Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Diplomarbeit Nr. 3275

Web Services for Human Interaction

Lina Sun

Course of Study:	Informatik
Examiner:	Jun.-Prof. Dimka Karastoyanova
Supervisor:	Dipl.-Inf. David Schumm
Commenced:	December 15, 2011
Completed:	June 15, 2012
CR-Classification:	C.2.4, H.4.1, H.5.2

Abstract

In recent years, with the continuous development of the network, the network society has been built up. To develop a more perfect and more powerful network society, a communication between members of this society is indispensable. The communication between human users is very common. There are already many means of communication between human users, for example, MSN, Mail, Skype, Twitter, etc. Business processes can also communicate with external services. Since the advent of Web Services, this kind of communication becomes platform-independent and flexible. However, how to integrate an automating business process with human users remains to be studied.

There are multiple ways how human users communicate electronically, for example via Mail, Skype, ICQ message, and SMS. The main task of this work is to implement such communication channels. To provide a flexible integration between human users and business processes these communication channels will be implemented via Web Services.

This work proposes concepts of two communication channels: Mail and Skype. Getting a request, processing a request, sending a message, and receiving a message are basic functions. As a very important component a database has been added to store all information.

To prove these concepts, they have been put in practice as part of this work. Two Web Services have been developed in this work: Mail Service and Skype Service.

In the very end, a set of test cases illustrates the capabilities of Mail Service and Skype Service.

Table of Contents

1 Introduction

This chapter introduces a model of communication system in human world. Compared with it in the virtual world such a communication system should be also existed.

1.1 Communication System

In society, people all interact with messages. Without interactions, a society cannot survive. Social interaction is always through messages. So communication can be defined as:

Communication is social interaction process by which a message or information is exchanged from a sender to a receiver [1].

Figure 1 is a model of communication system [3]. In this communication system there are three important components, channel, sender and receiver. A sender is a message source and sends out messages. Channel is a mean of exchanging a message. A receiver gets a message and can send feedback to the sender. The main function of a communication system is to share and process information between the members of a society.

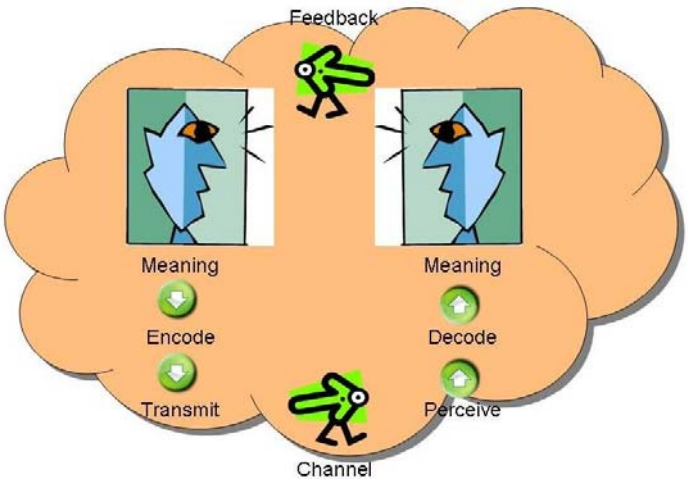

Figure 1. A model of communication system [3]

In recent years, with the continuous development of the network, the network society has been built up. To develop a more perfect and more powerful network society, a communication between members of this society is indispensable. The communication

between human users is very common. There are already many means of communication between human users, for example, MSN, Mail, Skype, Twitter, etc. In addition to human users, business processes are also considered as an important member of this society. Once business process and human users can exchange messages, the entire network society will have a higher degree of automation and efficiency. Today many systems have been built the communication up. But the types of communication channels are not rich. There are many channels that can be developed to support this communication between a business process and a human user.

1.2 Motivation

All over the world, thousands of business processes are automatically running in many companies every day. Efficiency and convenience are the great advantages of them. Most business processes are frequently used, and they can complete tasks independently. In many cases for users it is not the process but the result of it that is really important. Figure 2 shows a simple business process that calls an external Web Service to complete the intended goal.

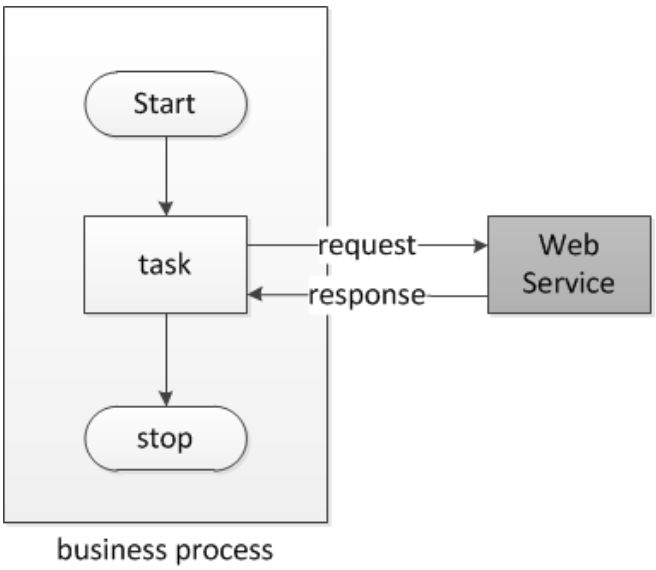

Figure 2. A business process calls a Web Service

In recent years a concept “Interaction” is proposed [2], and human interaction becomes increasingly important in the development of Web Services. Figure 2 shows the interaction between a business process and a Web Service. If in this example the Web Service is replaced by a human user, the business process will have more extensive range in usage. The business process is no longer closed to run, but can run in the interaction with human users. This kind of Interaction is considered as a kind of communication between a

process and a human user. Figure 3 shows a business process that interacts with human user.

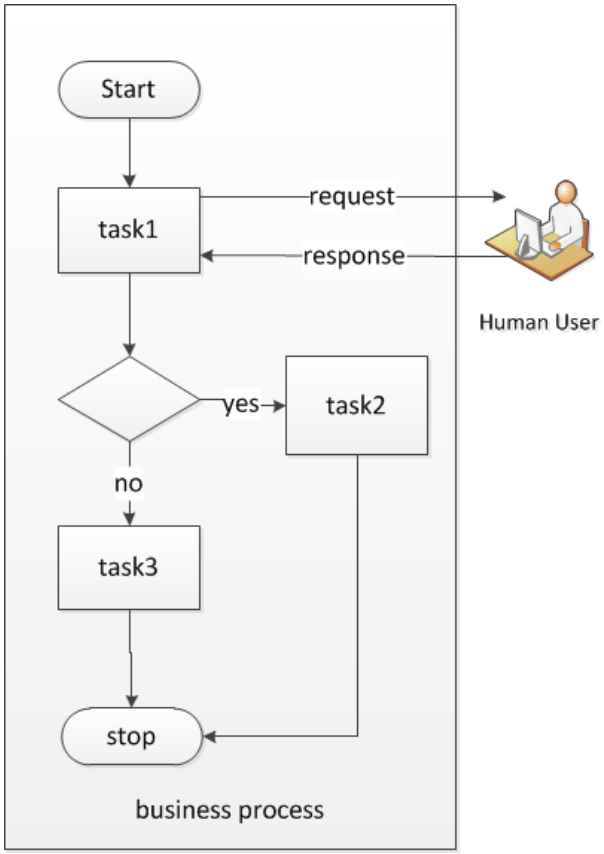

Figure 3. Business process interacts with human user

The importance of the interaction is the human factor. Similar to calling an external service, this time the business process just calls a human user. By issuing a request to a human user and receiving a reply from him, the business process can set some variables and make a decision based on the values of these variables.

The use of conventional workflow environments and service compositions is suitable for automating business processes. However, it is not yet completely applicable for interaction with human users. In particular, different ways of communication for involvement of human users are required.

Following is a use case: A business process is responsible for a meeting to determine the attendance list and send a card to everyone. First this process gets a predetermined contact list. According to this list the process contacts everyone and asks him, whether he will accept the invitation. After that, the process will receive each reply, check validation, and count the replies with “yes” from all valid replies to generate an attendance list. Finally

it will send a confirmation to everyone in the attendance list. This use case is shown in Figure 4.

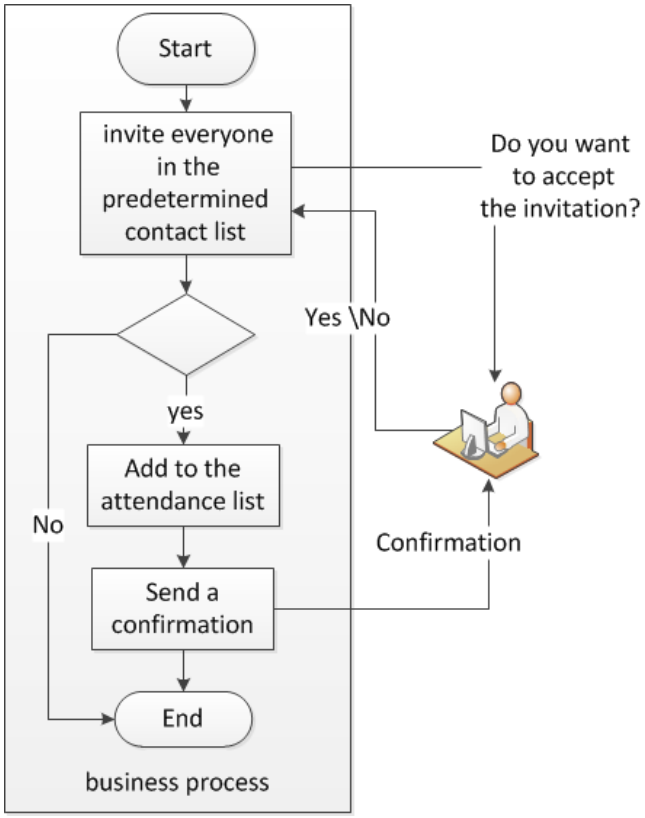

Figure 4. An instance of business process with human interaction

1.3 Objectives

Based on the motivation following objectives were defined for this work:

O1: Create a concept for communication services for human interaction. It should be able to deal with some special network communications (for example: Mail, Skype, and File Transfer Protocol) and enable interaction between a process and human users.

O2: According to the concept implement communication services. Communication services are responsible for getting a request from a business process, sending a message to a specified user and processing a response from a responding user.

O3: Make some instances to test the implemented services.

1.4 Document Structure

The document is divided into the following seven chapters:

Chapter 1 “Introduction” contains introductory information, introduces a communication system, explains the motivation, defines objectives and guides the reader through the rest of the document.

Chapter 2 “Fundamentals and Technologies” introduces the definition of business processes and Web Services as the fundamentals for this work. Nowadays many new technologies are used for Web Services. These technologies include: XML, XML Schema, WSDL and SOAP.

Chapter 3 “Concept” introduces the concept for communication services. Two channels are provided: Mail and Skype.

Chapter 4 “Specifications” defines the detailed information, for example, interface arguments and configuration data.

Chapter 5 “Implementation” introduces the whole process of implementation according to the concepts.

Chapter 6 “Test Cases” shows different scenarios to test the functionalities of Mail Service and Skype Service.

Chapter 7 “Summary and Outlook” summarizes the document and gives an outlook on the problems that were identified by this work.

Bibliography can be found in the end.

2 Fundamentals and Technologies

This chapter introduces some theoretical concepts in the background and the necessary technologies are involved in them.

2.1 Business Process

As fundamental asset of companies, enterprise applications and information systems have become very important. Companies rely on them to be able to perform business operations. Based on them business processes are modelled to realize a business goal.

A business process consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each process is enacted by a single organization, but it may interact with business process performed by other organization [4]. Following the same well-defined process model, these activities are repeated over and over. There are three levels of interaction between the different business processes [5]:

- An activity in a business process invokes another business process. That new business process executes totally independently of the original business process.
- An activity in a business process invokes another business process and waits until that new business process has completed.
- An activity in a business process invokes another business process, and an activity later in the business process waits until that new business process has completed.

An important nature of business processes is dynamic. Business processes should be optimized and adapted in an agile manner to the customers, thus the responsiveness of the whole company can be improved.

Some of business process activities can be performed by the company's employees manually or by the help of information systems. The others can be enacted automatically by information systems and Web Services, without any human involvement. In this thesis the latter business process is as a research object. The goal is to extend a kind of interaction between business process and human.

2.2 Web Services

The term Web Service is used very often nowadays. The Web Service Architecture working group of the World Wide Web Consortium (W3C) developed the following definition for a Web Service [7]:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Web Services provide a standard means of interoperation between software applications, running on a variety of platforms and/or frameworks. For exchanging information they are represented by the request/response pattern. They receive a Simple Object Access Protocol (SOAP) message as request. After processing this message, they send another SOAP message as response. Web Services have the following characteristics [8]:

- Loose coupling

Web Services enable applications to work cooperatively together using the principle of loose coupling. In a loosely coupled exchange, applications need not know how their partner applications behave or are implemented. This greatly reduces the difficulty of application integration.

- Service granularity

Web Services may vary in function from simple requests to complex systems that access and combine information from multiple sources. Simple services are atomic.

- Synchronicity

There are two types of services: synchronous and asynchronous. A synchronous service functions as a method call with a set of arguments, which returns a response. While a client invokes an asynchronous service and it does not need to wait for a response before it continues with the remainder of its application.

- Well-definedness

The service interaction must be well defined in a Web Service Definition Language (WSDL) document.

- Flexibility

Web Services can be dynamically found and can be embedded into remotely located applications.

- Dependence

Web Services are platform-independent. A Web Service is a self-contained software module that performs a single task. The module describes its own interface, i.e. the operations available, data type and access protocols. Through this description others can know what it does, how to invoke its functionality and what result will be returned.

Web Services are the most suitable technology for realization of Service Oriented Architecture (SOA). Figure 5 illustrates bind/publish/find approach in SOA as described in [11]. First, the provider must provide an abstract definition of the service, including the appropriate way to bind this service. Second, the provider needs to publish this service, including the precise information what it does and how to connect to use it. Third, the requestors who require services need to find what services are available that meet their needs.

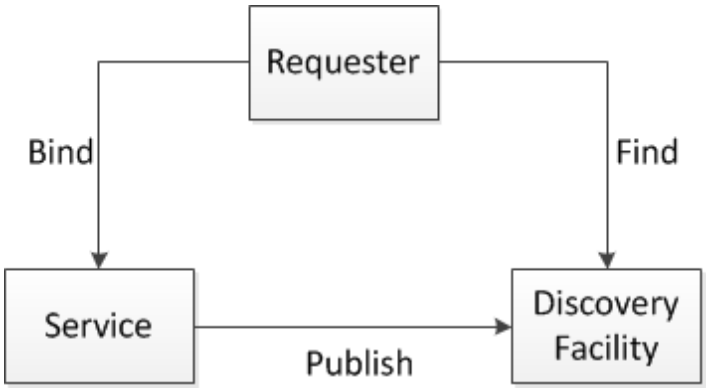

Figure 5. The SOA Triangle [11]

To implement Web Services several specifications are used in Figure 6. Universal Description Discovery and Integration (UDDI) is a standard and a mechanism to register and locate Web Service applications. WSDL describes the interface for a Web Service, i.e. what methods are present in a Web Service, what parameters and return values each method uses, and how to communicate with them. SOAP is a message format for a Web Service. UDDI, WSDL and SOAP are the platform-independent and XML-based format.

Figure 6. The web service technology stack [12]

2.3 XML, XML Schema

XML stands for Extensible Markup Language. XML is a set of rules for defining semantic tags that break a document into parts and identify the different parts of the document. It is a meta-markup language that defines a syntax used to define other domain-specific, semantic, structured markup languages [13]. The definition of XML is independent of the platform and defined using Unicode. Therefore XML has rapidly become the official format for data interchange between disparate entities.

- XML Basics

XML document consists of elements, which are defined through its name, a set of attributes and some children. Listing 1 is a simple XML document called "`customer.xml`". `Customer` element has two children, `Name` element and `Address` element. The definition of an element starts with a start tag `<Customer>` and ends with an end tag `</Customer>`. Attributes are name-value pairs that associated with an element. An element can have any number of attributes and is written as `<Customer name1=″value1″ name2=″value2″…>`.

```
1.  <?xml version=”1.0”?>
2.  <Customer>
3.    <Name>AAA Company</Name>
4.    <Address>
5.      <Line1>219-241 Cleveland St</Line1>
6.      <Line2>0711-3456-234</Line2>
7.    </Address>
8.  </Customer>
```

Listing 1. XML example “customer.xml”

- XML Schema

XML Schema is a document structuring and type definition specification that the World Wide Web Consortium (W3C) developed. It uses XML syntax. But it is powerful to define the structure of a XML document. It defines a set of primitive data types to define attribute and element values. Listing 2 is an XML Schema file called “`customer.xsd`” that defines the elements of the XML document above (“`customer.xml`”). `Customer` element is a complex type because it contains other elements.

```
1. <?xml version=”1.0”?>
2.   <xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema>
3.   <xs:element name=”Customer”>
4.   <xs:complexType>
5.     <xs:sequence>
6.       <xs:element name=”Name” type=”xs:string”>
7.       <xs:element name=”Address” type=”AddressType”>
8.     </xs:sequence>
9.   </xs:complexType>
10.  </xs:element>
11.  <xs:complexType name=”AddressType”>
12.    <xs:sequence>
13.      <xs:element name=”Line1” type=”xs:string”/>
14.      <xs:element name=”Line2” type=”xs:string”/>
15.    </xs:sequence>
16.  </xs:complexType>
17.  </xs:schema>
```

Listing 2. An example of XML Schema "customer.xsd"

- XML Namespace

To guarantee uniqueness named elements and attributes in an XML document XML Namespaces is required. It is a way to scope element and attribute names to a namespace so that their usage is always unique, regardless of their context. In Listing 2 line 2 indicates that the elements and data types used in the schema come from the “http://www.w3.org/2001/XMLSchema” namespace and should be prefixed with xs:.

