5 Modeling service behavior with GUI building blocks

In this section, we describe and analyze the model results of three sub-services provided by our proposed PTT service: login, instant personal talk, and *ad hoc* instant group talk with our PTT client GUI presented in section 3.1.1. The modeling process includes three steps:

- Modeling the GUI blocks.
- Modeling the service blocks.
- Connecting the GUI blocks with the corresponding service blocks.

The GUI is designed for the PTT client; hence, a GUI building block is an activity block with one participant (i.e., a PTT client). A GUI building block listens to all possible events generated by the PTT clients. The GUI building blocks perform the following tasks:

- Show the user interface components; for example, a form with several text boxes and buttons, etc.
- Listen to the events produced by the user interface components; for example, a button is clicked, a text box has input, etc.
- Get the values of the user interface components for further processing; for example, get the user name and password in a login form, etc. The result of this process may be to show another user interface.

A service block includes UML collaborations and activities. Hence the second step includes both UML collaborations and activities modeling. The UML collaborations illustrate the structure of the service. They consist of participants of the service and the sub-services between them (as described in the previous chapter). The UML activities illustrate the behaviors of the service. The behavior of the activity is described by *token* flow. In this chapter, we describe the procedure for modeling the behavior of three sub-services of a PTT system: login, instant personal talk, and *ad-hoc* instant group talk.

Finally, when a GUI building block and a related service are ready, they are composed together to make a complete service with a user interface. They can be added (by dragging & dropping) to the related participants in the service, e.g. the GUI building block is in the client side, and the service is in both the client and the server side. The GUI building block is automatically colored blue; while the service block is colored black. The following sections show the result of our modeling for a PTT service.

5.1 Login service

Figure 5-1 depicts the modeling of the behavior of the PTT login service building bock with the login GUI building block. The GUI building block c1 is an instantiation of the activity block *LoginGUI* in Figure 5-2. It is placed by dragging and dropping in the collaboration role *user*. The Login building block c0 is an instantiation of the service block *Login* in Figure 5-3. It is a cross-cut between the collaboration role *user* and collaboration *pttServer*.

The activity of the building block *LoginService* in Figure 5-1 is active when the token passes the starting parameter node *start*. Then the token arrives at the input pin *start* of the login GUI building block c1. The behavior of the login GUI block c1 is then triggered in the collaboration role *user* of this building block. The detail behavior of the building block c1 is shown in Figure 5-2.

As shown in Figure 5-2, when the behavior of the login GUI building block is active, the call operation action *showLogin* of this login GUI building block is executed to open the login dialog described in Figure 3-3 in section 3.1.1. Then the login GUI block waits for a signal EVENT_LOGIN which is triggered when the user enters user name, password, and presses the OK button to login. When the signal EVENT_LOGIN is received, the login information is encapsulated and sent to the output parameter node *sendUser*. The activity of the login GUI building block is still active after sending the login information via this node since it waits for an event for a further processing, such as opening the main dialog or showing a notification if the login is unsuccessful. Hence, the output parameter node *sendUser* is a streaming node.

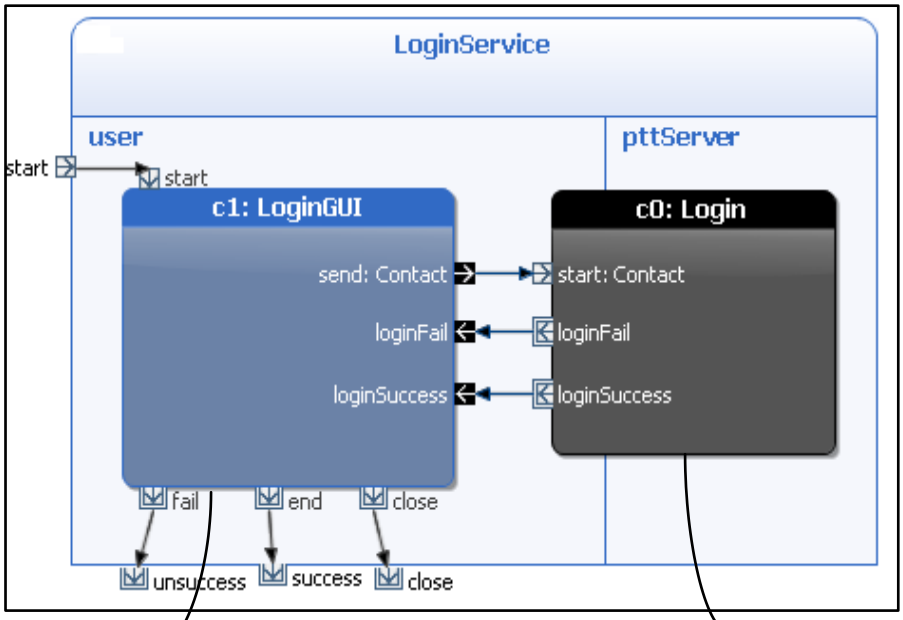


Figure 5-1: Login service building block

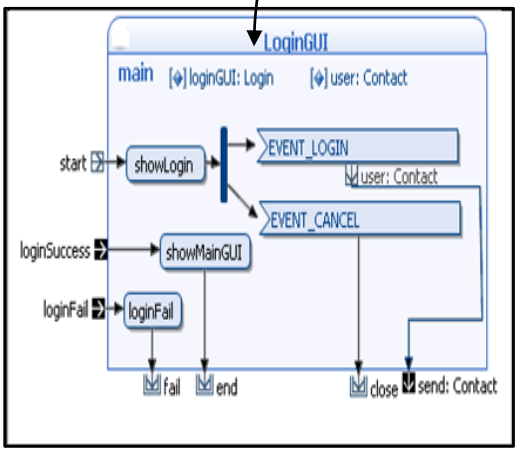


Figure 5-2: LoginGUI building block

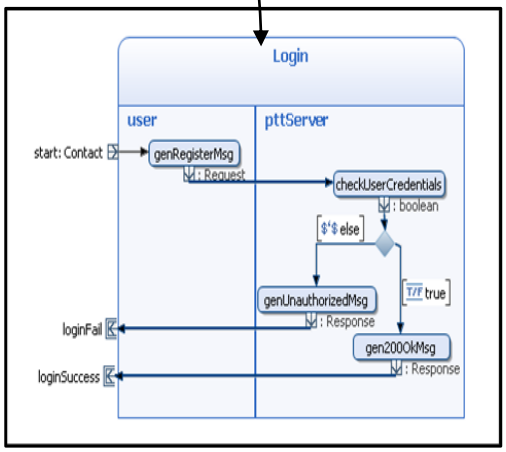


Figure 5-3: Login building block

Then the login information is transferred from the login GUI building block c1 to the login building block c0 via the input pin *start* of the building block c0 (see Figure 5-1). The activity of the login building block c0 is then active. The detail activity of the login building block c0 is shown in Figure 5-3.

In Figure 5-3, when the login building block is active, the token arrives at the call operation action *genRegisterMsg*. This call operation action is then executed to generate a SIP REGISTER request holding the user's login information. Then this SIP REGISTER request is delivered to the PTT server in an object flow. When the PTT server receives the request at the call operation *checkUserCredentials* node, it checks the user credentials encapsulated in the request by executing this call operation action:

- If the checking process returns true, then the PTT server generates a SIP 200OK message in the call operation action *gen200OkMsg* and then sends this response to the

PTT client. The activity of the login block is terminated by passing the terminating parameter node *loginSuccess* (see Figure 5-3).

In Figure 5-1, when the token arrives at the output pin *loginSuccess* of the login building block c0, this token is forwarded to the login GUI building block c1 via the input pin *loginSuccess* of the building block c1.

In Figure 5-2, when the login GUI building block receives an input event *loginSuccess*, the call operation action *showMainGUI* of this building block is executed to open the main dialog described in Figure 3-7 in section 3.1.1. Then the activity of the login GUI building block is terminated via the terminating parameter node *end.*

In Figure 5-1, when the token arrives at the output pin *end* of the login GUI building block c1 this token passes to the terminating parameter node *success* of the building block *LoginService.* Then overall activity of the building block *LoginService* is terminated.

- If the checking process returns false, then the PTT server generates a SIP 401 Unauthorized response and sends this response to the user's client. The activity of the login building block terminates via the terminating parameter node *loginFail* (see Figure 5-3).

  In Figure 5-1, when the token passes the output pin *LoginFail* of the building block c0, it arrives at the input pin *LoginFail* of the building block c1.