XML document and XML Schema are widely used in the network field. All of the Web Services specifications have associated schemas written using XML Schema. SOAP

message is an example of an XML vocabulary. WSDL is another example. Most of the Web Service specification format is defined by XML vocabulary. A XML document is well-formed if it adheres to all the XML syntax rules. The validity of a XML document can be detected by its XML Schema. A well-formed document is also valid if it conforms to some XML Schema.

2.4 WSDL

Web Services Description Language (WSDL) plays an important role in the overall Web Services Architecture. WSDL is a specification defining how to describe Web Services in a common XML grammar and describes four critical pieces of services [9]:

1. Interface information describing all publicly available functions
2. Data type information for all message requests and message responses
3. Binding information about the transport protocol to be used
4. Address information for locating the specified service

WSDL uses eight major elements [9]:

definitions

The *definitions* element is the root element of all WSDL documents. It defines the name of the Web Service and namespaces.

types

The *types* element is a container for data type. It defines abstract data type using XML Schema.

message

The *Message* element describes a message for input or output. It contains 0 or more *part* elements, which can refer to parameters or return values. Name and type are defined for *part* elements.

operation

The *operation* element describes all functions in this Web Service. They are defined by an exchange of message.

portType

The *portType* element abstractly describes a set of operations.

port

The *port* element defines an individual endpoint by specifying a single address for a binding.

binding

The *binding* element defines the concrete protocol and data format for this service.

service

The *service* element defines the address for invoking a service. It includes a URL for invoking the service.

2.5 SOAP

Simple Object Access Protocol (SOAP) [10] is a specification for exchanging structured information in the implementation of Web Services and its format is based on XML. A valid SOAP message is a well-formed XML document. There are three elements in following Figure 7 SOAP message structure: SOAP-ENV: Envelope, SOAP-ENV: Header, and SOAP-ENV: Body. Envelope defines the start and end of the message. Header contains any optional attributes of the message used in processing the message. Body contains the XML data comprising the message being sent. Among of them Envelope and Body elements are Mandatory, Header element is optional.

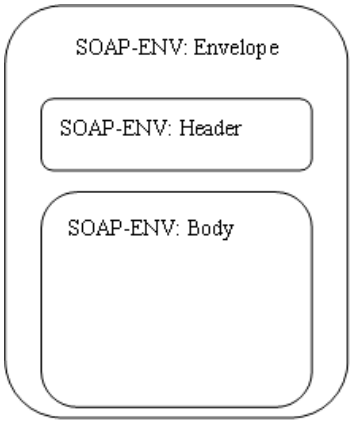

Figure 7. SOAP message structure [11]

The Web Services specification includes: SOAP, UDDI and WSDL. UDDI allows clients to discover Web Services. WSDL describes the functionality and attributes of Web Services. Web Services interoperate through SOAP. Figure 8 gives an architectural representation of the Web Services Model [16]. First, a service provider describes its service using WSDL. This definition is published to a directory of services. The directory use UDDI here. Second, a Web Service User locates a service through the directory and determines how

to communicate with that service. Third, part of the WSDL provided by the service provider is passed to the service user. This tells the service user what the requests and responses are for the service provider. Fourth, the service user uses the WSDL to send a request to the service provider. Finally, the service provider provides the expected response to the service user.

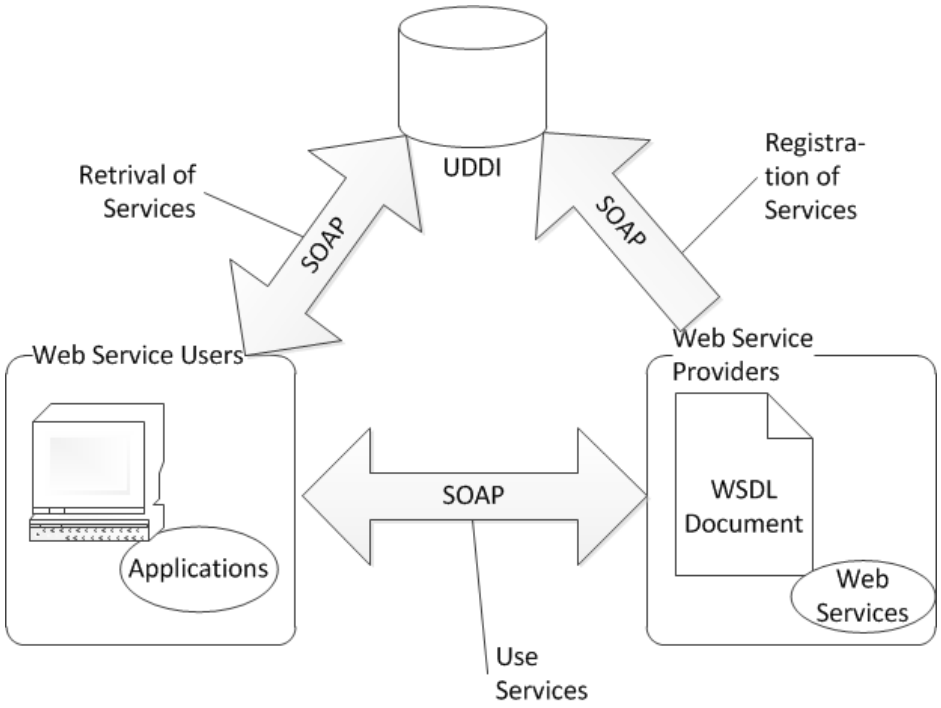

Figure 8. Web Services Model [16]

3 Concepts

This chapter introduce different aspects of a communication system, for example, participants, classes and means of the entire communication system. Afterwards, with the help of these aspects, some concepts are proposed for Mail Service and Skype Service.

3.1 Participants of Communication

In this thesis the mentioned communication is the interaction between a business process and a human user. Such communication is rather special. Today the interaction between two processes has been widely used. Due to Web Services, this interaction uses the principle of loose coupling and it is platform-independent and language-independent. In contrast, the interaction with human users has not been developed so mature and has several following problems. First, a human user can have several different contacts, for example Mail, Skype or MSN. Then which way can a communication system choose to contact him? Second, if the user does not respond promptly, how will the communication system deal with this situation? Third, a response from a human user may be a variety of errors in the content and format. How will the system check it? There are so many similar problems that this interaction with human has the high complexity.

In a communication system there are four components: a communication initiator, the human communication manager, communication services and human user [14].

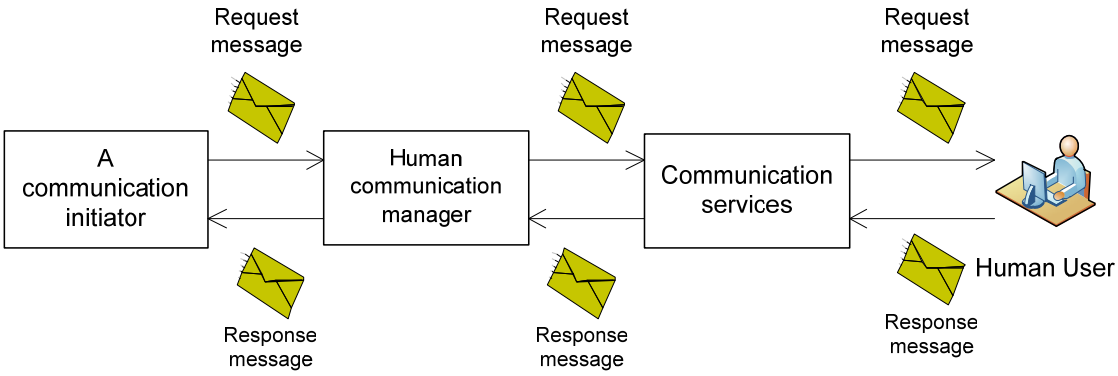

Figure 9. Participants in communication system

1. A communication initiator

A communication initiator can be an application or a service and wants to communicate with a human user. It sends a request message to the human communication manager.

2. The human communication manager

The human communication manager establishes a communication between a communication initiator and a communication service. It is able to do these tasks:

1) receive the request message from initiator
2) check the request message
3) choose the best communication service channel
4) send a special message to this service
5) receive the response message from communication service
6) send a response message to initiator

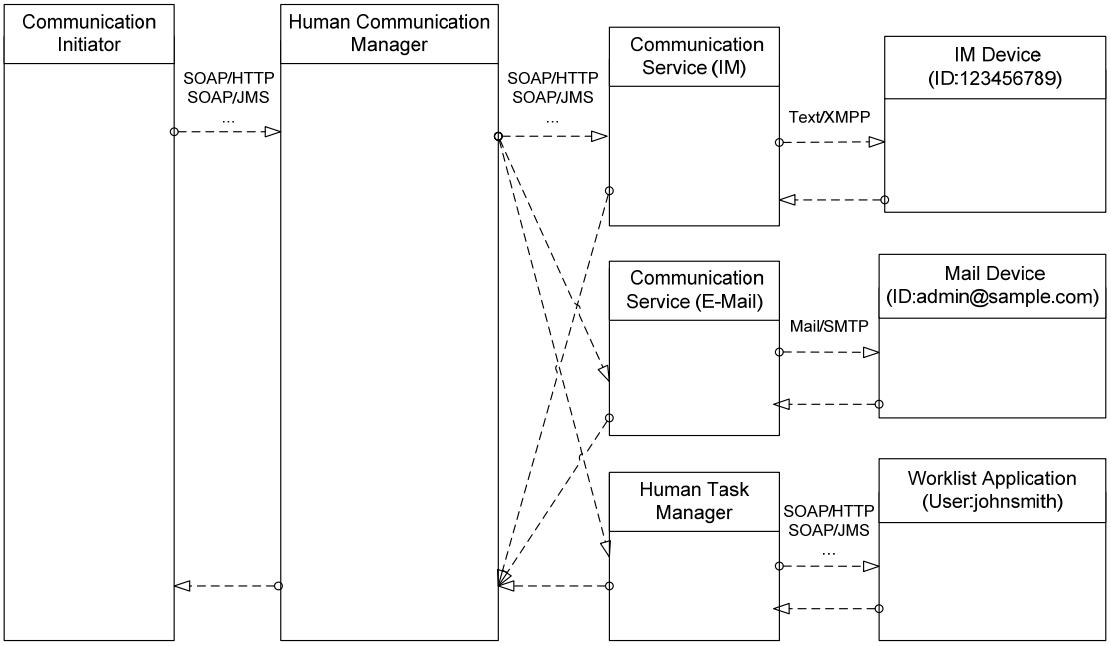

Figure 10. Communication participants [14]

3. Communication services

A communication service is a concrete endpoint for a selected communication channel. For example, Mail communication channel, Skype communication channel, FTP communication channel, Twitter communication channel. This thesis is focused on this component.

4. Human user

Human user receives a message from the communication system and sends a response to the system.

3.2 Classes of Communication

As proposed in [17], there are at least three main classes of communication with humans: notification, decision, and control.

3.2.1 Notification

This class of communication refers to reports sent to a user in order to inform him about a particular status of an application, service, workflow etc. The reports may be regular status reports, fault reports, processing completeness updates etc. A notification does not require any response by the user.

3.2.2 Decision

This class of communication refers to sending information to a user to let him decide how to react to a particular situation. A decision may be a simple approval/rejection, may offer multiple choices to select from, may require specifying particular parameters, and may require a user to check or correct certain data that cannot be checked or corrected automatically.

3.2.3 Control

This class of communication refers to remote control of an application that can be steered by a user. Depending on the capabilities of an application for remote control, processing may be paused, resumed, aborted, retried in case of a fault, iterated, or skipped.

This thesis mainly focuses on two classes of communication, notification and decision. In this work, a request which doesn't need a response is defined as a notification request. A request which needs a response is defined as a decision request.

3.3 Means of Communication

Different means of communication are in the communication system as different channels. According to the different characteristics of these means, a standard to choose a channel can be made for the component human communication manager. This thesis focuses on two means of communication, Mail and Skype.

3.3.1 Mail

Mail mentioned in this thesis refers to e-mail. It is a kind of asynchronous exchange of information by electronic means of communication. It allows users to send or receive text, picture and recorded voice and other forms of information. Mail system is based on client-server model. Every mail involves the sender and receiver. By mail client, the sender sends a message to the Simple Mail Transfer Protocol (SMTP) server. To retrieve a mail from a remote server Post Office Protocol 3 (POP3) is used as an internet standard protocol and makes it possible to download a message sent by any SMTP server. The sent message is stored on the server until it is retrieved. Then the message is removed from the server and stored on the local hard drive of a user.

The advantage of mail is not to require the recipient online, which means that it is an asynchronous way. Communication system needs only an address from the recipient.

His disadvantage is that attachment size is limited. For example, Yahoo! Mail lets user send mails up to 10MB in total size. This size encompasses both the message itself and all its attachments. With Google Mail, user can send and receive messages up to 25MB in size. This limitation can be one of the criteria for choice of the channel.

3.3.2 Skype

Skype [18] is a service for its users to communicate with each around the world over the internet, including video and audio calls, chat (instant messaging), sending file and conference audio calls. In our communication system we use only instant messaging and sending file. Before user call Skype he must register, sign in and add a new contact. Then he can send a message to this user if he is online now. This is both its advantage also its disadvantage. Through Skype the system may receive replies in the shortest time.

3.3.3 FTP

File Transfer Protocol (FTP) is a standard network protocol used to transfer files from one host to another host over a TCP-based network, such as the Internet. FTP is built on a client-server architecture and uses separate control and data connections between the client and the server. FTP users may authenticate themselves using a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it.

3.3.4 Twitter

Twitter [19] is an online social networking service and microblogging service that enables its users to send and read text-based posts of up to 140 characters.

3.4 Mail Service

This thesis is mainly responsible for design and implementation of component communication services. There are two subservices in this component, Mail Service and Skype Service.

3.4.1 First Concept of Mail Service

The main function of Mail Service is to use mail to contact human and return a response to the process. Mail Service should be able to receive a request from a running process. According to the request message it should extract all useful information, for example, the receiver, the subject, the mail text, an attachment and so on, then build a mail message and use a dedicated mail address to send this mail. Following the sending, Mail Service should determine, whether this mail needs a response. If this mail needs a response, the process should wait for a reply until the timeout. Finally, Mail Service will return a message to notify the process, whether the request is accepted or not.

According to the main function of Mail Service, the architecture for Mail Service can be shown in Figure 11.

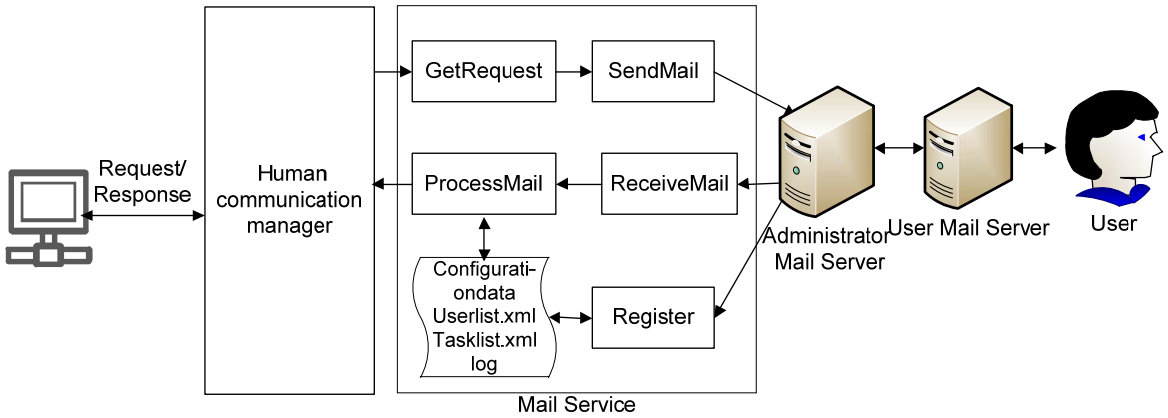

Figure 11. Architecture of first concept

In the architecture there are six components: 1) a requestor, 2) human communication manager, 3) Mail Service, 4) administrator mail server, 5) user mail server, 6) human user. The requestor sends a request to human communication manager, after some processing the human communication manager calls Mail Service. Mail Service has an administrator's mailbox. All the messages sent by this service are sent through this administrator's mailbox.

There are four data files in Mail Service: configurationdata, Userlist.xml, Tasklist.xml, and a log file.

- Configurationdata: all information for administrator account, log information. For example, administrator mail address, receive/send host, receive/send port, username and password.
- Userlist.xml: this file stores all registered mail address.
- Tasklist.xml: this file records every task in Mail Service.
- Log file: this file is a record of everything that happens in Mail Service.

In Mail Service there are two subservices: one is to process the request, another is to register new user.

- Register subservice: every mail address should be in Mail Service registered. The registration method is to use the mail address to send a mail to the administrator mailbox with a subject “register” or including a word “register”. Register subservice checks every mail in administrator mailbox, whether this mail is a register mail or not. If this mail is a register mail, this mail address will be added into Userlist.xml. For unregistered address an information mail will be sent from administrator and the human user can reply this mail to register for Mail Service. This register subservice runs automatically.
- Process request subservice: after this subservice receives a request, a new mail will be created and sent to the corresponding user through mail server according to this request. If this request needs a response, the subservice will regularly receive mail from administrator mailbox until the timeout. If this mail is replied, the reply will be checked, whether it is valid or not. The valid reply will be as a response returned to human communication manager.

3.4.2 Problem of First Concept

This concept has a major weakness, it is unreliable. Before Mail Service returns a response to the requestor, the emergence of the error may cause data loss, shown in Figure 12.

1. Example:

During the sending server is restarted. After the restart the request is lost.

2. Example:

An error has occurred in register subservice. A mail address is not registered successfully, and the request to this mail address will be declined.

3. Example:

The connection to administrator's mail server is failed. The request is not completed, but Mail Service can not process it again.