  In Figure 5-2, when the GUI building block receives an input event *LoginFail*, it executes the call operation action *LoginFail* to show a login fail notification to the user. Then the activity of this building block is terminated at the terminating parameter node *fail.*

  In Figure 5-1, when the token arrives at the output pin fail, it goes to the terminating parameter node *unsuccess*. Then the whole activity of the building block *LoginService* is terminated.

If the user closes the login dialog or presses the Cancel button on the login dialog, the login GUI building block is triggered by the signal *EVENT_CANCEL*. The building block is then terminated on the terminating parameter node *end* (see Figure 5-2). Then in Figure 5-1, the whole activity of the LoginService is terminated via the terminating parameter node *close.*

The three parameter nodes *fail*, *end*, and *close* of the login GUI building block are the alternative terminating parameter nodes. By passing one of these two parameter nodes, the activity of the login GUI building block is terminated. Likewise, *unsucess*, *success*, and *close* are three alternative terminating parameter nodes of the building block *LoginService* in Figure 5-1. The activity of the building block *LoginService* can terminated when the token arrived at one of these nodes.

5.2 Instant personal talk service

The instant personal talk collaboration has two sub-collaborations: talk session initiation c0 and talk session termination c1, as shown in Figure 4-2. In this section we model an instant personal talk service with the contact view building block. The GUI of the contact view is described in Figure 3-7 in section 3.1.1. The detail of the contact view building block is given in section 5.2.1. The internal activities of the talk session initiation collaboration and the talk session termination collaboration are described in section 5.2.2 and 5.2.3 of this chapter. Finally, the activity instant personal talk service is presented in section 5.2.4.

5.2.1 Contact view building block

Figure 5-4 shows the internal details of the contact view building block for the user interface interaction in an instant personal talk session. When a user selects a contact in the contact list page of the PTT client GUI, the contact view building block gets started. The selected contact information is encapsulated in a Contact type defined by a Java class. Before receiving the EVENT_PUSH signal for pushing the Call button, the selected contact information is saved to the variable *receiver* of the building block in the call operation action *set receiver*. After the Call button is pushed, the variable *receiver* is read in the call operation action *get receiver* to pass the data to the terminating parameter node *call*. The activity of the building block is still active by passing this node, thus the parameter node *call* is a streaming node.

If the input event *initiationFail* of the contact view block contains data, the *initiationFail* call operation action is called to show a notification of the session initiation fail for the initiator. Then the flow activity is terminated by going to the terminating parameter node *fail*.

When the talker finishes speaking, the Call button is released. That means that in the contact view building block the signal EVENT_UNPUSH is received. The activity of the contact view building block is then terminated with the terminating parameter node *releasePTTbtn*.

The call operation action *playsound* of the contact view block is executed to play a short ring tone when the input event *ringing* of the building block is triggered. Following this the token is removed. However, the whole activity of the building block is still active until the talker release the Call button that means the input event *fisnishSpeaking* is triggered.

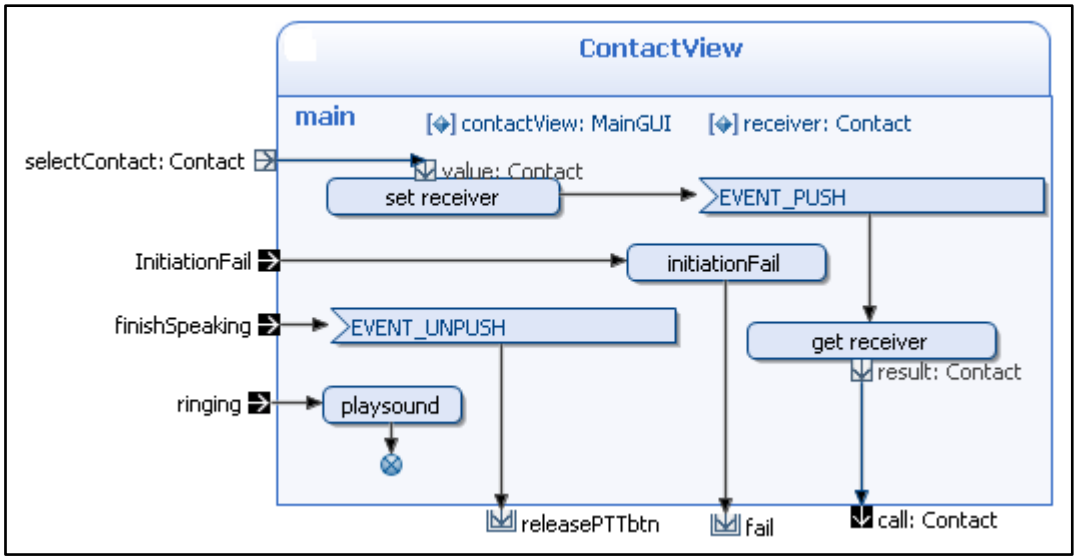


Figure 5-4: Contact view building block

5.2.2 Instant personal talk session initiation collaboration

The instant personal talk session initiation c0 activity describes the behavior of the instant personal talk session initiation collaboration in Figure 5-5. After the talker selects a contact on his or her contact list, the talker presses the Call button to make an instant personal talk session with the targeted contact. As input on the collaboration, the activity gets the receiver contact information which is packed in the Contact object via the starting parameter node *call*. The input token then arrives at the call operation action *genINVITE* on the participant *sender*. In the call operation action *genINVITE*, a SIP INVITE request is generated. After that, the SIP INVITE message is transported to the PTT server in an object flow as an input of the call operation action *gen100TRYING*. The PTT server invokes the information encapsulated in the input request to generate a SIP 100 TRYING response by executing the call operation action *gen100TRYING*. After that, the token arrives at a fork node. This fork has two outgoing edges:

- The first edge is an object flow carrying the generated SIP 100 TRYING response to the sender. The response reaches the sender by arriving at the call operation action *recv100TRYING* to confirm that the PTT server already received its SIP INVITE request. Following this the token of the activity on this edge is removed. The whole activity of the collaboration keeps continuing on the other edge.
- The second edge takes the token to the call operation action *checkRegStatus* in the PTT server. In this operation, the PTT server checks if the sender is able to create an instant personal talk with the corresponding receiver.
    - If the checking process return true, the PTT server will generate a SIP INVITE request based on the SIP INVITE request it received from the sender by executing the call operation action *genINVITE*. Later, this SIP INVITE request is passed to the receiver in an object flow.

- If the sender is not able to create a session, the PTT server generates a SIP 401 Unauthorized response and sends to the sender in an object flow. Then the sender terminates the whole activity of the collaboration via the terminating parameter node *InitiationFail*. Then in the GUI of the sender, a session initiation fail notification is shown since the input event *InitiationFail* of the contact view building block is active.

The process is carried on similarly in the rest of Figure 5-5. Each call operation action is used to process one corresponding SIP message. Each SIP message is carried in an object flow to the destination by arriving at an activity node of the destination.

When the receiver produces the output event *ringing*, a short ring tone is played on the GUI of the receiver. This event is triggered the input event *ringing* in the contact view building block as described above. In case of the receiver, when it produces the output event *receiverRinging*, the incoming call dialog is shown as described in Figure 3-4 in section 3.1.1.

The activity is terminated in both the sender and the receiver via the terminating parameter nodes *readyToSendMedia* and *readytoreceiveMedia.* After this, the media can start streaming.