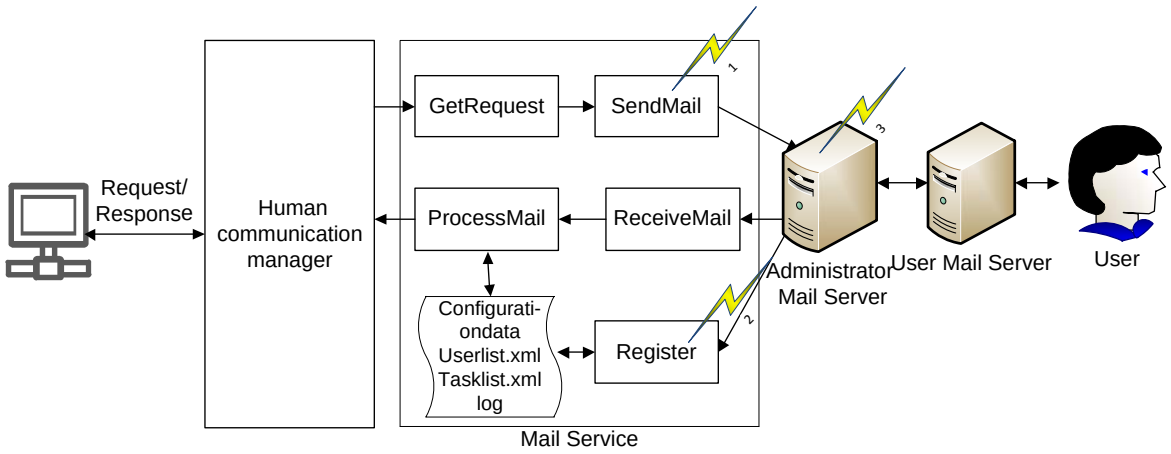

Figure 12. Data Loss

Second weakness is the independence of function modules. In process request Subservice there are four function modules: GetRequest, SendMail, ReceiveMail, and ProcessMail. As long as any function has problem, it will lead to the failure of the entire subservice.

Third weakness is instability. Two subservices should always repeatedly connect the mail server and read the content in mailbox of administrator. But the connection is instable. Once the connection fails, the entire mail service will be in a state of paralysis.

3.4.3 Robustness

Strong robustness [20] is defined in computer science, that computer system has the ability to cope with errors during execution or the ability of an algorithm to continue to operate despite abnormalities in input. For interaction between process and human user a strong robustness is very important. In communication services human users as respondents can give any form of response. That is uncertainty. This requires that communication system can handle all kinds of unexpected circumstances. For example, response's template should be flexible in order to expand the scope of validation. For a variety of invalid reply, communication system is not terminated, but can make the appropriate processing.

3.4.4 New Concept

In order to overcome the weaknesses mentioned above, a new concept is developed and makes improvements in the following points. The following Figure 13 is the architecture of the new concept.

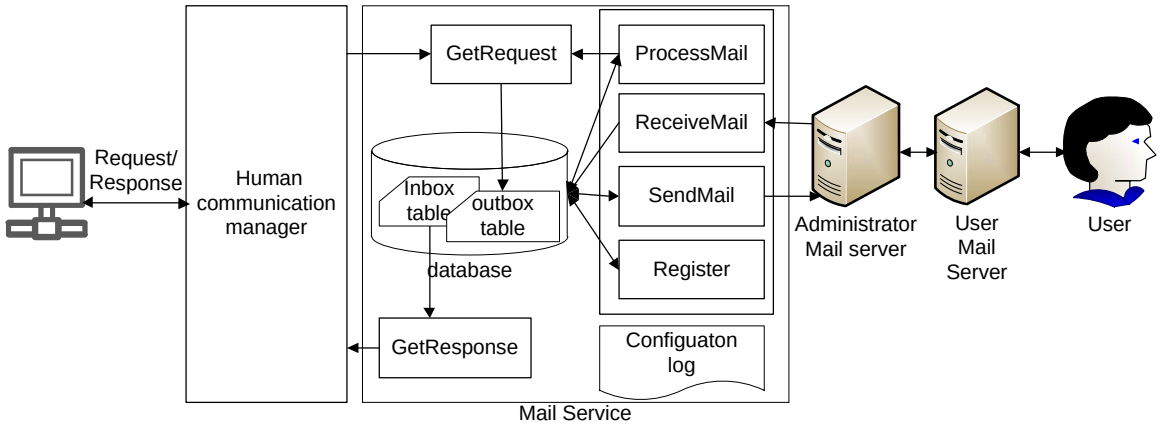

Figure 13. Architecture of new concept

There are two main tables in the database. One stores all requests from external and internal, similar to the mailbox Outbox. An external request is a request sent by an external process which wants to call Mail Service. An internal request is a request generated by Mail Service. The other stores all mails, that the administrator receives, similar to the mailbox Inbox. Mail Service by GetRequest subservice gets each request from an external process and puts the request in Outbox table. SendMail subservice reads every new request in Outbox table and according to this request sends a mail. ReceiveMail subservice puts each new mail in Inbox table. ProcessMail subservice processes the mail in Inbox table. Mail Service by GetResponse subservice returns a message to the external process.

3.4.4.1 Guaranteed Delivery

In communication sender, receiver, and network connecting the two do not all have to be working at the same time. If the network is not available, the messaging system stores the message until the network becomes available. Likewise, if the receiver is unavailable, the messaging system stores the message and retries delivery until the receiver becomes available [15], shown in Figure 14.

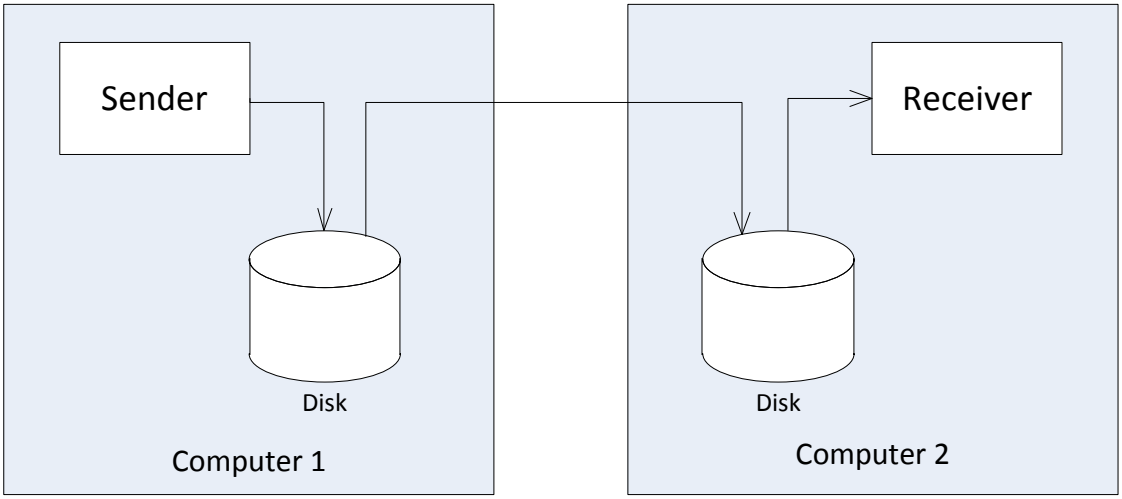

Figure 14. Guaranteed Delivery

Similarly in Mail Service, in order to improve reliability a database will be added in new concept. This database stores all mails from the administrator's mailbox, the register information, all sent mails.

3.4.4.2 Independent subservices

GetRequest subservice is a public service, and other external services can call this subservice in order to send a request to Mail Service. This GetRequest subservice receives a request and stores it in database. It runs only when someone calls it.

ReceiveMail, ProcessMail, SendMail, Register as internal subservices run in parallel, regularly and independently. They should start when the server starts. ReceiveMail stores any new mails from administrator's mailbox in database. SendMail sends a mail according to a request. ProcessMail processes every mail in database. Register stores a new registered address in database.

GetResponse subservice is a public service, and an external service can call this subservice to query whether some request has a response or not. To specify a request an id for the request is required and passed into GetResponse subservice as input parameter.

3.4.4.3 Stability

In order to improve stability, only two subservices have to connect with mail server. They are SendMail and ReceiveMail. Other subservices only need to connect with database to read and write data. The Connection with database is more stable than the connection with mail server.

3.4.5 Database Concept

In the new concept a database is a primary storage. All requests and mails from mail server will be stored in this database. This approach is to enhance the reliability and shorten the time for data processing. As public storage this database must provide all appropriate data for every subservice.

GetRequest subservice

An external process calls GetRequest subservice to send a request, so GetRequest subservice should store every received valid request in database and return a message for this request to the process.

Register subservice

Register subservice should store information of every registered mail in database.

ReceiveMail, SendMail, ProcessMail subservices

Similar to the mail account for these three subservices the database should provide all information of every mail in inbox and in outbox. In order to identify one mail every mail must have a unique identifier.

GetResponse subservice

GetResponse subservice extracts a response for a request from the database. To specify a request a unique identifier for the request will be passed into GetResponse subservice as input parameter.

3.4.6 Classes of Mails

In inbox of mail account all mails can be divided into three types: register mails, response mails, and other mails. The other mails are generally advertising mails from this mail account provider and these mails make no sense for Mail Service. In outbox of the mail account every mail is a request mail. Based on the above analysis there are five classes of mail in mail service:

- NOTIFICATION: One mail that belongs to this class is a mail sent from the administrator.
- REGISTER: One mail that belongs to this class is a register mail from a new user.
- REQUEST: One mail that belongs to this class is a request from the process.
- RESPONSE: One mail that belongs to this class is a response.
- UNKNOWN: All mails that do not belong to other three classes. They are generally advertising mails.

3.4.7 Mail States

State diagrams [21] are used to give an abstract description of the behaviour of a system. This behaviour is analyzed and represented in series of events that could occur in one or more possible states. Figure 15 is a state diagram for Mail Service.

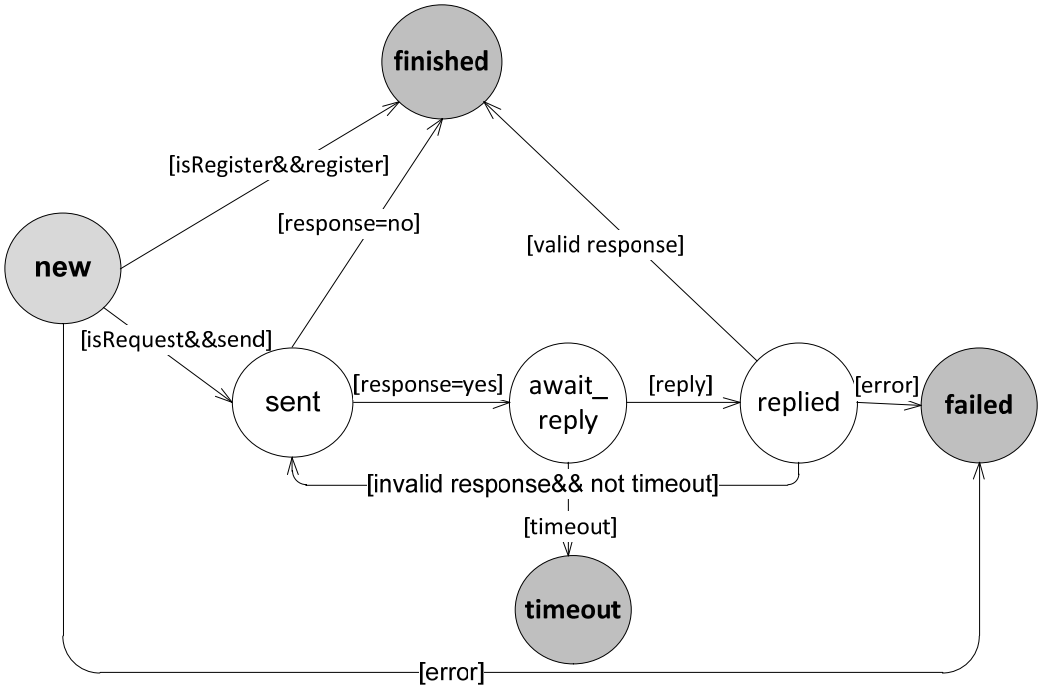

Figure 15. State diagram of Mail Service

7 different states are defined for Mail Service: NEW, SENT, AWAIT_REPLY, REPLIED, FINISHED, FAILED, TIMEOUT. Among them, NEW is a start state, and FINISHED, FAILED, TIMEOUT are end states

Whether it is in the Inbox or Outbox, first all mails will be stored as a new mail [NEW]. Then a new mail will be determined, which class of mail it is, REGISTER, REQUEST or RESPONSE. After registration processing register mail is complete [FINISHED]. If it is a response mail, it may be a valid response, or it may be an invalid response. So its next status may be two states, completely processed [FINISHED] or failed [FAILED]. If it is a request mail, this request is sent out first [SENT], and then is determined whether it needs to wait for a response, if necessary [AWAIT_REPLY], the corresponding response must be handled within certain time limits[REPLIED]. If no response is received within the time limit, this request will be judged to be timeout [TIMEOUT]. Error will result in processing incomplete [FAILED].

3.5 Skype Service

In this work Skype Service has not been completely implemented and it is written as a concept in this chapter. An external process can call Skype Service to send a text message or a file to a human user. Skype Service checks a request, whether it needs a response or not and return a valid response to the external process.

The concept for Skype Service is very similar to the concept for Mail Service. Figure 16 illustrate the components in the architecture of Skype Service. There are database, configuration data, GetRequest, ProcessMessage, ReceiveMessage, SendMessage, GetResponse and Register. Every message has seven different states: NEW, SENT, AWAIT_REPLY, REPLIED, FINISHED, FAILED, TIMEOUT. Among them, NEW is a start state, and FINISHED, FAILED, TIMEOUT are end states. There are three classes of the received messages: RESPONSE, REGISTER, UNKNOWN.

Some tables will be created in the database to store dates for different themes, for example, a table for all sent text message, a table for all sent file, a table for all received text message, a table for all received file, and a table for Skype username of all register.

GetRequest gets a request from an external process, checks the validation of this request and puts the valid request in the database. ProcessMessage, ReceiveMessage, SendMessage, Register runs in parallel in the background. They get the dates only from the database and are independent each other. The external process can call GetResponse to query, whether Skype Service has received a valid response for some request.

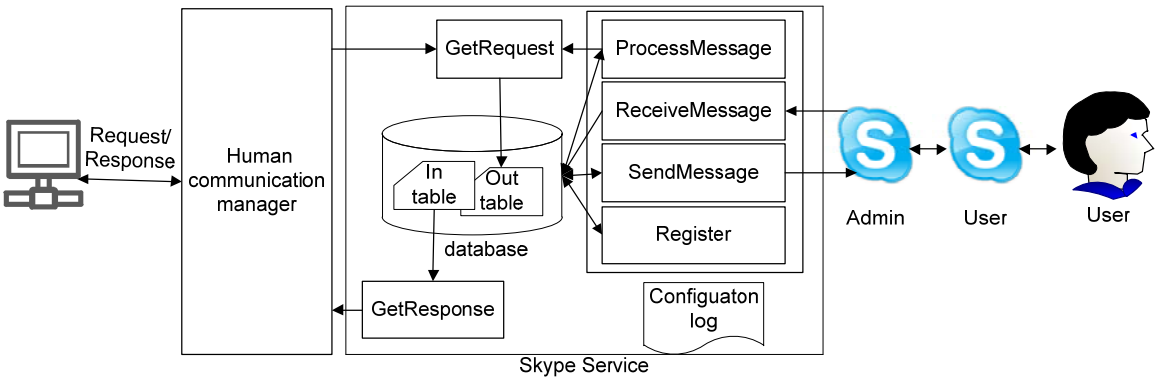

Figure 16. Skype Service architecture

There are several new points for Skype Service:

For Skype Service application Skype is required and must be started and log in with administrator account before Skype Service runs.

To send a message to someone via Skype the username of the recipient must be added into the administrator's contact list before.

In a mail text and file can be sent at the same time. But in Skype a message to be sent is either a text or a file at the same time.

All contacts of administrator's contacts list have different state, for example, online, away, Do Not Disturb. If a contact is online, the communication with this contact may be synchronous. Compared to mail system asynchronous communication is one of the most significant features.

4 Specifications

Before the implementation starts, the specifications for interface parameter of Mail Service and Skype Service and configuration dates will be determined in this chapter.

4.1 Interface Parameters

4.1.1 Mail Service

An external process can call GetRequest subservice to send a request. For a mail request some parameters are required. For example, address of recipient, subject, content and attachments. In addition, three parameters for a response are also required. One indicates, whether the request requires a response or not (parameter response). The other defines the time limit for replying (parameter timeout). The third is the template of a response (parameter responseTemplate).

According to the above mentioned, 7 parameters will be defined as interface parameters in GetRequest subservice shown in Figure 17: recipient, subject, message, attachments, timeout, response, responseTemplate. Among them parameter response indicates, whether the request requires a response or not. It has two value “yes” or “no”.

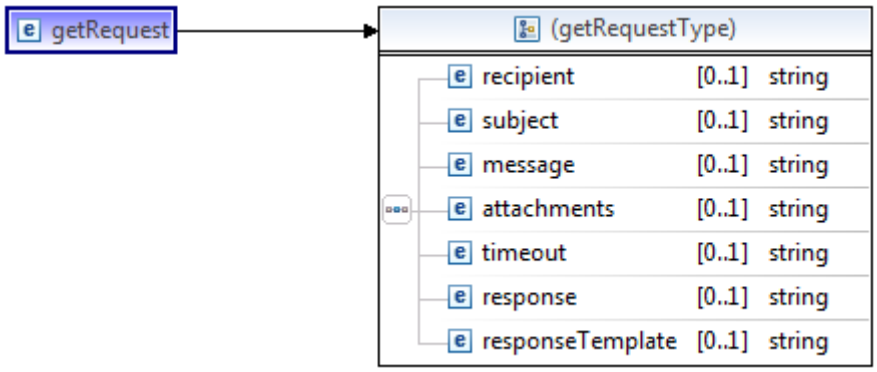

Figure 17. Input parameters for GetRequest subservice in Mail Service

As return value a string should be returned to notify, whether calling Mail Service is successful or not, shown in Figure 18.