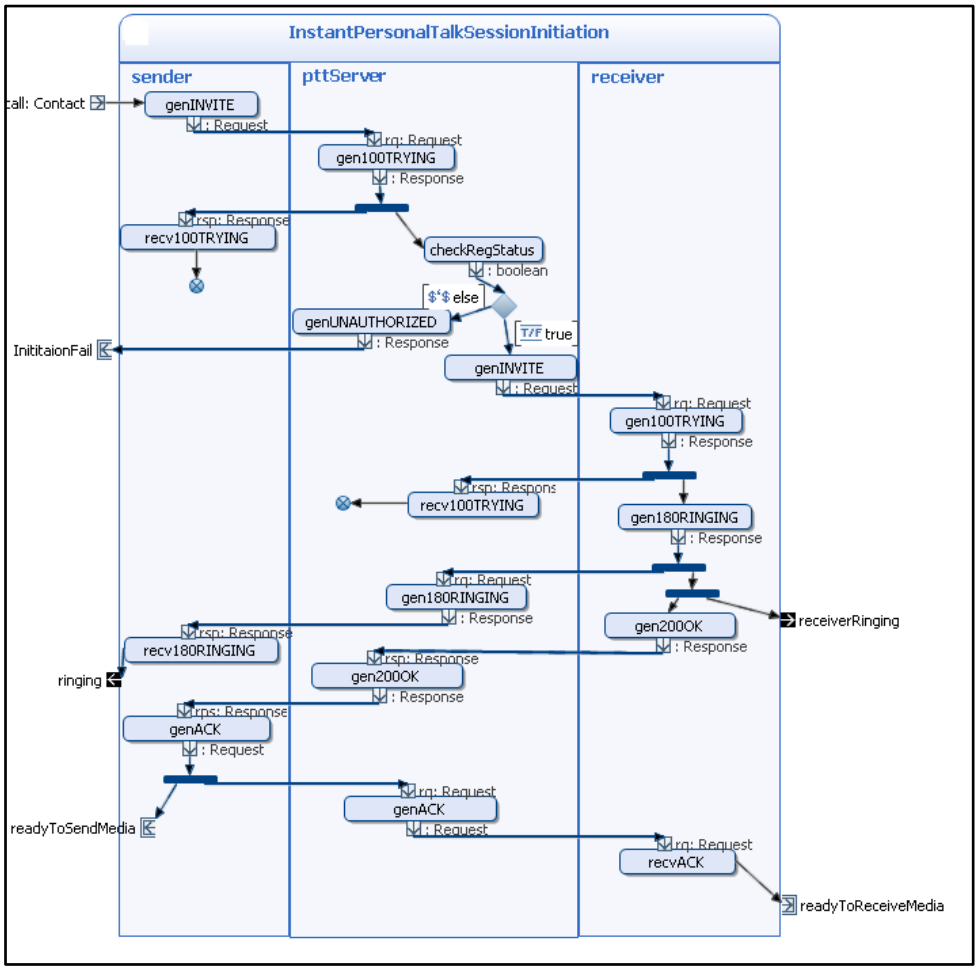


Figure 5-5: Instant personal talk session initiation

5.2.3 Talk termination collaboration

The activity of the talk termination collaboration is activated when the sender releases the PTT button to finish speaking. The activity diagram for the talk termination collaboration is shown in Figure 5-6. In this case, the participant receiver of the activity has the multiplicity attribute of one since the instant personal talk is an one-to-one communication.

The activity starts when data is available in the starting parameter node *releasePttbtn*. From the starting node, the token arrives at the call operation action *genTbRELEASE* in the sender. One object flow carrying a TB_RELEASE message is sent to the PTT server and arrives at the PTT server via call operation action node *genTbIDLE*. The call operation action *genTbIDLE* processes the received TB_RELEASE message to generate a TB_IDLE message for both the sender and the receiver.

After arriving at the call operation action *genTbIDLE*, the token enters a fork node where the TB_IDLE message is duplicated twice on two edges. One token follows an outgoing object flow to the receiver then from which the activity flow is ended. The second token arrives at the sender. After the sender receives the TB_IDLE message, it generates a SIP BYE message by executing the call operation action *genBYE*. Then following the object flows, the generated SIP BYE message is delivered to the receiver. The receiver afterward generates a SIP 200 OK response which is carried to the sender by the object flows. The activity is terminated in both the sender and the receiver via the terminating parameter node *senderEnd* and *receiverEnd*.

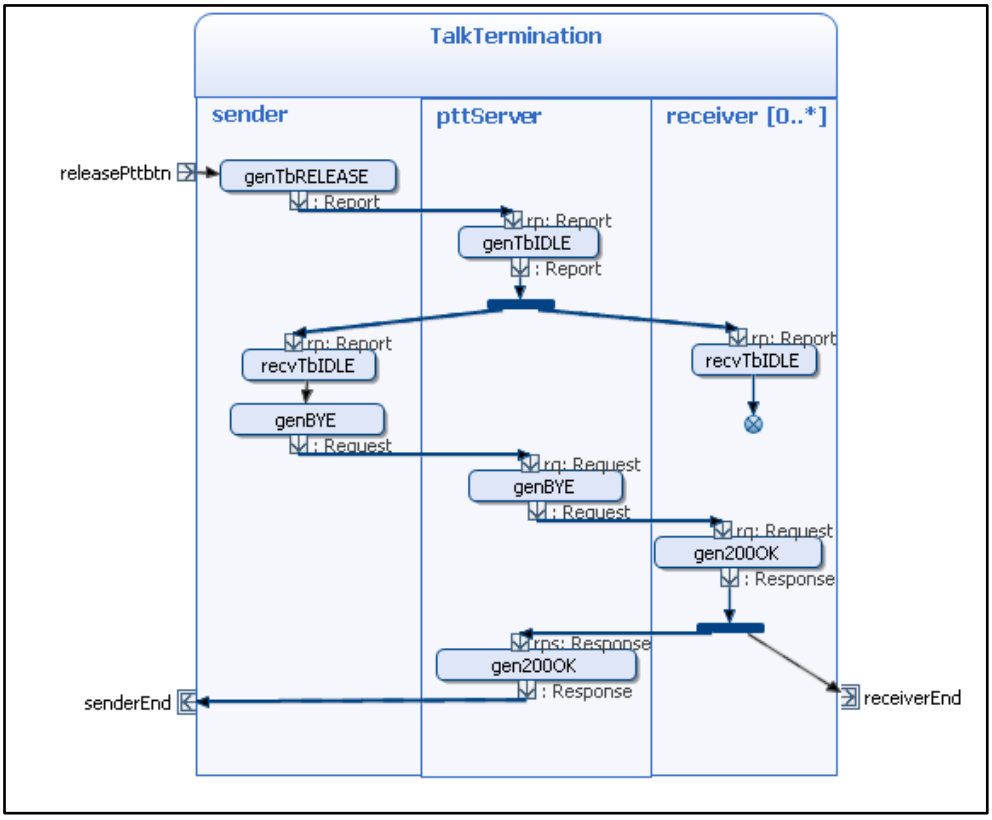


Figure 5-6: Talk termination session

5.2.4 Instant personal talk service

The screen short from Arctis of the activity diagram describing the behavior of the instant personal talk service is shown in Figure 5-7. This activity diagram connects four building blocks: *contact view c2, instant personal talk session initiation c0, media stream c3,* and *talk termination c1*. The *media stream* c3 building block is not yet implemented in our project, but it is shown in Figure 4-10 as an illustration.

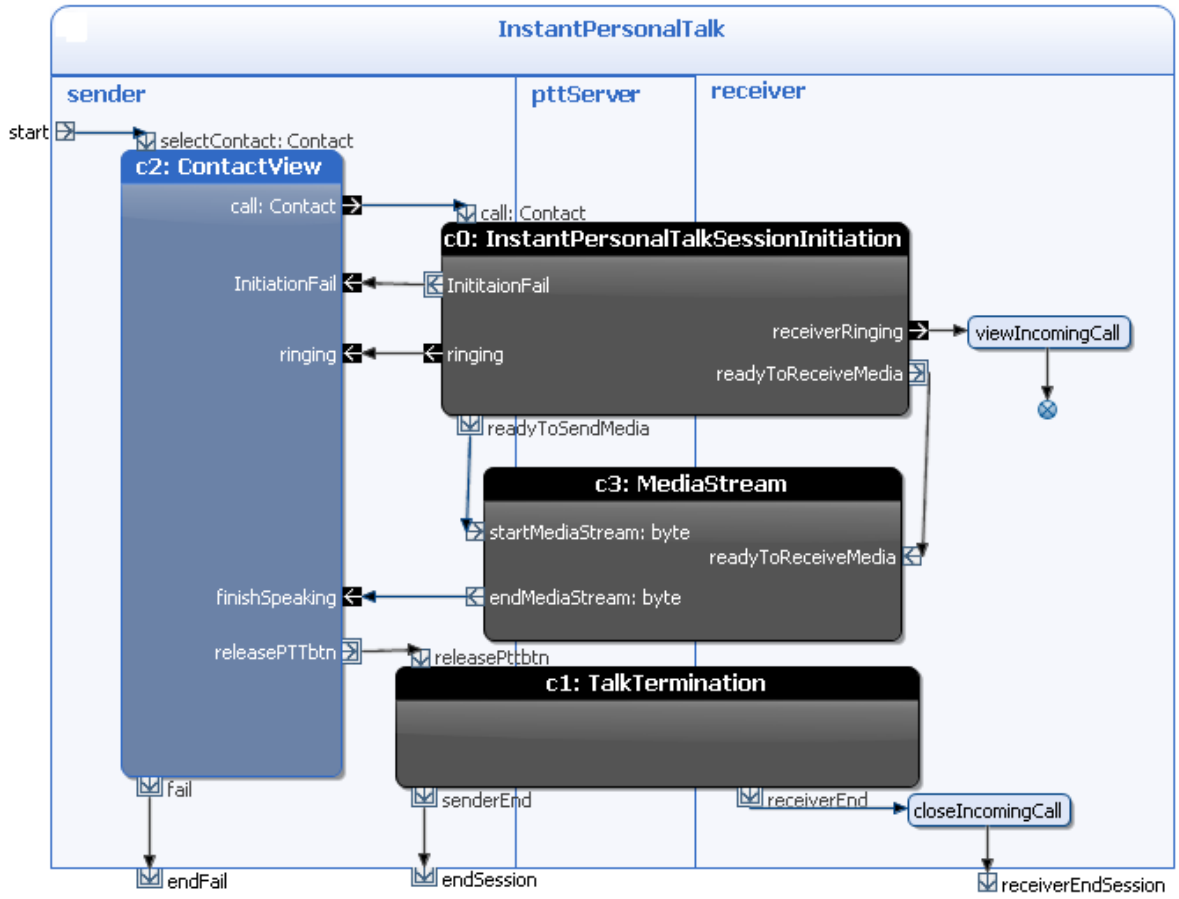


Figure 5-7: Instant personal talk activity diagram

When the sender selects a contact in the contact list to create an instant personal talk session, the building block c2 is active. When the sender pushes the PTT button, the output pin *call* of building block c2 transfers the required data to the input pin of the building block c0. Then the sender starts building block c0 with the PTT server and the corresponding receiver. The internal activity of the collaboration c0 is shown in Figure 5-5 as described above.

When the building block c0 needs information for the fail initiation output pin, the token goes to input pin *InitialFail* of the building block c2 to trigger this input event of this building block. Then the token arrives at the output pin *fail* of the building block c2 as described in section 5.2.1. After this pin, the token passes to the terminating parameter node *endFail* of the instant personal talk service *InstantPersonalTalk*. Then the whole activity of the service is terminated.

When the token arrives at the output pin *ringing* of the building block c0, it is forwarded to the input pin ringing in the contact view building block to play a sound in the sender's contact view block.

When the sender finishes speaking, the PTT button is released. Then the output pin *releasePTTbtn* of the contact view building block provides the data for the input pin of the building block *talk termination c1*. Hence, the building block c1 gets started. The detail behavior of collaboration c1 is shown in Figure 5-6. The building block c1 terminates via the output pins *senderEnd* and receiverEnd in the sender and receiver, respectively. On the sender side, before the whole activity of the service is terminated, the call operation action *closeIncomingCall* is executed to close the incoming call dialog in the receiver's GUI.

5.3 *Ad hoc* instant group talk service

The *ad hoc* instant group talk collaboration has five sub-collaborations as shown in Figure 4-4. To reduce the size and the complexity, we assume that all the sub-services are established successfully as described in section 3-3. With the same type of collaboration roles, more behavior in the *ad hoc* instant group talk service can be input in later.

5.3.1 Group view building block

The building block *GroupView* is shown in Figure 5-8. The purpose of this building block is to receive the user's input from the group list page of the user interface (see Figure 3-8). The user can create an *ad hoc* group talk session, invite a new user to join a group, leave a group, or join a group.

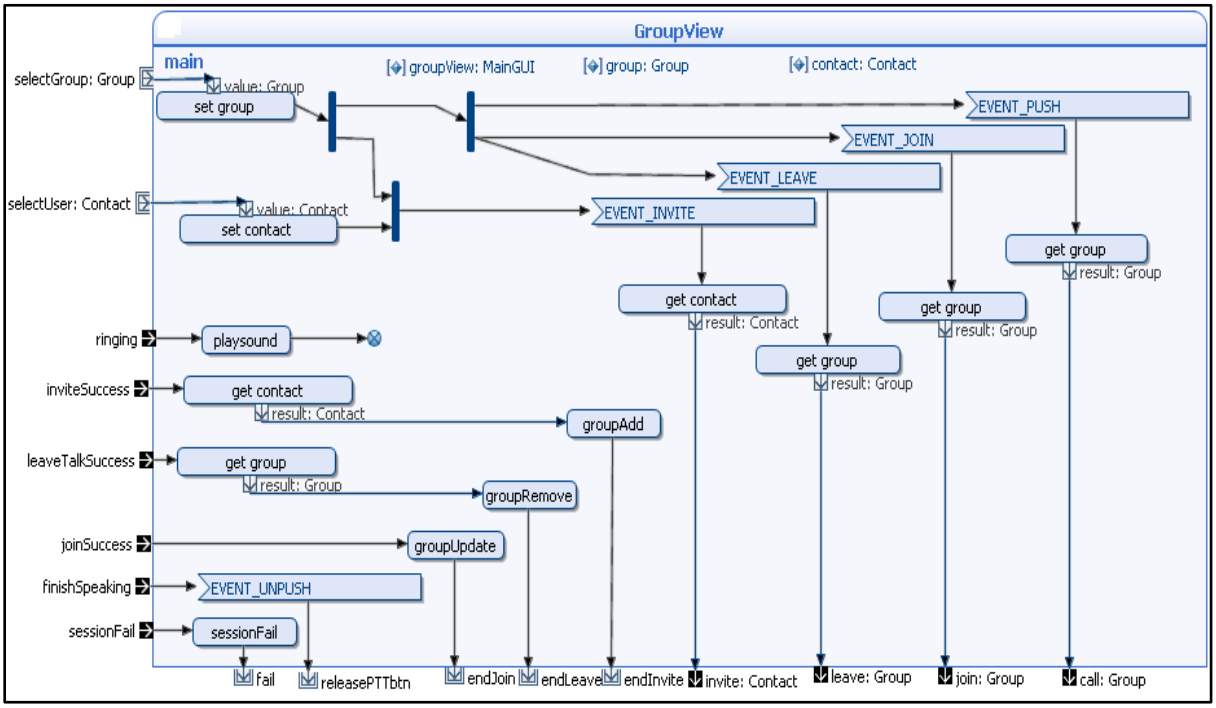


Figure 5-8: Group View building block

Two stating parameter nodes *selectGroup* and *selectUser* are alternative stating parameter nodes of the building block *GroupView*. This building block can start either via the starting parameter node *selectGroup* when the sender selects a group in the sender's group list or via the starting parameter node *selectUser* when the sender selects a contact to invite him or her to join a group, for instance.

When the building block is active via the starting parameter node *selectGroup*, the selected group is packed in a *Group* object defined by a Java class. Then the group information is saved in the variable *group* in the call operation action *set group*. Similarly, when the building block is active via the starting parameter node selectUser, the selected contact is stored in a Contact object defined by a Java class. Then the contact information is saved in the variable *contact*. The using of *set* and *get* variable call operation actions is quite convenient since the objects do not have to follow object flows; the values of the variables can be stored and retrieved by using of these two call operations.

When the token arrives to the second fork node in the building block, it has three edges to go out depending on the signal the building block receives. One of these signal is triggered if the user presses the PTT button to make a call in a group, or he/she presses the Join button to join a group, or he/she presses the Leave button to leave a group, or he/she releases the PTT button to finish a talk session. The last event that the building may receive happens when the user presses the Invite button to invite a contact to join a group. This event is controlled by both starting parameter nodes of the building block.

- The signal EVENT_PUSH occurs when the talker wants to call the selected group. After the signal EVENT_PUSH is received, the selected group information is read and sent to the streaming output node *call.*. When the sender finishes speaking, he or she releases the Call button. Then a signal EVENT_UNPUSH is received. After that, the building block terminates via the terminating parameter node *releasePttbtn*.
- The signal EVENT_JOIN occurs when the sender joins the selected group. After the signal EVENT_JOIN is received, the group information is read and sent to the streaming output node *join*.
- The signal EVENT_LEAVE occurs when the sender wants to leave the selected group. After the signal EVENT_LEAVE is received, the group information is read and sent to the streaming output node *leave*.
- The signal EVENT_INVITE occurs when the sender invites the selected user to join selected group. After the signal EVENT_INVITE is received, the group information is read and sent to the streaming output node *invite*.

If any of these services fails, the building block starts at the streaming input node *sessionFail*. Then a *sessionFail* operation is executed to inform the sender that the service fails.

The building block has five alternative terminating parameter nodes: *fail, releasePTTbtn, endJoin, endLeave* and *endInvite*.

- The building block is terminated via node *fail* if any above service fails.