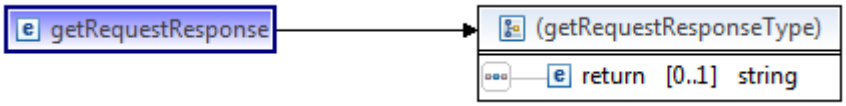

Figure 18. Output parameter for GetRequest subservice in Mail Service

4.1.2 Skype Service

SkypeGetRequest subservice in Skype Service is an interface to the external. It is responsible for getting a request from an external process which sends a request to Skype Service. To send a message via Skype Skype id and text are required. Similar to the above input definition of Mail Service parameter response, timeout and responseTemplate are also required. Because Skype sends either a text message or a file at the same time, a parameter messageType is required to indicate the type of a message, a text message or file. Figure 19 shows the definition of input parameters for Skype Service. Output parameter for Skype Service has the same structure of Mail Service in Figure 18.

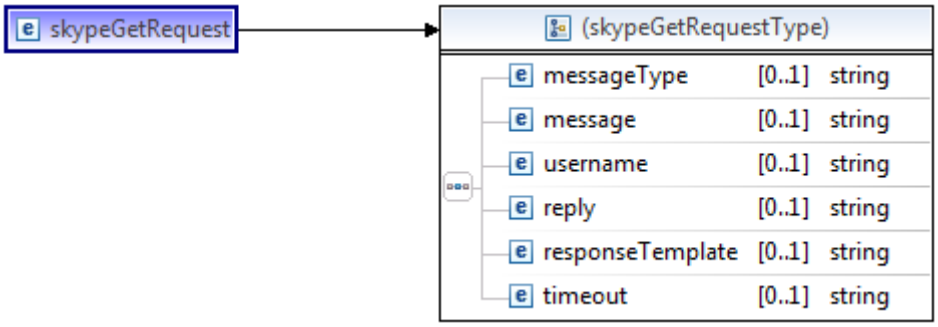

Figure 19. Input parameters for SkypeGetRequest subservice in Skype Service

4.2 Configuration

File configuration.conf is a list of configurationsarguments. After these arguments in this file are set, they are in the entire communication service fixed. Every time the service is called, these arguments will be read from this file.

Table 1 lists the arguments for Logging of Mail Service.

Argument	Description
GetRequestLogFilePath	location of logfile for GetRequest subservice
ReceiveMailLogFilePath	location of logfile for ReceiveMail subservice
SendMailLogFilePath	location of logfile for SendMail subservice

RegisterLogFilePath	location of logfile for Register subservice
ProcessMailLogFilePath	location of logfile for ProcessMail subservice
MailLayout	format for the output of an appender
MailLogFileSize	size of logfile
MailMaxBackupIndex	limit number of backups
MailAppender	an output destination
MailPattern	message format

Table 1. Arguments for Logging of Mail Service in configuration.conf

Table 2 lists the arguments for Administrator's account of Mail Service.

Argument	Description
MailAddress	address of Administrator's account
ReceiveHost	host for receiving a mail
ReceivePort	port for receiving a mail
SendHost	host for sending a mail
SendPort	port for sending a mail
Username	username of administrator's mailbox
Password	password of administrator's mailbox

Table 2. Arguments for Administrator's account of Mail Service in configuration.conf

Table 3 lists the arguments for the Mail Service database in configuration.conf.

Argument	Description
db_username	username of database
db_password	password of database
db_url	url of database
db_table_mailbox	tablename
db_table_sentmails	tablename
db_table_mailattachments	tablename

db_table_mailregister	tablename

Table 3. Arguments for the Mail Service database in configuration.conf

Table 4 lists the argument for saving attachments of Mail Service.

Argument	Description
AttPath	location to save the attachments

Table 4. Argument for saving attachments of Mail Service

Table 5 lists the arguments for Administrator's account of Skype Service.

Argument	Description
skype_username	username of Skype application
skype_password	password

Table 5. Arguments for Administrator's account of Skype Service in configuration.conf

Table 6 lists the arguments for Logging of Skype Service.

Argument	Description
SkypeGetRequestLogFilePath	location of logfile for SkypeGetRequest subservice
ReceiveMessageLogFilePath	location of logfile for ReceiveMessage subservice
SendMessageLogFilePath	location of logfile for SendMessage subservice
SkypeRegisterLogFilePath	location of logfile for Register subservice
ProcessMessageLogFilePath	location of logfile for ProcessMessage subservice
SkypeLayout	format for the output of an appender
SkypeLogFileSize	size of logfile
SkypeMaxBackupIndex	limit number of backups
SkypeAppender	an output destination
SkypePattern	message format

Table 6. Arguments for Logging of Skype Service in configuration.conf

Table 7 lists the arguments for the Skype Service database in configuration.conf, i.e. the information of connecting to the database.

Argument	Description
skydb_username	username of database
skydb_password	password of database
skydb_url	url of database
skydb_table_register	tablename
skydb_table_senttextmessage	tablename
skydb_table_sentfilemessage	tablename
skydb_table_receivedtextmessage	tablename
skydb_table_receivedfilemessage	tablename

Table 7. Arguments for the Skype Service database in configuration.conf

5 Implementation

This chapter describes all components in Mail Service and Skype Service that have been implemented. Bottom-up approach is used to develop Web Services. First a component level is explained – which components exist. Then the internal structure of each component is described. Afterwards, the corresponding Java classes are described.

5.1 Components Overview

Communication services have two channels, Mail Service and Skype Service. The two services are independent of each other. Database provides a stable and secure source of data for two services. All data will be stored in the database and read from it. File configuration.conf is an external data. It provides arguments to connect to the database, set properties of log files and read administrator's data. This file can be modified later. Figure 20 shows the components of communication services.

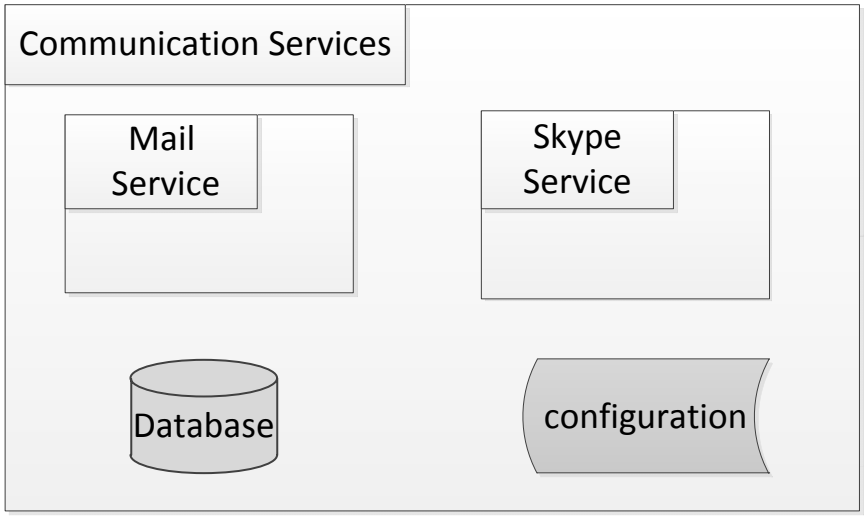

Figure 20. Components in communication service

5.2 Database MySQL

In order to improve service reliability and data security, database design is required. In this thesis MySQL will be used.

MySQL is a database management system that is particularly well suited for use to the Internet. It is relatively fast and reliable. The data can be managed using the internal SQL command set. It is multi-user and multi-tasking capability, and handles large amounts of data quickly and is relatively stable. MySQL is included in databases, tables and auxiliary data. There are also tools for editing the data. MySQL also has a set of instructions, which is similar to the standard database query language SQL. MySQL can also be as open-source distribution from the Internet and used accordingly.

Specific information of MySQL:

- MySQL Server 5.5 – database server
- MySQL Workbench 5.2 – tool for manage the database

Before using MySQL with the plan, an account must be set up.

- Username: root
- Password: root

5.2.1 Database in Mail Service

For Mail Service database “maildb” is created. Figure 21 shows five tables in maildb database.

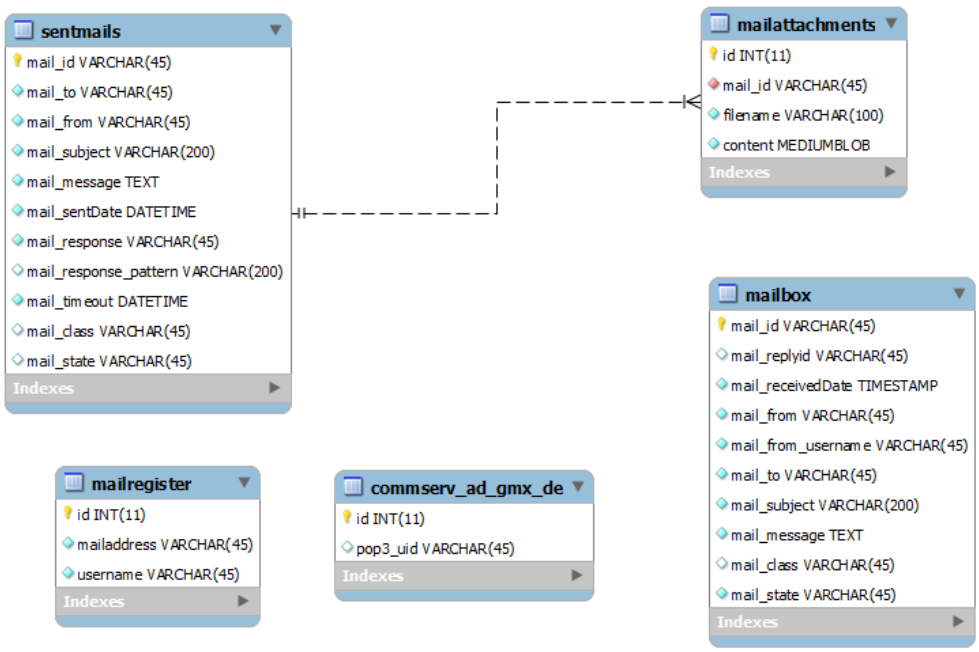

Figure 21. Five tables in database maildb

1) maildb.sentmails

This table stores request data from a business process, which calls Mail Service. A request requires a unique identifier for Mail Service, the recipient's address, the subject of mail, the context of mail, the information, whether this mail needs a response, the class of mail, the state of mail. There is also optional information: the sender's address; if this mail needs a response, which form the response should have; the timeout, the current system time, when this mail will be sent by Mail Service.

Column Name	Datatype	value	Description
mail_id	VARCHAR(45)	UUID	unique identifier
mail_to	VARCHAR(45)	mail address	recipient's address
mail_from	VARCHAR(45)	mail address	sender's address
mail_subject	VARCHAR(200)	string	subject of mail
mail_message	TEXT	string	context of mail
mail_sentDate	DATETIME	YYYY-MM-DD hh:mm:ss	time, when this mail will be sent
mail_response	VARCHAR(45)	yes or no	whether this mail needs a response
mail_response_pattern	LONGTEXT	xml schema	form of response
mail_timeout	DATETIME	tt:hh:mm:ss, (00:10:15:00 means 10 hours 15 minutes time limit)	time limit
mail_class	VARCHAR(45)	NOTIFICATION, REQUEST, RESPONSE, REGISTER, UNKNOWN	class of mail
mail_state	VARCHAR(45)	NEW, SENT, AWAIT_REPLY, REPLIED, FAILED, TIMEOUT, FINISHED	state of mail

Table 8. Definition of table sentmails

2) maildb.mailbox

All mails in the inbox of the administrator's mail box will be imported into this table.

Column Name	Datatype	Value	Description
mail_id	VARCHAR(45)	UUID	unique identifier
mail_replyid	VARCHAR(45)	UUID	unique identifier of request
mail_receivedDate	TIMESTAMP	YYYY-MM-DD hh:mm:ss	when this mail has been received
mail_from	VARCHAR(45)	mail address	sender's address
mail_from_username	VARCHAR(45)	string	registered username
mail_to	VARCHAR(45)	mail address	recipient's address
mail_subject	VARCHAR(200)	string	subject of mail
mail_message	LONGTEXT	XML	context of mail
mail_class	VARCHAR(45)	NOTIFICATION RESPONSE REGISTER UNKNOWN	class of mail
mail_state	VARCHAR(45)	NEW SENT AWAIT_REPLY REPLIED FAILED TIMEOUT	state of mail

Table 9. Definition of table mailbox

3) maildb.mailregister

This table mailregister saves mail address and username. After register Mail Service can send a mail to this address. So every new mail address must be registered by sending a mail containing the word "register" in its subject or replying a register request from Mail Service. The username and mail address will be extracted from his mail.

Column Name	Datatype	Description
id	INT(11)	own id
mailaddress	VARCHAR(45)	
username	VARCHAR(45)	

Table 10. Defintion of table mailregister

4) maildb.mailattachments
   This mailattachments table manages all attachments of mails in Mail Service. Each attachment record has own id, from which mail, filename and content.

Column Name	Datatype	Description
id	INT(11)	own id
mail_id	VARCHAR(45)	point to his mail
filename	VARCHAR(100)	
content	MEDIUMBLOB	

Table 11. Defintion of table attachments

5) maildb.commserv_ad_gmx_de
   The purpose of this table is to judge, whether a mail in the mailbox is a new mail. POP3 supports no permanent flags. In particular, the Flags.Flag.RECENT flag will never be set for POP3 message. This is one drawback of POP3 Server. It is up to determine which messages in a POP3 mailbox are “new”. There are several strategies to accomplish this, depending on the needs of the application and the environment [22]:

- A simple approach would be to keep track of the newest message seen by the application.
- An alternative would be to keep track of the unique identifiers (UIDs) of all messages that have been seen.
- Another approach is to download all messages into a local mailbox, so that all messages in the POP3 mailbox are, by definition, new.

A unique identifier (UID) [23] is an arbitrary server-determined string, consisting of one to 70 characters in the range 0x21 to 0x7E, which uniquely identifies a message within a mail drop and which persists across sessions.

Column Name	Datatype	Description
id	INT(11)	
pop3_uid	VARCHAR(45)	

Table 12. Definition of table commserv_ad_gmx_de

5.2.2 MySQLAccess in Mail Service

This Java class is responsible for connecting and disconnecting to the database maildb. It is placed in following Java package:

```
de.unistuttgart.iaas.interaction.mail.database
```

MySQL offers standard database driver connectivity for using MySQL with applications and tools that are compatible with industry standards ODBC and JDBC [24]. To connect to MySQL in Java Program MySQL Connector/J driver is required. It is the official JDBC driver for MySQL.

The file name of the MySQL Connector/J driver is

```
mysql-connector-java-5.1.18-bin.jar
```

The Path is added into the classpath and the connection url must be changed.

```
url=jdbc:mysql://localhost/maildb
```

This class offers two functions:

1) Connect to the database and return an interface object Connection.

2) Disconnect to the databse.

Function1 is implemented by the public method connDataBase. It takes the data from configuration.conf. First it loads driver class by the method Class.forName(String driver) (line 4). Then it establishes a connection to specified database URL (line 5). After the connection is established, all tables for subservices must be checked. If the tables do not exist in this database, they must be created (line 6- 17). Finally this connection is returned (line 31). The method, that does this, is shown in Listing 3.

```
1.  public Connection connDataBase(ConfigFileHandler config) throws Exception{
2.  …
3.  try{
4.    Class.forName("com.mysql.jdbc.Driver");
5.    connect=DriverManager.getConnection(db_url, db_username,db_password);
6.    String table1Sql = "create table if not exists maildb."+table_mailbox…
7.    String table2Sql = "create table if not exists maildb."+ table_sentmails…
8.    String table3Sql = "create table if not exists maildb."+ table_attachments…
9.    String table4Sql = "create table if not exists maildb."+ table_mailids…
10.   String table5Sql = "create table if not exists maildb."+ table_register…
11.   if(connect!=null){
12.        statement = connect.createStatement();
13.        statement.executeUpdate(table1Sql);
14.        statement.executeUpdate(table2Sql);
15.        statement.executeUpdate(table3Sql);
16.        statement.executeUpdate(table4Sql);
17.        statement.executeUpdate(table5Sql);
18.    }else{
19.     logger.error("No database.");
20.     System.out.println("No database.");
21.    }
22.  }catch(ClassNotFoundException e){
23.     Session session = AdminConnector.SendConnector(config);
24.     ErrorHandler.sendErrorMessageForConnDB(session, config, e);
25.     logger.error("Failed to load the MySQL drive");
26.  }catch (SQLException e1) {
27.     Session session = AdminConnector.SendConnector(config);
28.     ErrorHandler.sendErrorMessageForSQL(session, config, e1);
29.     logger.error("Failed to connect database.");
30.  }
31.     return connect;
32. }
```

Listing 3. Database connection method

The input variable `config` is defined in class `ConfigFileHandler`. It reads full information from the file configuration.conf. In Listing 4 all arguments in the file configuration.conf are listed (line 8-27).

```
1.  public class ConfigFileHandler
2.  {
3.      protected String strFile = "";
4.      FileInputStream propInFile = null;
5.      public Properties propConfig = new Properties();
6.      private static Logger logger =
7.      Loger.getLogger(ConfigFileHandler.class);
8.      static String MailAddress = "MailAddress";
9.      static String RecieveHost = "RecieveHost";
10.     static String RecievePort = "RecievePort";
11.     static String SendHost = "SendHost";
12.     static String SendPort = "SendPort";
13.     static String Username = "Username";
14.     static String Password = "Password";
15.     static String AttPath = "AttPath";
16.     static String db_username ="db_username";
17.     static String db_password ="db_password";
18.     static String db_url = "db_url";
19.     static String db_table_mailbox = "db_table_mailbox";
20.     static String db_table_sentmails = "db_table_sentmails";
21.     static String db_table_mailattachments = "db_table_mailattachments";
22.     static String db_table_mailregister = "db_table_mailregister";
23.     static String xsdHeader = "xsdHeader";
24.     static String xsdFoot = "xsdFoot";
25.     static String xmlHeader = "xmlHeader";
26.     static String xmlFoot = "xmlFoot";
27.     static String tag = "tag";
28. …}
```

Listing 4. Arguments in configuration.conf are listed in class ConfigFileHandler

Function 2 is implemented by another public method close. It closes object resultSet, statement and connection. The method, that does this, is shown in Listing 5.

```
1.  public void close() {
2.  try {
3.    if (resultSet != null) {
4.      resultSet.close();
5.     }
6.     if (statement != null) {
7.      statement.close();
8.     }
9.     if (connect != null) {
10.       connect.close();
11.    }
12.   } catch (Exception e) {
13. }
```

Listing 5. Database close( ) methode

5.2.3 Database in Skype Service

For Skype Service database “skydb” and five tables are created.

1) skydb.register

This table register saves id and username of a Skype user. After register Skype Service can send a message to this Skype user.

Column Name	Datatype	Description
idregister	INT(11)	own id
skype_username	VARCHAR(45)	Skype id
name	VARCHAR(45)	Skype username

Table 13. Defintion of table skydb.register

2) skydb.senttextmessage

This table saves all requests which want to send a text message to a Skype user. This text message will be stored as a string.

Column Name	Datatype	Description
id	VARCHAR(45)	UUID
to_username	VARCHAR(45)	Skype id of recipient
from_username	VARCHAR(45)	Skype id of Administrator
textMessage	LONGTEXT	content of message
sentDate	DATETIME	time, when this message will be sent
reply	VARCHAR(45)	whether this request needs a response
responsePattern	LONGTEXT	form of response
timeout	DATETIME	time limit
state	VARCHAR(45)	state of message

Table 14. Definition of table skydb.senttextmessage

3) skydb.sentfilemessage

This table saves all requests which want to send a file message to a Skype user.

Column Name	Datatype	Description
id	VARCHAR(45)	UUID
to_username	VARCHAR(45)	Skype id of recipient

from_username	VARCHAR(45)	Skype id of Administrator
fileMessage	BLOB	content of message
sentDate	DATETIME	time, when this message has been received
reply	VARCHAR(45)	whether this request needs a response
responsePattern	LONGTEXT	form of response
timeout	DATETIME	time limit
state	VARCHAR(45)	state of message

Table 15. Definition of table skydb.sentfilemessage

4) skydb.receivedtextmessage

This table saves all text messages which are received by administrator's Skype client.

Column Name	Datatype	Description
id	VARCHAR(45)	UUID
requestId	VARCHAR(45)	UUID of the request
from_username	VARCHAR(45)	Skype id of Administrator
textMessage	LONGTEXT	content of message
receivedDate	DATETIME	time, when this mail will be sent
class	VARCHAR(45)	class of message
state	VARCHAR(45)	state of message

Table 16. Definition of table skydb.receivedtextmessage

5) skydb.receivedfilemessage

This table saves all files which are received by administrator's Skype client.

Column Name	Datatype	Description
id	VARCHAR(45)	UUID
requestId	VARCHAR(45)	UUID of the request
from_username	VARCHAR(45)	Skype id of Administrator

fileMessage	LONGTEXT	content of message
filename	VARCHAR(45)	name of the file
receivedDate	DATETIME	the time, when this message has been received.
class	VARCHAR(45)	class of message
state	VARCHAR(45)	state of message

Table 17. Definition of table skydb.receivedfilemessage

5.2.4 MySQLAccess in Skype Service

This Java class is responsible for connecting and disconnecting to the database skydb. It is placed in following Java package:

```
de.unistuttgart.iaas.interaction.skype.database
```

The Path is added into the classpath and the connection url must be changed.

```
url=jdbc:mysql://localhost/skydb
```

This class also offers two functions:

1) Connect to the database and return an interface object Connection.
2) Disconnect to the databse.

5.3 Mail Service

Mail Service consists of multiple subservices. Each will be described in this chapter. Figure 22 shows the components diagram from Mail Service.

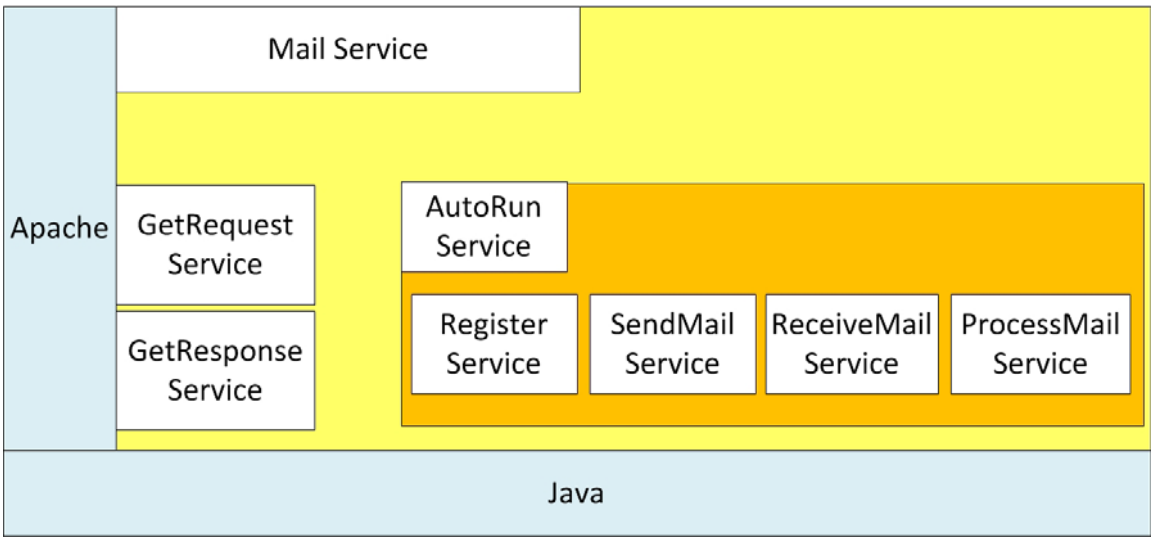

Figure 22. Components diagram for Mail Service

5.3.1 Apache Axis2

Apache Axis2 [27] is a Web Services/ SOAP/ WSDL engine, which is present with program languages Java and C and it is supported by Apache-Licence 2.0. It is the successor to the widely used Apache Axis SOAP stack. The W3C specifications have been realized for Axis2, for example, WS-Addressing, WS-ReliableMessaging, WS-MetadataExchange, WS-Policy, WS-AtomicTransaction and WS-Security.

A tool in Axis2 is `Java2WSDL`, which reads a java class and generates a WSDL for invoking the class methods as a Web Service. This is known as Bottom-Up approach to Web Service development.

5.3.2 JavaMail API

Mail Service in this thesis uses JavaMail API [28]. This API is used to receive and send mail via SMTP, POP3 and IMAP. Through the JavaMail API these mailing functionalities can be added to a Java application. The JavaMail API provides a platform-independent and protocol-independent framework to build mail and messaging applications.

5.3.3 GetRequest Service

This subservice's main purpose is to get a request from a business process, check the validity and save it in the table maildb.sentmails. It is placed in following Java package:

```
de.unistuttgart.iaas.interaction.mail.services
```

Main function is implemented by the public and static method `getRequest`. Figure 23 shows the function of GetRequest service step by step.

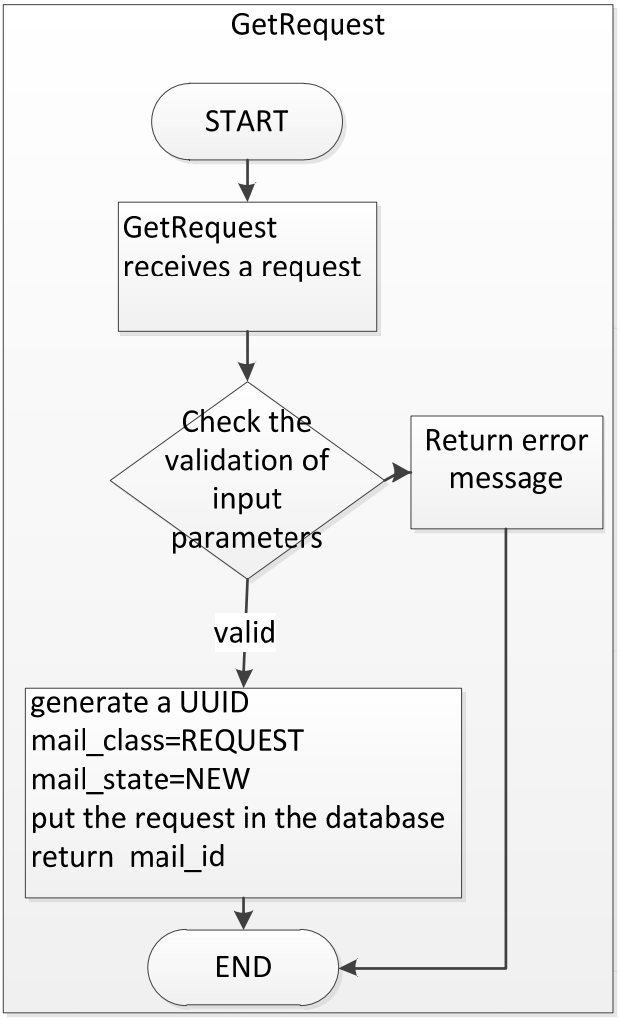

Figure 23. Flow Diagram for GetRequest

GetRequest service takes a list of input variables and checks the validity of a request. After some transformations it saves a valid request in the database, table maildb.sentmails.

Request validity is defined as follows:

1) `recipient`, `subject`, `message`, `response` can not be null.
2) Format for `timeout` is 00:00:00:00. The first number from the left means days; the second number means hours; the third number means minutes and the last number means seconds. For example 00:10:05:00 represents that time limit is 10 hours and 5 minutes. `Timeout` can be null.
3) If the value `response` is yes, `responseTemplate` can be not null.

All requests in line with the above conditions are valid for Mail Service. Mail Service will generate a universally unique identifier (UUID) for each valid request. This UUID will be

stored in the database table maildb.sentmails as mail_id. Then some transformations must be done to meet the structure of the database table. In Table 18 `timeout` and `responseTemplate` must be transformed. The transformation from timeout to mail_timeout is simple, for instance, current time is 2012-05-03 13:00:56 and the value of timeout is 00:03:05:00, then the value mail_timeout is 2012-05-03 16:05:56.

Input variable	Field in the table maildb.sentmails	Transformation
recipient	mail_to	the same value
subject	mail_subject	the same value
message	mail_message	the same value
timeout	mail_timeout	current time + timeout
response	mail_response	the same value
responseTemplate	mail_response_pattern	convert into Xml Schema

Table 18. Input variable transformations

The transformation from `responseTemplate` to `mail_response_pattern` is a transformation from XML format to XML Schema format. This function implemented by the public and static method `getResponseTemplate`. It takes the context of `responseTemplate` and finally returns a string which context is based on XML Schema.

The context of `responseTemplate` is respresented by a string, which is based on xml format. It contains four tags, `<wsResponse>`, `<name>`, `<type>`, `<enumeration>`, `<min>`, `<max>`. For multiple questions in a request each question must be named. This name is the correlation between question and its answer. An answer can be an integer or a string. It can be enumerated. If it is an integer, his range can be set. Listing 6 is an instance of `responseTemplate`. This instance contains two templates of two answers. One is for the question with the name “question”. This answer is a natural number between 1 and 10. The other is for the question with the name “question2”. This answer is a string, either yes or no.

```
1.  <ser:responseTemplate>
2.  <wsResponse>
3.      <name>question1</name>
4.      <type>integer</type>
5.       <enumeration/>
6.       <min>1</min>
7.       <max>10</max>
8.  </wsResponse>
9.  <wsResponse>
10.     <name>question2</name>
11.     <type>string</type>
12.     <enumeration>yes|no</enumeration>
13.     <min/>
14.     <max/>
15. </wsResponse>
16. </ser:responseTemplate>
```

Listing 6. An instance of responseTemplate

The transformation function is implemented by the public and static method `getResponseTemplate`. It is placed in following Java package and class:

`de.unistuttgart.iaas.interaction.mail.util`

and

`CommonTools.java`

After the transformation a piece of XML Schema code will be generated. Listing 7 is the XML Schema code corresponding the above responseTemplate instance in Listing 6. GetRequest service set the felds `mail_class` and `mail_state` in the maildb.sentmails for a request as fixed value, REQUEST and NEW. If a request has an attachment, then the UUID, the name and the content of attachment will be stored in the database table maildb.attachments.

Once a request is stored in the database maildb.sentmails, another service SendMail service can process this request.

```
1. <element name='response'>
2.   <complexType>
3.      <sequence>
4.        <element name='question1' type='tns:question1'/>
5.        <element name='question2' type='tns:question2'/>
6.      </sequence>
7.   </complexType>
8. </element>
9. <simpleType name='question1'>
10.  <restriction base='integer'>
11.     <minInclusive value='0'></minInclusive>
12.     <maxInclusive value='10'></maxInclusive>
13.  </restriction>
14.</simpleType>
15.<simpleType name='question2'>
16.  <restriction base='string'>
17.     <enumeration value='yes'></enumeration>
18.     <enumeration value='no'></enumeration>
19.  </restriction>
20.</simpleType>
```

Listing 7. Xml Schema code for response template

5.3.4 SendMail Service

This subservice selects all requests with mail_state=NEW in the database table maildb.sentmails and according to a selected request SendMail service sends a mail by the administrator' mailbox. It is placed in following Java package:

`de.unistuttgart.iaas.interaction.mail`

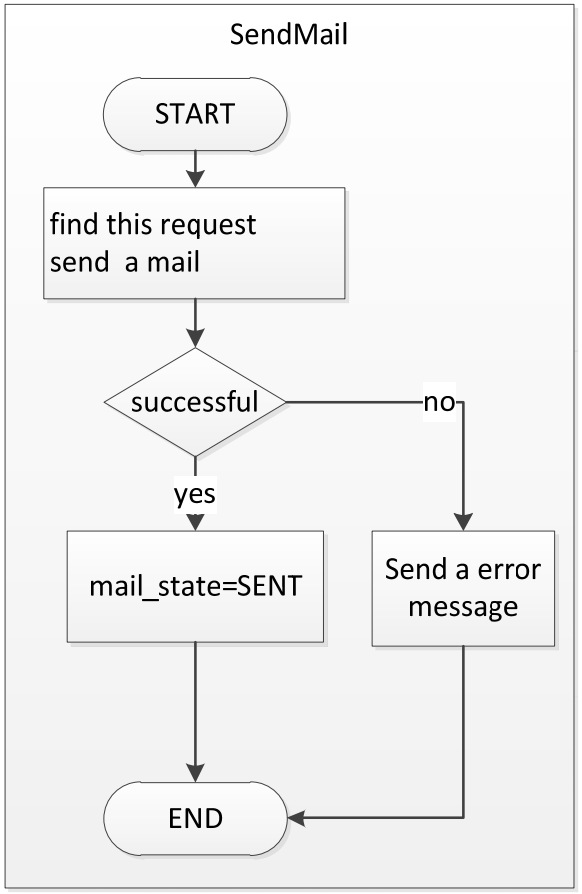

Figure 24. Flow Diagram for SendMail

Main function is implemented by the public and static method `sendMail`. Figure 24 shows the function of SendMail service step by step.

It connects the database, selects each new request in the database table maildb.sentmails and according to this request set all required arguments for a `message` object to send a mail. Listing 8 is a series of data processing in SendMail service. As the correlation between request mail and response mail mail_id is added in the subject of `message` (line 3). By transformation a response format is generated. This transformation is implemented by the public and static method `xsdToXml`. It takes the content of field `mail_response_pattern` and transforms it to the xml formatted string (line 8-9). Then this string will be added in the body of mail (line 20-22). Listing 9 is a response instance according to the above response template. Later when the recipient replies to the mail, with this format he just has a slight modification and then he can make an accurate response.

```
1.  mail_id = (String)sentmail.get(0);
2.  mail_to = (String)sentmail.get(1);
3.  mail_subject = (String)sentmail.get(3)+" Mail-ID:"+mail_id;
4.  mail_message = (String)sentmail.get(4)+"\n";
5.  mail_response = (String)sentmail.get(6);
6.  mail_responseTemplate = (String)sentmail.get(7);
7.  if(!mail_responseTemplate.equals("")){
8.     mail_responseTemplate=
9.       CommonTools.xsdToXml(config,mail_responseTemplate);
10. }
11. Statement usernameStatement = conn.createStatement();
12. ResultSet usernameResultset = usernameStatement.executeQuery(
13. "select username from maildb." + config.getTable_mailregister()+
14. " where mailaddress='"+mail_to +"'");
15. if(usernameResultset.next()){
16.    mail_message="Hello "+usernameResultset.getString("username")+
17.                  ",\n"+mail_message;
18. }
19. if(mail_response.equalsIgnoreCase("yes")){
20.    mail_message = mail_message+"\n\n Requested result for
21.    mat:\n"+mail_responseTemplate+"\n You can modify this format
22.    and copy it in your reply.";
23. }
24. msg.setFrom(new InternetAddress(config.getMailAddress()));
25. msg.setRecipients(Message.RecipientType.TO,
26. InternetAddress.parse(mail_to, false));
27. msg.setSubject(mail_subject);
28. msg.setText(mail_message);
```

Listing 8. Data process in SendMail service

```
<response>
<tns:question1>[1-0(integer)]</tns:question1>
<tns:question2>yes|no</tns:question2>
</response>
```

Listing 9. A response instance

The attachment processing is shown in Listing 10. If a request has no attachment, then only the corresponding message text will be set. If a request has some attachments, then a MimeBodyPart object will be created and the message text will be set (line 4-25).

```
1.   if(list_attId.size()!= 0)
2.   {
3.     //create the message part
4.     MimeBodyPart messageBodyPart = new MimeBodyPart();
5.     messageBodyPart.setText(mail_message);
6.     Multipart multipart = new MimeMultipart();
7.     multipart.addBodyPart(messageBodyPart);
8.     for(int k=0;k<ii;k++)
9.     {
10.     String attId = (String) list_attId.get(k);
11.     Statement sendStatement3 = conn.createStatement();
12.     ResultSet rsAtt = sendStatement3.executeQuery("select * from maildb."
13.     + config.getTable_mailattachments()+" where id='" + attId +"'");
14.     while(rsAtt.next()){
15.       filename=rsAtt.getString("filename");
16.       file = rsAtt.getBlob("content");
17.       CommonTools.saveFile(config.getAttPath()+filename, file);
18.       messageBodyPart = new MimeBodyPart();
19.       DataSource fileDataSource =
20.       new FileDataSource(config.getAttPath()+filename);
21.       messageBodyPart.setDataHandler(new DataHandler(fileDataSource));
22.       messageBodyPart.setFileName(filename);
23.       multipart.addBodyPart(messageBodyPart);
24.       msg.setContent(multipart);
25.    }
26. }
```

Listing 10. Add an attachment in a mail

5.3.5 ReceiveMail Service

This subservice's main purpose is to receive new mails from administrator's mail account. It is placed in following Java package:

`de.unistuttgart.iaas.interaction.mail`

Main function is implemented by the public and static method `receiveMail`. The flow diagram depicted in Figure 25 shows the function of ReceiveMail service step by step.