- The building block is terminated via node *releasePTTbtn* when the user releases the PTT button to finish a talk session.
- The building block is terminated via node *endJoin* when the user successfully joins a group. Before terminating, the building block updates the group list on the group list page of the user by executing the call operation action *groupUpdate*.
- The building block is terminated via node *endLeave* when the user successfully leaves a group. Before terminating, the building block removed the group information from the group list page of the user by executing the call operation action *groupRemove*.
- The building block is terminated via node *endInvite* when the user successfully invite a new user to a group. Before terminating, the building block adds a new user in the group member list of the group by executing the call operation action *groupAdd*.

5.3.2 *Ad hoc* instant group talk initiation collaboration

The activity of the *ad hoc* instant group talk initiation building block is shown in Figure 5-9. The building block is active via the starting parameter node *call*. The building block has four alternative terminating parameter nodes: *sessionFail, initiationFail, readyToSendMedia,* and *readytoReceiveMedia.*

- The activity of the building block is terminated via the terminating parameter node *sessionFail* when the PTT server fails to check either the user's credentials or the user's capabilities to create an *ad hoc* instant group talk.
- The activity of the building block is terminated via the terminating parameter node *initiationFail* when the PTT server checks that the user can not have the right to speak since currently another user has a floor in the group talk session. The checking process is done via the call operation *checkTbStatus*. In this operation, the PTT server checks that it already sent a TB_IDLE message to each member of the group talk session, and then the PTT server forwards the SIP INVITE request from the sender to the receivers. Later on, when the exchanging of SIP messages is finished to create the session, the PTT server sends a TB_GRANT message to the sender to give him or her the right to speak. Also the PTT server sends TB_TAKEN messages to the other members of the group talk session.
- The activity of the building block is terminated via the node readyToSendMedia/readytoReceiveMedia in sender/receivers when the ad hoc instant group talk intiation is successfully created.

The building block produces two output events *ringing* and *viewIncomingCall*. These events act the same as these events in the instant personal talk session initiation.

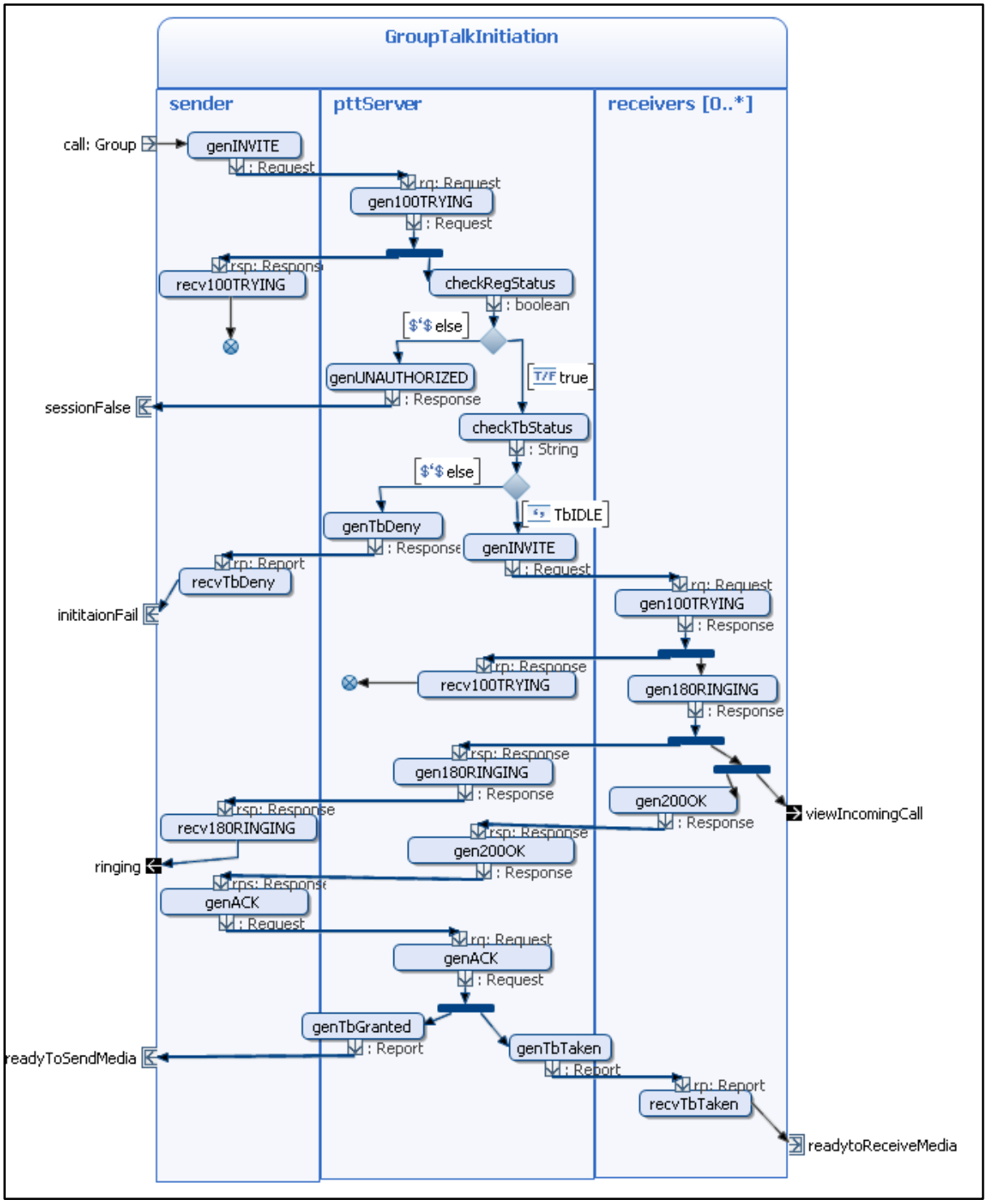


Figure 5-9: *Ad hoc* instant group talk initiation session initiation

5.3.3 Group joining collaboration

The activity of the group joining collaboration is shown in Figure 5-10. In this collaboration, if nobody is talking in the group at the moment that the sender joins the group, the sender receives a TB_IDLE message from the PTT server. Then the building block of the collaboration is terminated at the terminating parameter node *joinSuccess* in the sender. In

other cases, the sender receives a TB_TAKEN message from the PTT server. Then the building block of the collaboration is terminated at the terminating parameter node *readyToReceiveMedia*.