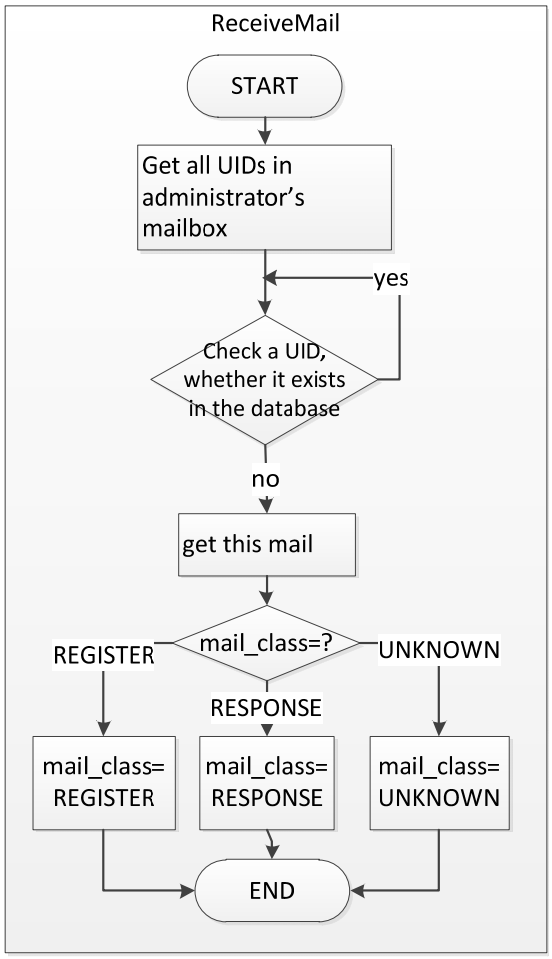

Figure 25. Flow Diagram for ReceiveMail

The main problem is how to determine that a mail in the administrator's mail account is a new mail. For each mail in administrator's mailbox there is a UID. For this mailbox this UID is unique and is saved in a relevant database table. By the method `getUID` in Listing 11 a UID can be read. In this work the table's name is maildb.commserv_ad_gmx_de. All UIDs are saved in this table.

```
String uid = ((POP3Folder) folder).getUID(msgs[i]);
```

Listing 11. Method getUID()

Before saving a mail in the database its UID will be read and compared to each UID in the table maildb.commserv_ad_gmx_de. If this UID exists already in this table, then the mail related to this UID is not a new mail for Mail Service. If this UID does not exist in this table, then the mail is a new mail. This mail will be saved in the database by following a few steps:

1) Get one mail object. This object is a `message` object.
2) Get information from this mail object, for example, sender, recipient, subject and context.
3) Set `mail_class` by the method `mailClassIdentifier`. This method determines the class of a mail. Each mail belongs to a class, for example, NOTIFICATION, REQUEST, RESPONSE and UNKNOWN. A NOTIFICATION mail is from the administrator and needs no response. This mail is finished after sending it. A REQUEST mail is got by GetRequest Service. A RESPONSE mail is a mail starting with a string ”RE:” or “AW:” and containing a UUID in subject. An UNKNOWN mail is a mail which does not belong to the above three classes.
4) Generate a UUID for this mail, set `mail_id`
5) Set `mail_receivedDate` by the method `getSentDate()`,
6) Set `mail_state` as NEW
7) Check whether this mail has attachments. If there is no attachment, put all dates in the database. If there are some attachments, first put the name of each attachment and the content of it in the database table maildb.mailattachements, then put all dates for this mail in the database table maildb.mailbox.

5.3.6 ProcessMail Service

This subservice’s main purpose is to process each mail in the table maildb.sentmails. It is placed in following Java package:

`de.unistuttgart.iaas.interaction.mail`

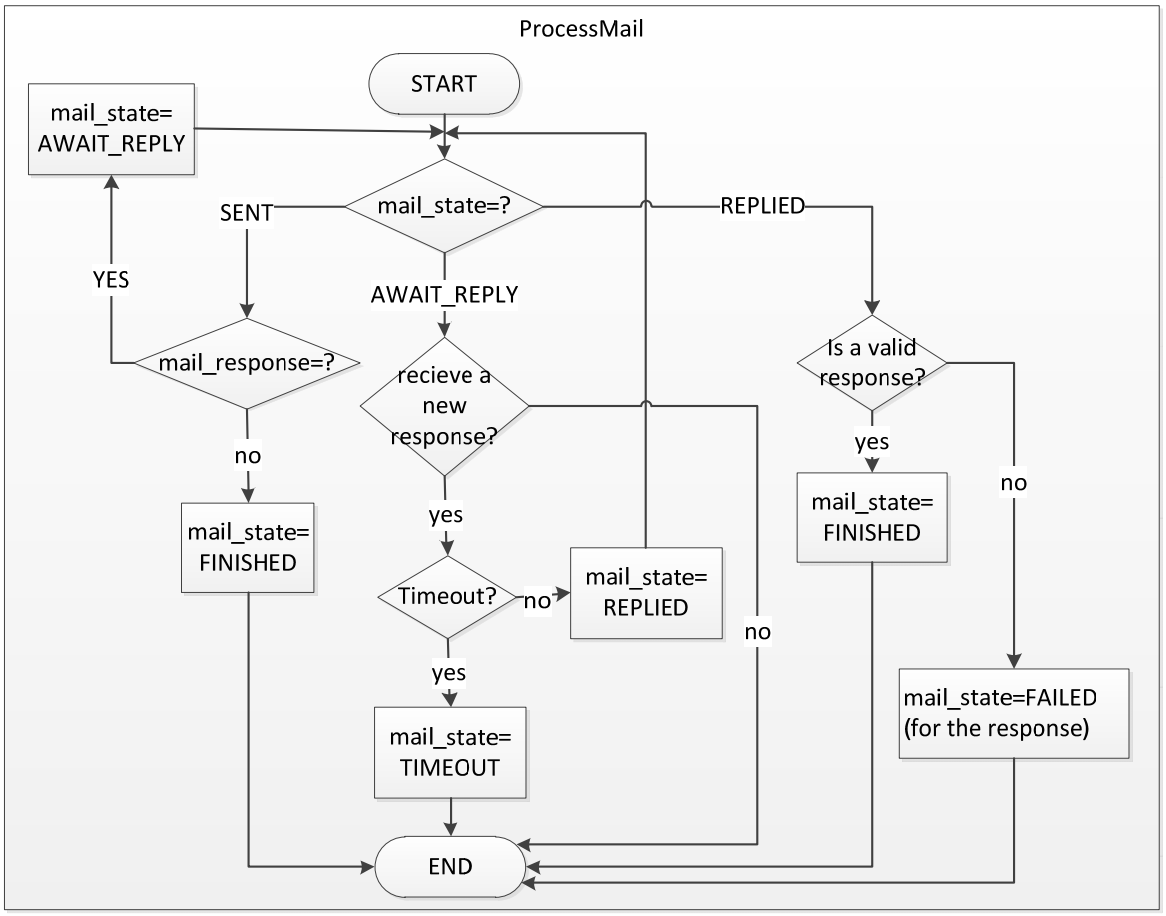

Figure 26. Flow Diagram for ProcessMail

Main function is implemented by the public and static method `ProcessMail`. Figure 26 shows the function of ProcessMail service step by step.

This method processes the mails with “SENT”, “AWAIT_REPLY” or “REPLIED” state. Each mail has a status indicator `mail_state` in the database and the content of this indicator changes at some time. For example, when GetRequest service gets a request from an external process and stores this request in the database, at this time `mail_state` is set as NEW. SendMail service chooses the mails with NEW state in the table maildb.sentmails and processes these mails. If these mails are sent successfully, the states of them are set as SENT. If an error occurs during the sending, then the states of this mail is set as FAILED. In Listing 12 there are three different methods for three different states:

```
1.  if(mail_state.equalsIgnoreCase("SENT"))
2.  {
3.    ProcessSentMail(config,mail);
4.  }else if(mail_state.equalsIgnoreCase("AWAIT_REPLY"))
5.  {
6.    ProcessAwaitReplyMail(config,mail);
7.  }else if(mail_state.equalsIgnoreCase("REPLIED"))
8.  {
9.    ProcessRepliedMail(config, mail);
10. }else{
11. }
```

Listing 12. According to mail_state ProcessMail calls the different submethods: ProcessSentMail, ProcessAwaitReplyMail, ProcessRepliedMail

- Method `ProcessSentMail` (line 3)

It has two input variables: `config` and `mail`. Variable `config` is ConfigFileHandler and by this object all information in the file configuration.conf can be read. Variable `mail` is an `ArrayList`, which contains all dates of the mail with the state SENT in the database.

One element in `ArrrayList` is `mail_response`. If the value of `mail_response` is no, this means that this mail is a notification and does not need a response. Then the `mail_state` of this mail in the database will be set as FINISHED. If the value is yes, this means that this mail waits for a response. Then the `mail_state` in the database will be set as AWAIT_REPLY. Finally the method `ProcessAwaitReplyMail` will be called to process this mail.

- Method ProcessAwaitReplyMail (line 6)

It has two input variables: `config` and `mail`. Variable `config` is described above. Variable `mail` is an `ArrayList`, which contains the dates of the mail with the state AWAIT_REPLY in the database.

One element in `ArrayList` is `mail_id`. According to this id method `ProcessAwaitReplyMail` looks for a response in the table maildb.mailbox, which is a response for the input mail. If such response exists in the database and it received in the time limit, then the `mail_state` in the database will be set as REPLIED. If this response is timeout, then the `mail_state` will be set as TIMEOUT.

- Method ProcessRepliedMail (line 9)

It has also two input variables: `config` and `mail`. Variable mail is an ArrayList, which contains the dates of the mail with the state REPLIED in the database.

The main function of this method is to check the validation for a response. This function is implemented by a public and static method `isValidResponse` in Listing 13. This method

use `javax.xml.validation` API. The `javax.xml.validation` API uses three classes to validate documents: `SchemaFactory`, `Schema`, and `Validator`. `SchemaFactory` reads the schema document from which it creates a `Schema` object (line 11). The `Schema` object creates a `Validator` object (line 12). Finally, the `Validator` object validates an XML document represented as a `Source` (line 14). To check the Validation these two strings must be transformed into two well-defined XML documents (line 4 and line 5). If a `Source` is invalid, the `validate( )` method throws a `Exception` and returns "false" (line 17-20). Otherwise, it returns "true" (line 16).

```
1.   public static boolean isValidResponse(ConfigFileHandler config,
2.                                        String xmlStr, String xsdStr){
3.  boolean isValidResponse = false;
4.  xmlStr = config.getXmlHeader()+xmlStr+config.getXmlFoot();
5.  xsdStr = config.getXsdHeader()+xsdStr + config.getXsdFoot();
6.  SchemaFactory factory = SchemaFactory.newInstance(SCHEMA_LANGUAGE);
7.  Reader xmlReader = new BufferedReader(new StringReader(xmlStr));
8.  Reader xsdReader = new BufferedReader(new StringReader(xsdStr));
9.  try{
10.     Source xsdSource = new StreamSource(xsdReader);
11.     Schema schema = factory.newSchema(xsdSource);
12.     Validator validator = schema.newValidator();
13.     Source xmlSource = new StreamSource(xmlReader);
14.     validator.validate(xmlSource);
15.     isValidResponse = true;
16.     return isValidResponse;
17. }catch(Exception ex)
18. {
19.    return isValidResponse;
20. }
21. }
```

Listing 13. Method isValidResponse()

5.3.7 Register Service

This subservice's main purpose is to add every new user as registered user in Mail Service. It is placed in following Java package:

`de.unistuttgart.iaas.interaction.mail`

Main function is implemented by the public and static method `register`. Each recipient in a request must be a registered user in Mail Service. The table maildb.mailregister saves the address and username of each registered user. If the recipient in a request is invalid user for Mail Service, then Mail Service builds an internal request. This request is to send a register mail from the administrator's mailbox to this recipient, and ask the recipient to reply this register mail. The content of this reply is not required. The most important is the subject containing a word "register". `ReceiveMail` service puts the mail with `mail_class=REGISTER` and `mail_state=NEW` in the database table maildb.mailbox.

Register service chooses these mails, extracts the address and username from these mails, and adds the dates in the database table maildb.mailregister.

5.3.8 AutoRun.java

Send a mail, Receive mails, register a new user, and process mails, these functions should run repeatedly after the server starts. Java class `AutoRun` is responsible for this. Listing 14 shows the Java code in the class `AutoRun` to schedule `ReceiveMail` to run repeatedly after the server starts.

`AutoRun` extends the `javax.servlet.http.HttpServlet` class of the `javax.servlet.http` package. The `HttpServlet` class is a subclass of `GenericServlet`, an implementation of the `Servlet` interface. There are three methods `init()`, `service()`, and `destroy()`. They are implemented by every servlet and are invoked at specific times by the server.

Class `AutoRun` uses two classes: `java.util.Timer` and `Java.util.TimerTask` [25]. A `TimerTask` object represents a task. The task can be called by the `Timer` any number of times at regular intervals. Threads can be assigned for different tasks. `Timer` class is capable to schedule the threads by the method `Timer.schedule(TimerTask task, long delay, long period)`(line 3). Method `Timer.schedule()` schedules the specified task for repeated fixed-delay execution, beginning after the specified delay. In Listing 14 ReceiveMail runs repeatedly every two minutes (`RECEIVE_PERIOD= 2*60*1000`) once the server starts.

The `TimerTask` is an abstract class and implements Runnable interface. So, as Runnable interface is implemented, `run()` method must be overridden when `TimerTask` is extended (line 4-line 10). Listing 14 illustrates the usage of `Timer` and `TimerTask` classes.

The other three functions are similar to ReceiveMail.

```
1.  Timer timer = new Timer(true);
2.  /**TimerTask for ReceiveMail()*/
3.  timer.schedule(new TimerTask(){
4.    public void run()
5.    {
6.      try{
7.            ReceiveMail.ReceiveMail(config);
8.         }catch(Exception e){}
9.    }
10.},0,RECEIVE_PERIOD);
```

Listing 14. ReceiveMail runs repeatedly

5.4 Skype Service

Skype Service consists of multiple subservices. Figure 27 shows the components diagram from Skype Service. Among the components SkypeGetRequest, SendMessage, Register, and ProcessMessage have been implemented to send a notification message. SkypeGetResponse and ReceiveMessage have not been implemented.

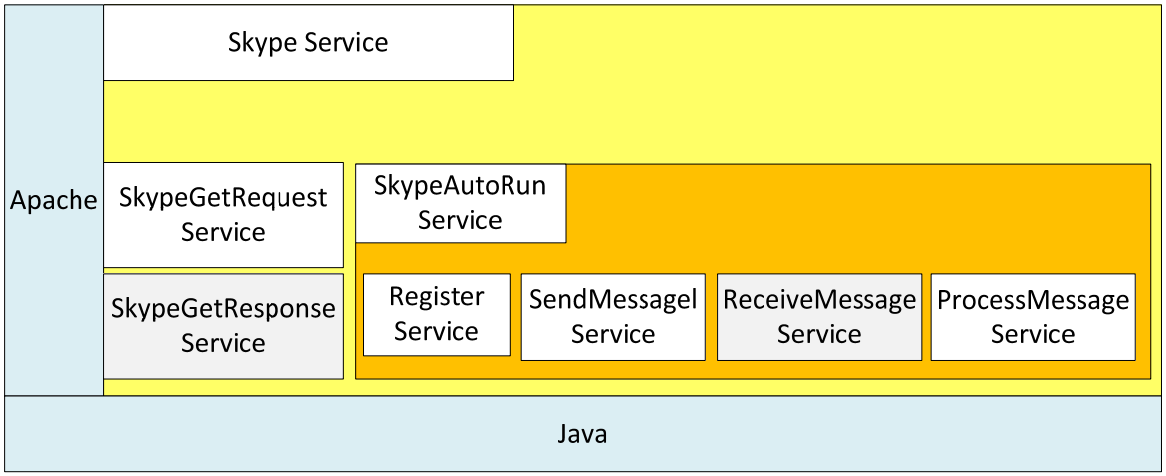

Figure 27. Components diagram for Skype Service

5.4.1 Skype API for Java

Skype Service in this thesis uses skype-java-api-1.1.jar. This Skype API [29] is used to integrate Skype services and functionalities inside a Java program. Skype API provides access to much of the functionality of the Skype client such as contact lists, making and receiving calls or conference calls, application to application messaging, moods, status, and chat. To use this Skype API Skype must be running, otherwise an error will be sent. Text message and files exchanging are used in Skype Service.

5.4.2 SkypeGetRequest Service

This subservice's main purpose is to get a request from a business process, check the validity and type of the request message. Because at the same time Skype can send either a text message or a file, so the type of the message must be indicated. Then according to the type of the request message `SkypeGetRequest` saves the request in the database table `skydb.senttextmessage` or `skydb.sentfilemessage`. Finally it returns a message to the process. It is placed in following Java package:

```
de.unistuttgart.iaas.interaction.skype
```

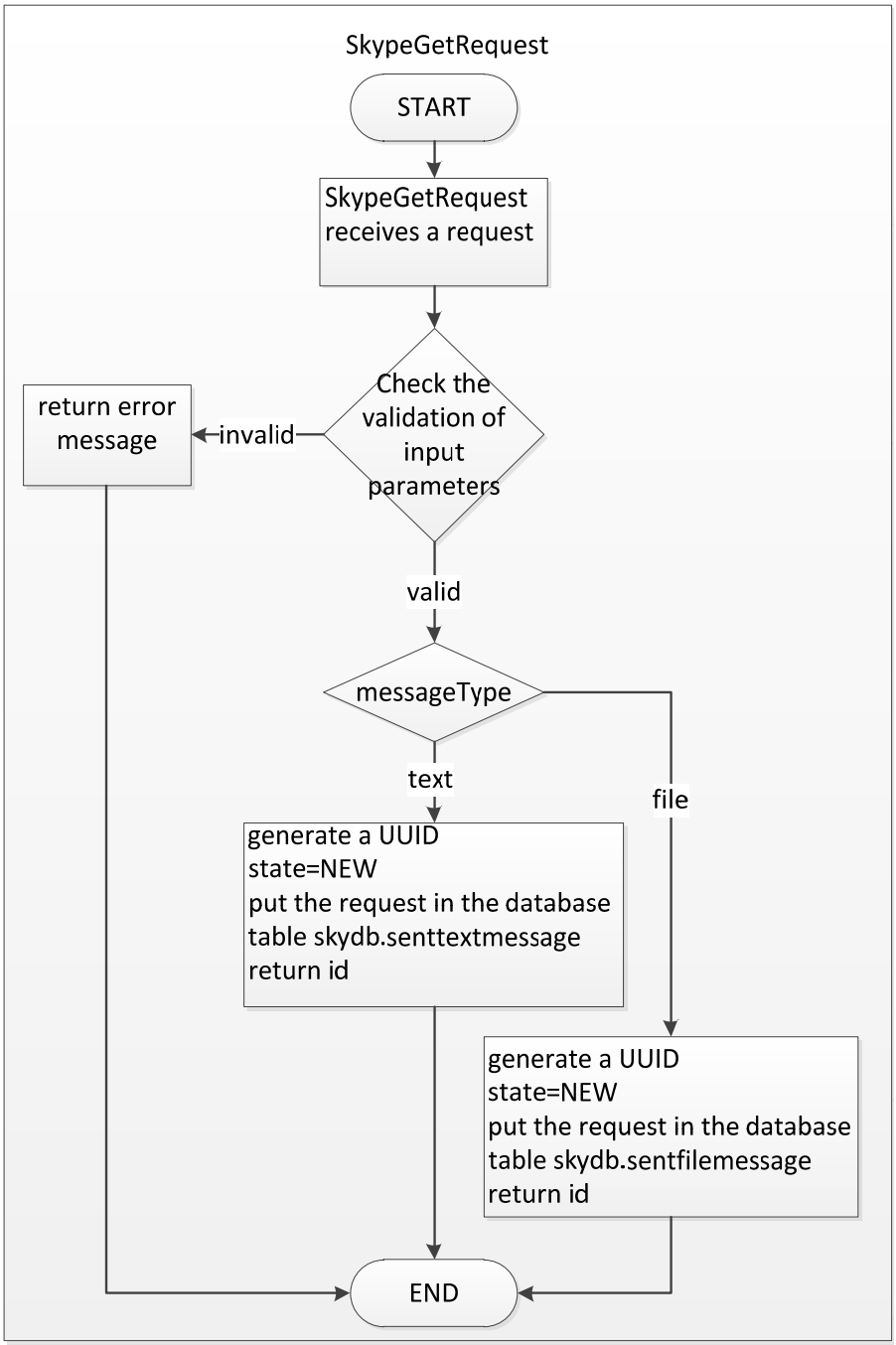

Figure 28. Flow Diagram for SkypeGetRequest

Main function is implemented by the public and static method `skypeGetRequest`. Figure 28 shows the function of SkypeGetRequest service step by step. Input parameter `messageType` indicates where a request should be saved, in skydb.senttextmesage or skydb.sentfilemessage. Parameter `messageType` has two values, “text” or “file”.

5.4.3 SendMessage Service

This subservice selects all requests with state=NEW in the database table skydb.senttextmessage and according to a selected request SendMessage service sends a message by Skype. It is placed in following Java package:

de.unistuttgart.iaas.interaction.skype

Main function is implemented by the public and static method `sendMessage`, shown in Listing 15 . This method selects all requests with `state`= NEW in the database table `skydb.senttextmessage` (line 1-8), and calls Skype API to send a text message (line 27-28). Finally database `skydb` will be updated.

```
1. Statement statementText = conn.createStatement();
2.  ResultSet rsText = statementText.executeQuery("select id from
3. skydb."+config.getSkydb_table_sentTextMessage()+" where state='NEW'");
4.  ArrayList list_new   = new   ArrayList();
5.  while(rsText.next()){
6.      list_new.add(rsText.getString(1));
7.      i++;}
8.  statementText.close();
9.  for(int j=0;j<i;j++){
10.     String lv = (String) list_new.get(j);
11.     Statement sendStatement1 = conn.createStatement();
12.     ResultSet sendResultset = sendStatement1.executeQuery("select * from
13.      skydb."+config.getSkydb_table_sentTextMessage()+
14.      " where id='"+lv+"'");
15.     ArrayList skype_message = new ArrayList();
16.     if(sendResultset.next()){
17.         for(int k=1; k<=9;k++){
18.             skype_message.add(sendResultset.getString(k));}
19.     }
20.     skype_id = (String)skype_message.get(0);
21.     skype_to = (String)skype_message.get(1);
22.     skype_textmessage = (String)skype_message.get(3)+"\n";
23.     skype_reply = (String)skype_message.get(5);
24.     skype_responseTemplate = (String)skype_message.get(6);
25.     Chat chat = Skype.getContactList().getFriend(skype_to).chat();
26. chat.send(skype_textmessage);  …
27. update.executeUpdate("update skydb."
28.    +config.getSkydb_table_sentTextMessage()+" set sentDate='"
29.    +sentTime+"', state='SENT' where id='" +skype_id+"'");
30. update.close();
31. }
```

Listing 15. Method sendMessage()

5.4.4 ProcessMessage Service

This subservice's main purpose is to process each request in the table skydb.senttextmessage and skydb.sentfilemessage. It is placed in following Java package:

`de.unistuttgart.iaas.interaction.skype`

Main function is implemented by the public and static method `processMessage`. This method processes the requests with “SENT”, “AWAIT_REPLY” or “REPLIED” state. Each request has a status indicator `state` in the database and the content of this indicator changes at some time. For example, when `SkypeGetRequest` service gets a request from an external process and stores this request in the database, at this time `state` is set as NEW. `SendMessage` service chooses the requests with NEW state in the table skydb.senttextmessage and skydb.sentfilemessage and processes these requests. If these requests are sent successfully, the states of them are set as SENT. If an error occurs during the sending, then the states of this mail is set as FAILED. There are three different methods for three different states, shown in Listing 16.

```
1.  if(skype_state.equalsIgnoreCase("SENT"))
2.  {
3.    ProcessSentMessage(config,message);
4.  }else if(skype_state.equalsIgnoreCase("AWAIT_REPLY"))
5.  {
6.    ProcessAwaitReplyMessage(config,message);
7.  }else if(skype_state.equalsIgnoreCase("REPLIED"))
8.  {
9.    ProcessRepliedMessage(config, message);
10. }else{
11. }
```

Listing 16. According to skype_state ProcessMessage calls the different submethods: ProcessSentMessage, ProcessAwaitReplyMessage, ProcessRepliedMessage

Only `ProcessSentMessage` method is completely implemented, shown in Listing 17. If the value of `reply` is no, that means this request do not need a response. `ProcessSentMessage` updates the database, sets the value of `state` as FINISHED. If the value of `reply` is yes, that means this request has been sent and is waiting for a response. `ProcessSentMessage` sets the value of `state` as AWAIT_REPLY, and then calls method `ProcessAwaitReplyMessage()`.

```
1.   if(reply.equalsIgnoreCase("no"))
2.   {
3.       update.executeUpdate("update
4.       skydb."+config.getSkydb_table_sentTextMessage()
5.       +" set state='FINISHED' where id='"+id+"'");
6.       System.out.println("id "+ id+"needs no response, set state=finished.");
7.   }else if(reply.equalsIgnoreCase("yes")){
8.       System.out.println("id "+ id +" needs response,
9.                           set state=await_reply.");
10.      update.executeUpdate("update
11.      skydb."+config.getSkydb_table_sentTextMessage()
12.      +" set state='AWAIT_REPLY' where id='"+id+"'");
13.      list_SentMessage.set(8, "AWAIT_REPLY");
14.      list_AwaitReplyMessage= list_SentMessage;
15.      ProcessAwaitReplyMessage(config, list_AwaitReplyMessage);
16.  }else
17.  {
18.      System.out.println("Error in ProcessSentMail( )");
19.  }
```

Listing 17. Method ProcessSentMessage()

5.4.5 Register Service

This subservice's main purpose is to add every new user as registered user in Skype Service. It is placed in following Java package:

`de.unistuttgart.iaas.interaction.skype`

Main function is implemented by the public and static method `register`. Each recipient in a request must be a registered user in Skype Service. The table skydb.register saves Skype id and username of each registered user. If the recipient in a request is invalid user for Skype Service, then Skype Service builds an error message and returns to the process. A new user must add administrator of Skype Service in his contract list of Skype. Skype sends automatically a request to the administrator. Listing 18 lists the main function of method `register()`. Register service gets all users which are waiting for authorization (line 3). If the Skype id of a user among them exists in the database table skydb.register, then this user is not new for Skype Service. Otherwise, Register service extracts his id and full name (line 7 and line 13) and saves the two values in the database table skydb.register. Then property `Authorized` is set as true and property `Blocked` is set as false (line 21-22). Finally Skype Service generates an internal request and will send a confirmation message to this new user (line 23).

```
1.  Connection conn = dao.connDataBase(config);
2.  Friend[] friends =
3.  Skype.getContactList().getAllUserWaitingForAuthorization();
4.  int friendNum = friends.length;
5.  for(int i = 1; i<=friendNum;i++)
6.  {
7.     String skype_username = friends[i-1].getId();
8.     Statement statement = conn.createStatement();
9.     ResultSet rs = statement.executeQuery("select skype_username from
10.    skydb."+config.getSkydb_table_register()+" where skype_username='"+
11.    skype_username +"'");
12.    if(!rs.next()){
13.        String name = friends[i-1].getFullName();
14.        PreparedStatement preStatementRegister=
15.         conn.prepareStatement("insert into skydb.register
16.         (skype_username,name) values(?,?)");
17.         preStatementRegister.setString(1,skype_username);
18.         preStatementRegister.setString(2,name);
19.         preStatementRegister.executeUpdate();
20.         preStatementRegister.close();
21.         friends[i-1].setAuthorized(true);
22.         friends[i-1].setBlocked(false);
23.         SkypeGetRequest.skypeGetRequest("text", "Your register is
24.          successful!", skype_username, "no", "", "");
25.     }
26. }
27.
```

Listing 18. Method register() in Skype Service

5.4.6 SkypeAutoRun.java

Similar to Mail Service, `SendMessage`, `Register`, `ProcessMessage` run repeatedly after the server starts. Java class `SkypeAutoRun` is responsible for this and it has the same structure with Java class `AutoRun` in Mail Service, see in Chapter 5.3.8

5.5 Deployment as Web Services

In this work Web Service is implemented by a bottom up approach. In short, the steps are as follows:

1) Create a project in the Eclipse workspace of type “Dynamic Web Project”, which will host Web Service.

2) Write the Java code that implements Web service functionality.

3) Use Eclipse to automatically generate the components (WSDL etc.) that will transform the Java code into a Web Service, and the ask Eclipse to run that Web Service.

In Chapter 5.3 and 5.4 Mail Service and Skype Service functionalities have been implemented in Java program. In this work five Java classes `GetRequest`, `AutoRun`,`GetResponse` in Mail Service and `SkypeGetRequest`, `SkypeAutoRun` in Skype

Service are deployed as Web Services. Eclipse generates automatically WSDL document for each Java class. Another important document is web.xml file in the path /WEB-INF/web.xml.

5.5.1 web.xml

The web.xml file is the Web Application Deployment Descriptor [26]. This is an XML file describing the servlets and other components that make up web application, along with any initial parameters and container-managed security constraints the server can start up and run with the specified servlets. The following Table 19 is the description of servlet elements in web.xml:

Element	Required/ Optional	Description
<servlet-name>	Required	Defines the canonical name of the servlet, used to reference the servlet definition elsewhere in the deployment descriptor.
<servlet-class>	Required	The fully-qualified class name of the servlet.
<init-param>	Optional	Contains a name/value pair as an initialization attribute of the servlet.
<load-on-startup>	Optional	The optional content of this element must be a positive integer indicating the order in which the servlet should be loaded. Lower integers are loaded before higher integers.

Table 19. Elements defintion in the file web.xml

To load a servlet on startup of the server, the following configuration in **Fehler! Verweisquelle konnte nicht gefunden werden.** must be added into web.xml. Servlet `AutoRun` is referred to the class `de.unistuttgart.iaas.interaction.mail.services.AutoRun`. It has one initial parameter with the name "`ConfigFile`" and the value "`C:\CommunicationService\configuration.conf`". The content of <load-on-startup> is 1. That means once the server starts up, this servlet will be loaded.

```
<servlet>
    <servlet-name>AutoRun</servlet-name>
     <servlet-class>
         de.unistuttgart.iaas.interaction.mail.services.AutoRun
    </servlet-class>
     <init-param>
        <param-name>ConfigFile</param-name>
     <param-value>
             C:\CommunicationService\configuration.conf
         </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet>
    <servlet-name>SkypeAutoRun</servlet-name>
    <servlet-class>
         de.unistuttgart.iaas.interaction.skype.services.SkypeAutoRun
    </servlet-class>
     <init-param>
         <param-name>ConfigFile</param-name>
         <param-value>
             C:\CommunicationService\configuration.conf</param-value>
     </init-param>
     <load-on-startup>1</load-on-startup>
</servlet>
```

Listing 19. Configuration in web.xml

To get the value of initial parameter a piece of code in Listing 20 must be added in the class `AutoRun.java`

```
1.  public class AutoRun extends HttpServlet
2.  {…
3.  public void init()
4.   {
5.   final String config = getServletConfig().getInitParameter("ConfigFile");
6.   …
```

Listing 20. The way to use an initial parameter in the file web.xml

5.5.2 WSDLs

Class `GetRequest` is deployed as a Web Service. A generated WSDL document GetRequest.wsdl describes GetRequest Web Service.

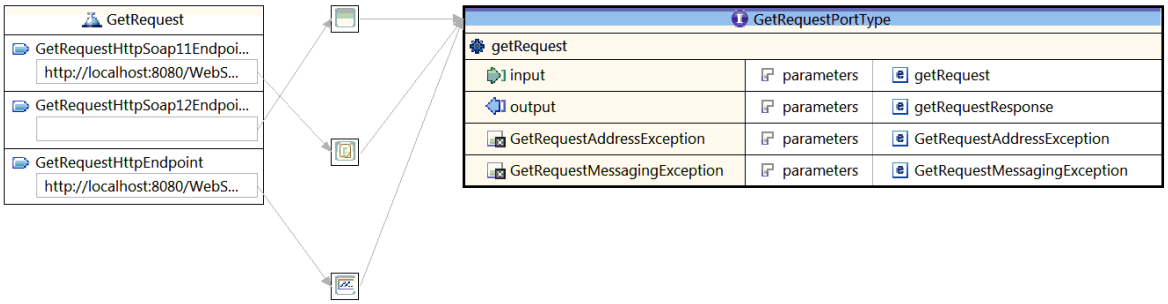

Figure 29. Graphical representation of GetRequest.wsdl

Figure 29, Figure 32, Figure 35 show three graphical representations of GetRequest.wsdl, GetResponse.wsdl and SkypeGetRequest.wsdl. Because this WSDL documents are generated automatically by Eclipse, three endpoints are defined for this services:

GetRequestHttpSoap11Endpoint

GetRequestHttpSoap12Endpoint

GetRequestHttpEndpoint

Each endpoint is related to a binding. The <portType> element in GetRequest.wsdl defines that, in GetRequest Web Service operation getRequest can be performed and for this operation message getRequest is an input parameter and message getRequestResponse is an output parameter. Figure 30 and Figure 31 show the definitions for input parameter and output parameter.

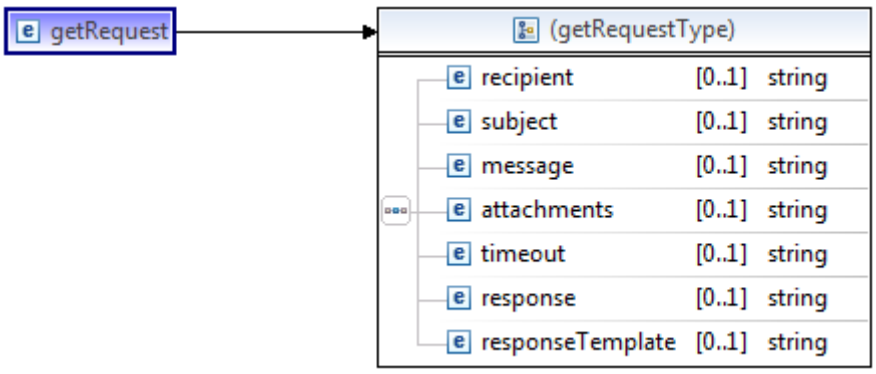

Figure 30. Definition of input parameter

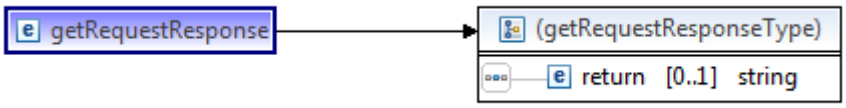

Figure 31. Definition of output parameter

The <portType> element in GetResponse.wsdl defines that, in GetResponse Web Service operation getResponse can be performed and for this operation message getResponse is an input parameter and message getResponseResponse is an output parameter. Figure 33 and Figure 34 show the definitions for input parameter and output parameter.