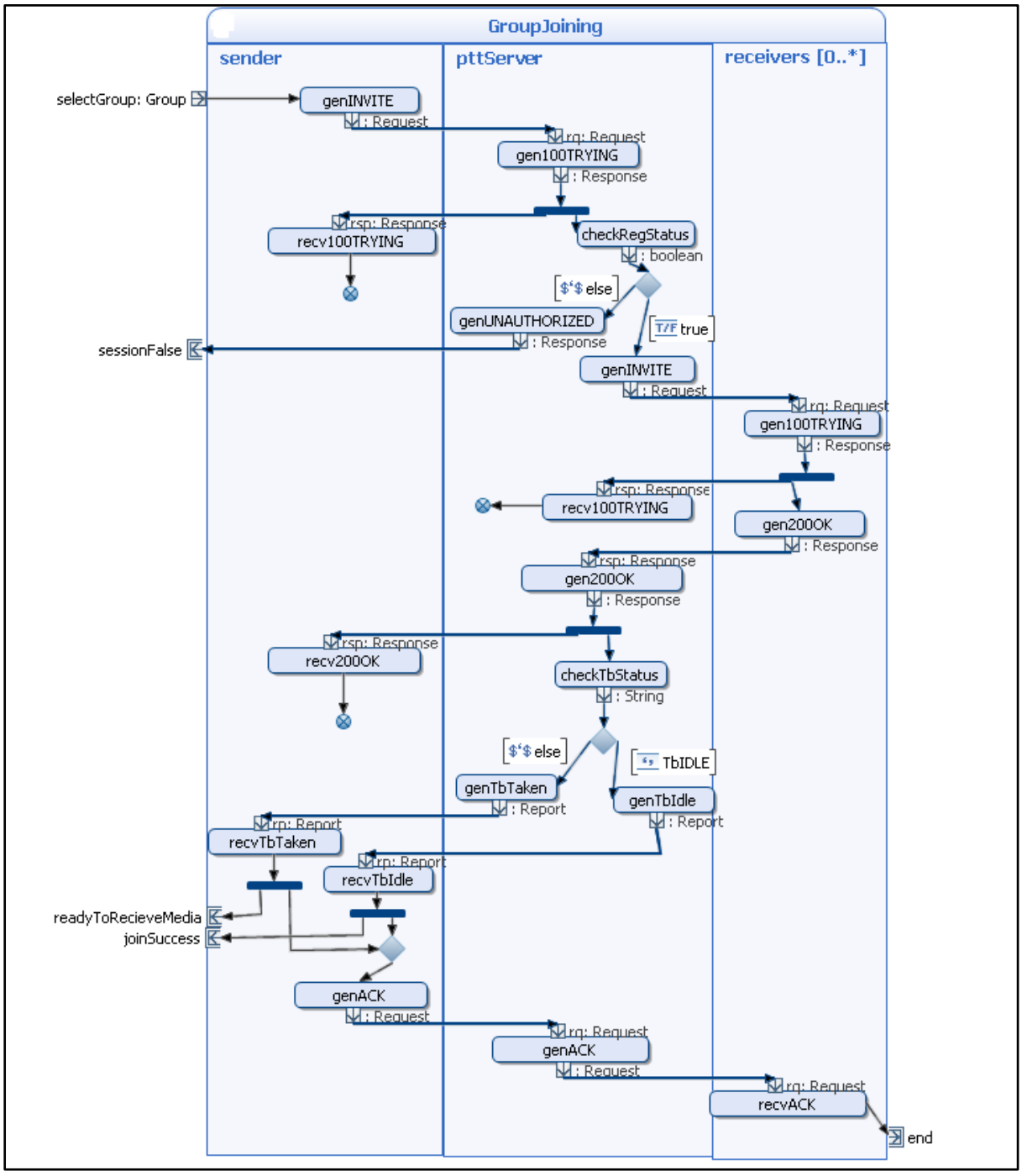


Figure 5-10: Group joining

5.3.4 Talk leaving

Figure 5-11 shows the internal activity of the talk leaving collaboration. The activity starts when there is a data available in the statring parameter node *selectGroup*. The input data

contains a group object specifying the group which the sender is about to leave. From the starting node, the sender executes the call operation action *genBYE* to generate a SIP BYE request which is carried in a object flow to the receivers. After the receiver get the SIP BYE request, it generates a SIP 200OK response by running the call operation action *gen200OK*. The SIP 200OK response is then carried in an object flow to the sender. The collaboration terminates via the output pin *leaveTalk* in the sender.

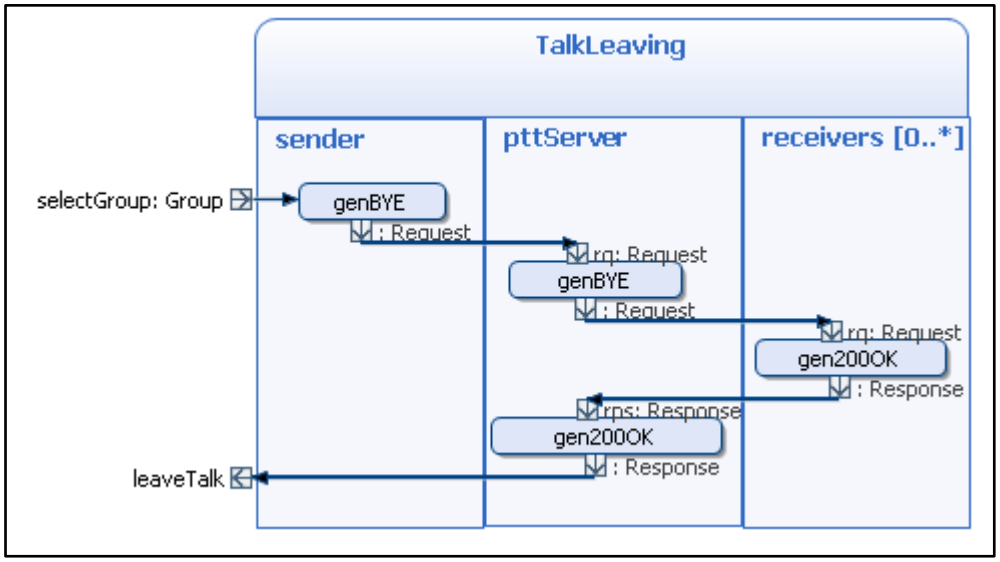


Figure 5-11: Talk leaving service

5.3.5 Add user to a group

Figure 5-12 shows the internal activity of the collaboration *userAdd*. This collaboration allows a PTT user add another PTT user to a group. In this case, the receiver has the multiplicity of one.

The building block of the collaboration has two output events *invitationAlert* and *invitationAccepted*. The former event shows a dialog to the receiver that he or she receives an invitation to join a group from the sender. The later event requires the receiver to accept the invitation by pressing a confirmation button on the dialog.

.

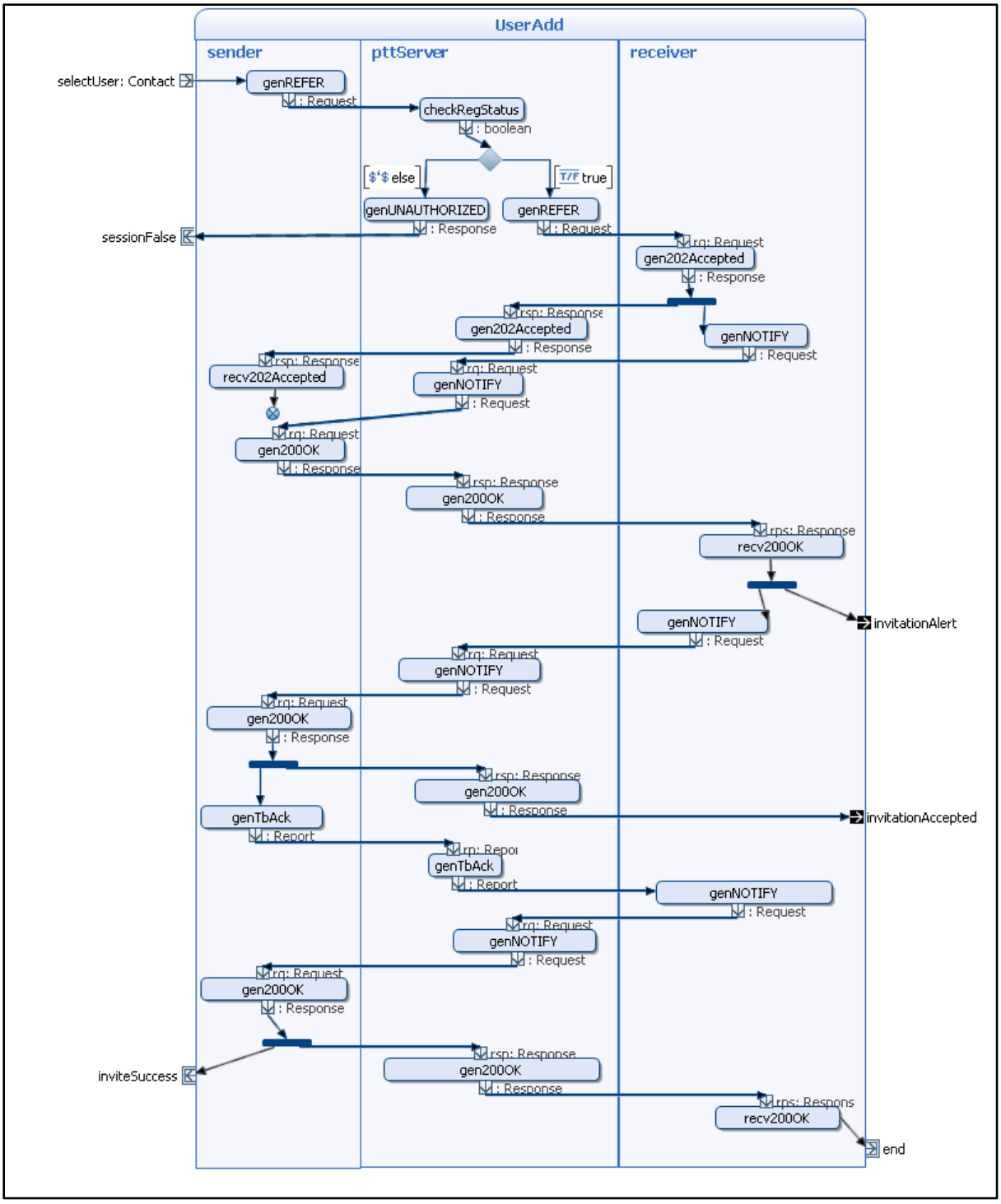


Figure 5-12: Adding user in a group

5.3.6 *Ad hoc* instant group talk collaboration

Figure 5-13 shows the *ad hoc* instant group talk collaboration with several sub-services. The activity has two alternative starting parameter node: *selectGroup* and *selectUser* corresponding to the event that the user selects a group to join/call/leave or the user selects a user to invite him/her to join a group. The detail behaviors of each building block in the diagram are presented in the previous sections.

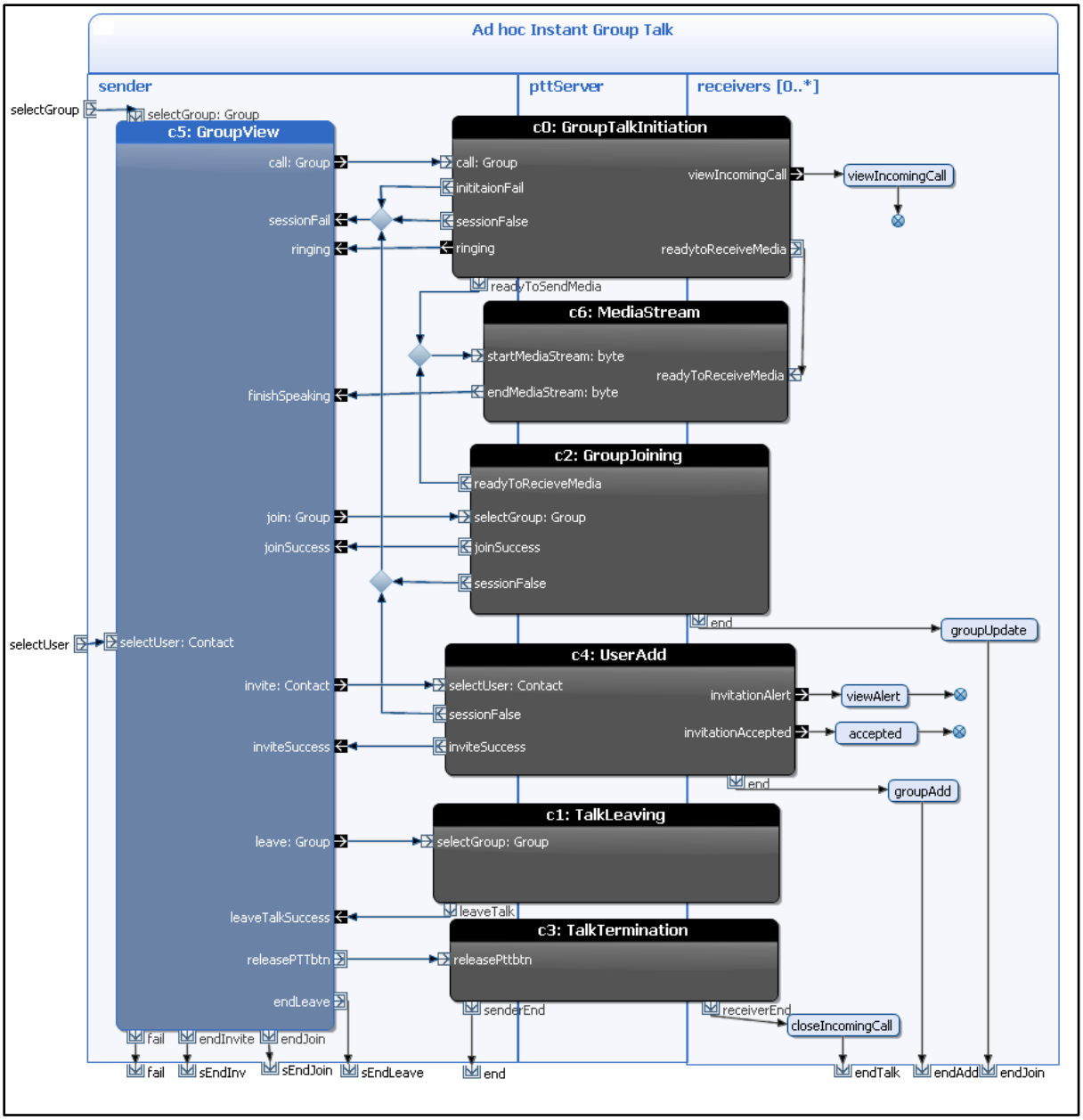


Figure 5-13: Ad-hoc instant group talk activity diagram

From the input pin *selectGroup* of the group view building block, the sender activates the building block group view c5. In this building block, the sender can call a group, join a group, leave a group, or invite another user to a group.

- If the group view building block terminates via the output pin *call*, the group talk initiation collaboration c0 starts among three participants: the sender, the PTT server and the receivers. If the group talk session initiation fails, it terminates at either output pin *initiationFail* or output pin *sessionFalse*, then the group view building block starts from the input pin *sessionFail*.

  If the group talk initiation collaboration c0 terminates at the output pin *ringing*, the group view building block starts from the input pin *ringing*. The group talk initiation

collaboration terminates at the output pin *readyToSendMedia*, the media streaming process gets started.

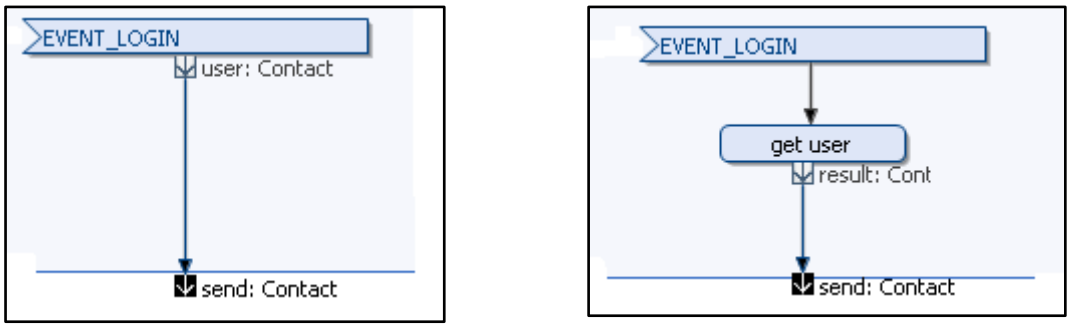
- If the group view building block terminates via the output pin *join*, the group joining collaboration c2 is active. If the group joining c2 fails and terminates at output pin *sessionFalse*, the token goes to the input pin *sessionFalse* of the group view building block. If the group joining successes, the group view building block start from the input pin *joinSuccess*. When the collaboration terminates at output pin *end* in the receiver, the token goes to call operation *groupUpdate* to update group information. Then the token arrives at the terminating parameter node *endJoin* in the sender. The whole activity is terminated then.
- If the group view building block terminates via the output pin *invite*, the collaboration c4 starts. The collaboration c4 terminates at the output pin *invitationAlert*,, the call operation action *viewAlert* in the receiver is executed to show a dialog of incoming invitation to the receiver. In case the collaboration c4 terminates at the output pin *invitationAccepted* the operation *accepted* in the receiver processes the event that user accept the invitation of the sender. After arriving to one of these call operations invitationAlert and invitationAccepted, the token goes to a flow final node and to be removed. The activity of the invitation continues.
- If the group view building block terminates via output pin *leave*, the talk leaving collaboration starts.
- If the group view building block terminates via output pin *releasePttbtn*, the talk termination collaboration starts. The collaboration terminates in both the sender and the receiver. After going out the output pin *receiveEnd* in the receiver, the token enters the call operation action *closeIncomingCall* to close the incoming call dialog in the receiver.

6 Discussion

PTT can be implemented in a semi-distributed manner, thus part of the PTT server runs at each access point (i.e., there is a local PTT server associated with each WLAN access point). Hence, every PTT client can register with the closet access point to get PTT service and each access point can know about the locally registered clients even when they move around. In a PTT communication session, the sender just send the media only once time to the closet access point, then the access point distributes the media to the other clients. In addition, PTT clients registered with the same access point can communicate directly to each other through that access point. Talk burst requests for a group communication can be queued in First In First Out (FIFO) order at the registered access point. These features reduce the latency of the PTT service and reduce overhead of routing packets in the network.

The SPACE method is well suited for creating the specification of a PTT service since the service can be modeled as building blocks of UML collaborations and activities. Furthermore, in a PTT session instantiation and termination, SIP message flow and RTCP message flow are exchanged between a sender and receivers through a PTT server. Each message flow can be modeled by an object flow between these participants in Arctis editor. The participant just has to generate the messages. Arctis will be responsible for transporting these messages to the destination. However, this transportation is at a lower layer between devices, not at the network layer.

The elements in a building block are not fixed when we create a building block. We can have multiple ways to design a building block by choosing different activity nodes. For example, Figure 6-1 describes two ways to retrieved data in the Login building block when the user enters the username and password to login to the system. The username and password information is packed in a Contact object. Then it can be transferred to the terminating parameter from an output pin of the accept action EVENT_LOGIN through an object flow to the terminating node. Another way to transfer the login information to the terminating node by executing a call operation action *get user* to retrieve the login information from the variable *user*.



(a) Accept signal with an object flow (b) Accept signal with a control flow

Figure 6-1: Data with accept signal action

Arctis should support the reusability of call operation actions. Experienced from our project, we find that a lot of call operation actions, for instance genINVITE, genBYE, gen100Trying,

etc., are created in both PTT client and PTT server. Hence, if the call operation actions can be reuse within a project, we can use them when needed. A solution for this problem can be figure out by the current implementation of Arctis by creating an activity building block that includes the target call operation action. Then the building block can be instantiate when it is needed. For example, we can create a building block that generate the a SIP INVITE message, the output pin of the building block contains the generated SIP INIVTE message.

Because a building block is active via an input pin, in other word the activity of the building block is activated via a starting parameter, this is a limitation of the building block. When a building block has several collaboration roles and the activity of the whole building block can get start only when, for example, all of the collaboration roles get started. However, this limitation can be improved when using a join node to join all the starting flows to a single starting flow of the whole building block. Furthermore, an input pin of the building block can contain a data object. Hence this is considered as another drawback of the building block when it needs a number of data input. There is a solution for this drawback that we can defined an object type of the input pin that includes all the data input of the building block. However, it is still a challenge when the input data comes from different input pin of multiple collaboration roles and they need to be asynchronous.

In our work, we choose several simple cases of PTT service to model. However, the model is quite large since there are a lot of collaborations in a service and they related to others. That means that the specification is not flat. The software developer needs to read them in parallel to track the system. But, a simulation of a token flow among these collaborations might reduce the burden of understanding the PTT specification.

7 Conclusions and Future Work

7.1 Conclusions

In this project, we gave an overview of the SPACE method and a PTT service. Then we design how to create building blocks for a PTT service by using the SPACE method. As a result, several building blocks of a PTT service have been proposed.

We consider that the SPACE method well suited for designing a PTT service for various reasons. Firstly, a PTT service always includes at least a sender and a receiver participant; hence the collaboration-oriented specification approach is really helpful to develop a system. Secondly, the separation of the system into building blocks supports the understanding of the system. Thirdly, when the structure and the behavior of the building blocks are specified, the state machines can be automatically transformed. Then from state machines, executable code is generated. Lastly, the building blocks can be stored in a library for reuse.

Currently, the COOS platform does not have any APIs or framework that support SIP protocol and media stream. In order to generate SIP messages and RTCP messages in our project, we use JAIN SIP API and Java Multimedia Framework. It is a good solution for a PTT service over a SIP protocol in the future. JAIN SIP APIs and JMF can be introduced in to the COOS platform afterward.

7.2 Future work

By utilizing the SPACE method, we presented a novel approach to build a PTT service over a SIP protocol in WLAN environment. In this project, we have been initialized the work for developing this PTT service. The study of our research will continue this approach to build a PTT service having the capabilities of presence, group list management, real-time media streaming. The success of the PTT service will lead us to develop other applications in telecommunication by using the SPACE method.

Supporting the reusability of call operation action is probably a good feature of Arctis in the future. Also other limitations of Arctis as discussed in chapter 6 including should be improved.

References

[1] **WinterGreen Research, Inc.** Push to Talk Over Cellular Market Assessment. [Online] http://www.wintergreenresearch.com/reports/push%20to%20talk%202007.html.

[2] **Frank Alexander Kraemer.** *Arctis and Ramses: Tool Suites for Rapid service Enigneering.* NIK 2007 (Norsk informatikkonferanse), Oslo, Norway. November 2007. Tapir Akademisk Forlag.

[3] **Frode Flægstad.** *Sea Cage Gateway.* EIT Tyholt 20090119 : TelCage / Telenor R&I, 2009.

[4] **Open Mobile Alliance.** *PoC User Plane, Version 1.0.* 09 June 2006.

[5] *The Swing Tutorial.* [Online] http://java.sun.com/docs/books/tutorial/uiswing/.

[6] Eclipse. [Online] http://www.eclipse.org/.

[7] Minisip. [Online] http://www.minisip.org/.

[8] Counterpath. [Online] http://www.counterpath.com/.

[9] Ekiga. [Online] http://www.gnomemeeting.org/.

[10] **Frank Alexander Kraemer, Rolv Bræk, Peter Herrmann.** *Compositional Service Engineering with Arctis.* Telektronikk x/2009, 2009 (to appear).

[11] **Frank Alexander Kraemer, Peter Herrmann.** *Formalizing Collaboration Oriented Service Specifications using Temporal Logic.* Networking and Electronic Commerce Research Conference 2007 (NAEC 2007). October 2007.

[12] **Nina Heitmann.** *Towards Modeling of Data in UML Activities with the SPACE Method.* Norwegian University of Science and Tchnology (NTNU). Department of Telematics, June 2008. Master thesis. http://daim.idi.ntnu.no/masteroppgaver/IME/ITEM/2008/4116/masteroppgave.pdf .

[13] **Frank Alexander Kraemer.** *UML Profiles and Semantics for Service Specifications.* Norwegian University of Science and Tchnology (NTNU). Department of Telematics. June 2008. Avantel Technical Report 1/2007 ISSN 1503-4097.

[14] **Frank Alexander Kraemer and Peter Herrmann.** *Transforming Collaborative Service Specifications into Efficiently Executable State Machines.* Proceedings of the 6th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007). Vol.7. 2007.

[15] Motorola. *Push-to-Talk over Cellular Consortium Phase 2 Specifications and Documentation.* [Online] http://www.motorola.com/content.jsp?globalObjectId=2647-4398.

[16] **Northstream study white paper.** *Push-to-talk Over Wireless Is the time right for Push-to-talk? Does it work over GPRS?* http://www.incodewireless.com/media/whitepapers/2004/PoC_White_Paper-Feb_2004.pdf. 2004.

[17] **Open Mobile Alliance.** *Push to talk over Cellular (PoC) - Architecture, Version 2.0.* 7 May 2008.

[18] **J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler.** *SIP: Session Initiation Protocol.* IETF RFC 3261. June 2002.

[19] **M. Handley, V. Jacobson.** *SDP: Session Description Protocol.* April 1998. IETF RFC 2327.

[20] **H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson.** RTP: A Transport Protocol for Real-Time Applications. [Online] July 2003. http://www.ietf.org/rfc/rfc3550.txt.

[21] **Open Mobile Alliance.** *OMA Control Plane, Version 1.0.* 09 June 2006.

[22] **Comneon, Ericsson, Motorola, Nokia and Siemens.** *Signaling Flows - Network to Network interface (NNI); PoC Release 2.0;*. June 2004.

[23] **Florian Maurer.** *Push-2-Talk in VoIP: Decentralized.* Royal Institue of Techonology (KTH). August 30, 2004. Master thesis.

[24] **Comneon, Ericsson, Motorola, Nokia and Siemens.** Push to Talk over Cellular (PoC); List Management and Do-not-Disturb V2.0.6, PoC Release 2.0. 2006.

[25] **Bi-Feng Yu, Jae-Oh Lee.** *A PoC Deployment in the IMS and its Simulation.* Dept. of Electrical and Electronics Engineering,Korea University of Technology and Education, Korea. 2008.

[26] **Meng-Hsun Tsai, Yi-Bing Lin.** *Talk Burst Control for Push-to-Talk over Cellular.* IEEE transactions on wireless communications, vol.7, No.7, July 2008.

[27] Telcage - Enabling Integrated Operations in Aquaculture. [Online] http://www.telcage.com/node/13.

[28] Connected Objects wiki. [Online] http://co-wiki.pats.no/.

[29] National Institute of Standards and Technology. *The IP telephony project.*

[Online] http://snad.ncsl.nist.gov/proj/iptel/.

[30] Sun Developer Network. *Java Media Framework API.* [Online] http://java.sun.com/javase/technologies/desktop/media/jmf/.

[31] **Peter Herrmann, Frank Alexander Kraemer.** *Design of Trusted Systems with Reusable collaboration Models.* Trust Management. Proceedings of the Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM 2007), IFIP International Federation for Information Processing, 2007, Moncton, Canada. Springer, 2007.

TRITA-ICT-EX-2009:59

www.kth.se