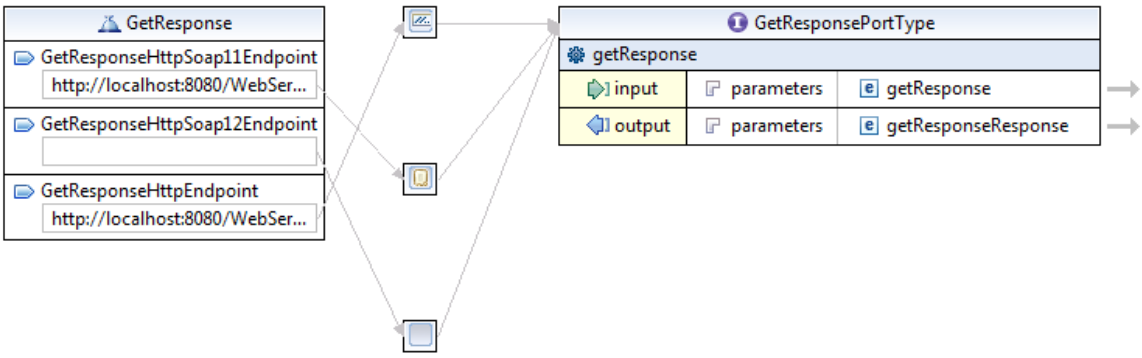

Figure 32. Graphical representation of GetResponse.wsdl

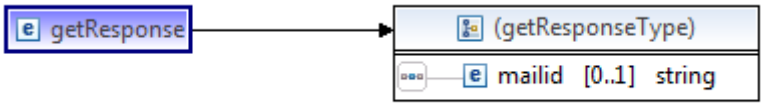

Figure 33. Definition of input parameter for GetResponse

Figure 34. Definition of output parameter for GetResponse

The <portType> element in SkypeGetRequest.wsdl defines that, in SkypeGetRequest Web Service operation skypeGetRequest can be performed and for this operation message skypeGetRequest is an input parameter and message getRequestResponse is

an output parameter. Figure 36 and Figure 37 show the definitions for input parameter and output parameter.

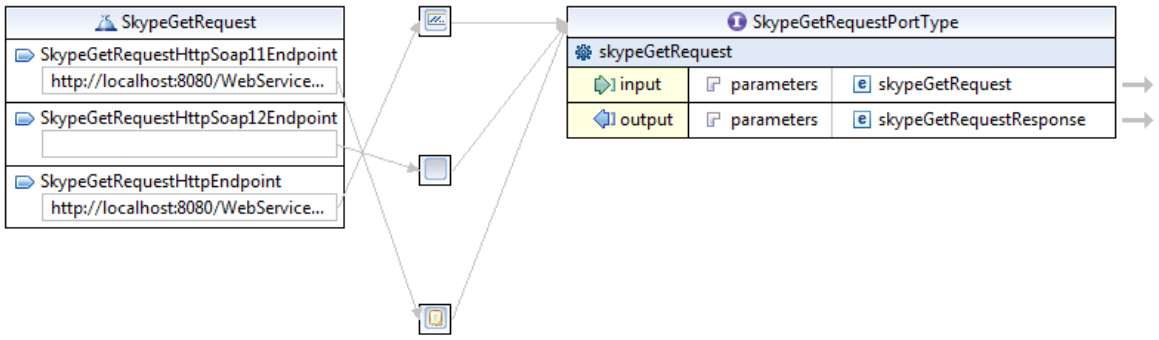

Figure 35. Graphical representation of SkypeGetRequest.wsdl

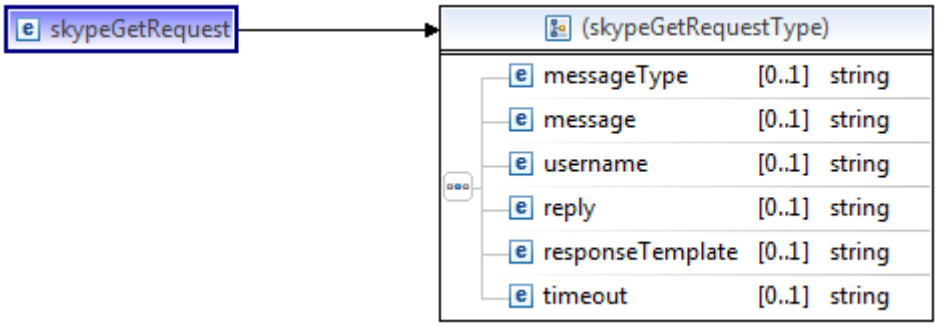

Figure 36. Definition of input parameters for SkypeGetRequest

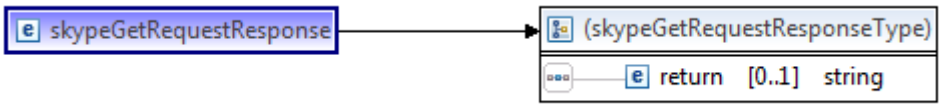

Figure 37. Definition of output parameter for SkypeGetRequest

In Chapter 6 test tool SoapUI and BPEL process use the link of the GetRequest WSDL document to test Mail Servcie.

6 Test Cases

This chapter presents test cases that were executed using Communication Services.

6.1 Test Tool: SoapUI

SoapUI [30] is a tool for Testing Web Services. Its functionality covers Web Service inspection, invoking, development, simulation and mocking, and functional testing.

In order to begin testing a service a WSDL file describing it is required. In this Chapter SoapUI uses GetRequest.wsdl, GetResponse.wsdl and SkypeGetRequest.wsdl. SoapUI can directly generate the SOAP messages from WSDL. The parameters can be manually filled in the SOAP message. Then the tests can be performed. Figure 38 illustrates the SOAP message for GetRequest.wsdl in Mail Service.

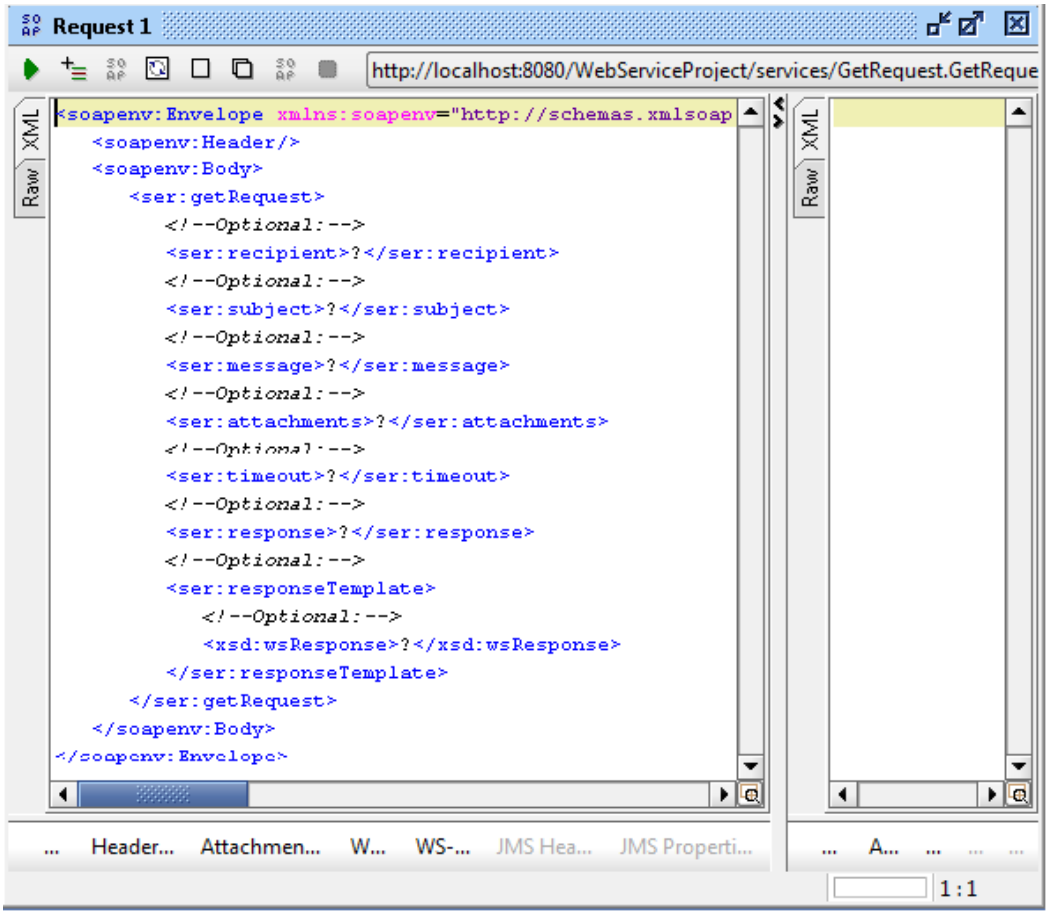

Figure 38. SoapUI generates a SOAP message

6.2 Test Tool: BPEL Process

Business Process Execution Language (BPEL) [31] is an OASIS (Organization for the Advancement of Structured Information Standards) standard executable language for specifying actions within business processes with Web Services.

Processes in BPEL export and import information by using Web Service interfaces exclusively. Figure 39 shows a simple BPEL process. In this process there are five activities: receiveInput, Assign, Invoke, Assign1, replyOutput. The combination of a receiveInput activity and a resplyOutput activity forms a request-response operation on the WSDL port type for the process. Assign activity provides a method for data manipulation, such as copying the content of one variable to another. Invoke activity invokes a synchronous Web Service or initiates an asynchronous Web Service. In the following test case, a BPEL process will be created to invoke Mail Service and Skype Service.

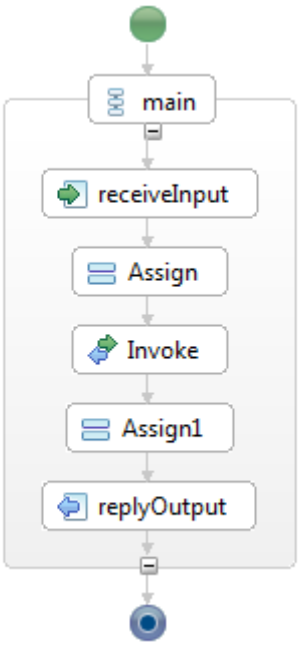

Figure 39. BPEL process

6.3 Test Cases

6.3.1 Test Case1—Mail Service Gets Invalid Request Message

There are several following reasons that a request message is invalid for Mail Service:

1) `recipient`, `subject`, `message`, `response` can be null.

2) `timeout` has false format.

3) `responseTemplate` can be null, but `response` is yes.

Listing 21 is an example for invalid request message. In this example recipient parameter is null.

```
<soapenv:Envelope xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
   xmlns:ser="http://services.mail.interaction.iaas.unistuttgart.de"
   xmlns:xsd="http://mail.interaction.iaas.unistuttgart.de/xsd">
      <soapenv:Header/>
      <soapenv:Body>
         <ser:getRequest>
            <!--Optional:-->
            <ser:recipient></ser:recipient>
            <!--Optional:-->
            <ser:subject>Notification from mail service</ser:subject>
            <!--Optional:-->
            <ser:message>
                We will maintain our system from 1:00 to 4:00. Mail Service
                is suspended during this period. Please understand it.
            </ser:message>
            <!--Optional:-->
            <ser:attachments></ser:attachments>
            <!--Optional:-->
            <ser:timeout></ser:timeout>
            <!--Optional:-->
            <ser:response>no</ser:response>
            <!--Optional:-->
            <ser:responseTemplate></ser:responseTemplate>
         </ser:getRequest>
      </soapenv:Body>
</soapenv:Envelope>
```

Listing 21. An example for invalid request message for Mail Service

Before GetRequest service accepts a SOAP message from SoapUI, it checks the validation of each parameter in this SOAP message. Once a parameter is invalid, the service will be stopped and a string will be returned. This request will not be saved in the database. The return message indicates that this request is rejected. The reason for rejecting this request will be saved in the log file and the reason is invalid recipient parameter. Listing 22 shows this message in SOAP message format.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
    <ns:getRequestResponse
    xmlns:ns="http://services.mail.interaction.iaas.unistuttgart.de">

        <ns:return>ERROR</ns:return>

    </ns:getRequestResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Listing 22. The response message for the request in Listing 21

6.3.2 Test Case2—Mail Service Register

In this test case, the mail address commserv.user@yahoo.de from an unregistered user in mail service is the recipient in the request message. That means, this address does not exist in the table maildb.mailregister.

```
<ser:getRequest>
  <ser:recipient>commserv.user@yahoo.de</ser:recipient>
  <ser:subject>Notification from mail service</ser:subject>
  <ser:message>
        We will maintain our system from 1:00 to 4:00. Mail service
        is suspended during this period. Please understand it.
  </ser:message>
  <ser:attachments></ser:attachments>
  <ser:timeout></ser:timeout>
  <ser:response>no</ser:response>
  <ser:responseTemplate></ser:responseTemplate>
</ser:getRequest>
```

Listing 23. The request message with an unregistered recipient

If a request message is passed the validation check, as next step GetRequest service must check, whether the recipient exists in the database or not. If the recipient is an unregistered user in mail service, the following SOAP message in Listing 24 are generated and returned. In this SOAP message a string “ERROR” are returned to SoapUI.

```
<soapenv:Envelop
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
      <ns:getRequestResponse
          xmlns:ns="http://services.mail.interaction.iaas.unistuttgart.de">

         <ns:return>ERROR</ns:return>

      </ns:getRequestResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 24. The response message for the request with an unregistered user in Listing 23

At the same time GetRequest service generates another new register request and puts the register request in the database table maildb.sentmails. Now the request is ready to be sent. Table 20 shows the complete information of a register request. SendMail service which runs periodically processes this request. It sends a mail to commserv.user@yahoo.de, asks the user to register in mail service and updates mail_state from “NEW” to “SENT”.

Columns in maildb.sentmails	Value
mail_id	eb3b8340-9f33-11e1-8d32-0021869473a9
mail_to	commserv.user@yahoo.de
mail_from	CommServ.ad@gmx.de
mail_subject	Please register for mail service!
mail_message	you are invalid user. Please answer this mail for register.
mail_sentDat	2012-05-16 10:50:38
mail_response	
mail_response_pattern	
mail_timeout	NULL
mail_class	Request
mail_state	NEW

Table 20. An example of a register request

RegisterMail service runs also periodically and for it the subject of a reply is important. The key to confirm a Reply as a register mail is, whether the subject contains “register” this word or contains a word which is very similar to “register”, for example, “regiter” or “resgiter”. After the user receives this mail, the user can directly send a reply without any modification, or he can also send a new mail to the address CommServ.ad@gmx.de and the subject of the mail contains “register” or a word which is similar to “register”. Table 21 is an example. The user directly replies the register request, and then ReceiveMail service gets this mail from the administrator’s account, puts it in the database table maildb.mailbox and set mail_state as “NEW”.

Column in maildb.mailbox	Value
mail_id	98c97c50-9f35-11e1-8d32-0021869473a9
mail_replyid	
mail_receivedDate	2012-05-16 11:00:36
mail_from	commserv.user@yahoo.de
mail_from_username	CommServ User
mail_to	CommServ.ad@gmx.de

mail_subject	Re: Please register for mail service! Mail-ID:eb3b8340-9f33-11e1-8d32-0021869473a9
mail_message	_____ Von: "CommServ.ad@gmx.de" <CommServ.ad@gmx.de> An: commserv.user@yahoo.de Gesendet: 1:00 Donnerstag, 1.Januar 1970 Betreff: Please register for mail service! Mail-ID:eb3b8340-9f33-11e1-8d32-0021869473a9 you are invalid user. Please answer this mail for register.
mail_class	REGISTER
mail_state	NEW

Table 21. An example of a reply for a register request

When Register service runs automatically and periodically, it processes the above reply. Register service extracts mail_from and mail_from_username, adds them into the database table maildb.mailregister and then set mail_state in the above table as "FINISHED". Finally, Register service generates a confirmation request to tell the user that his register is successful. After this confirmation mail is sent, the registration process is completely.

6.3.3 Test Case3—Mail Service Gets a Notification Request

6.3.3.1 A Text-Notification Request

A Notification Request means that the recipient does not reply the mail which is sent according to this request from Mail Service. The important key in a request message is variable `response` which must be set as "no". Listing 25 is a notification request message in SoapUI.

```
<ser:getRequest>
    <ser:recipient>commserv.user@yahoo.de</ser:recipient>
     <ser:subject>Notification from mail service</ser:subject>
     <ser:message>
        We will maintain our system from 1:00 to 4:00. Mail service
        is suspended during this period. Please understand it.
     </ser:message>
     <ser:attachments></ser:attachments>
     <ser:timeout></ser:timeout>
     <ser:response>no</ser:response>
     <ser:responseTemplate></ser:responseTemplate>
</ser:getRequest>
```

Listing 25. An example of a notification request

GetRequest Service receives this request, generates a UUID for it, and puts every data into the database table `maildb.sentmails`. In this table two columns `mail_class` and `mail_state` for this request are set as “REQEUST” and “NEW”. And the UUID (`<ns:return>16405970-a0bb-11e1-b415-0021869473a9</ns:return>`) as the return value is returned to SoapUI. SendMail service which is running in the background finds this new request, for it sends a mail to commserv.user@yahoo.de, and the value of `mail_state` is change to “FINISHED”.

Figure 40 illustrates the notification mail from Mail Service, which the user commserv.user@yahoo.de has received.

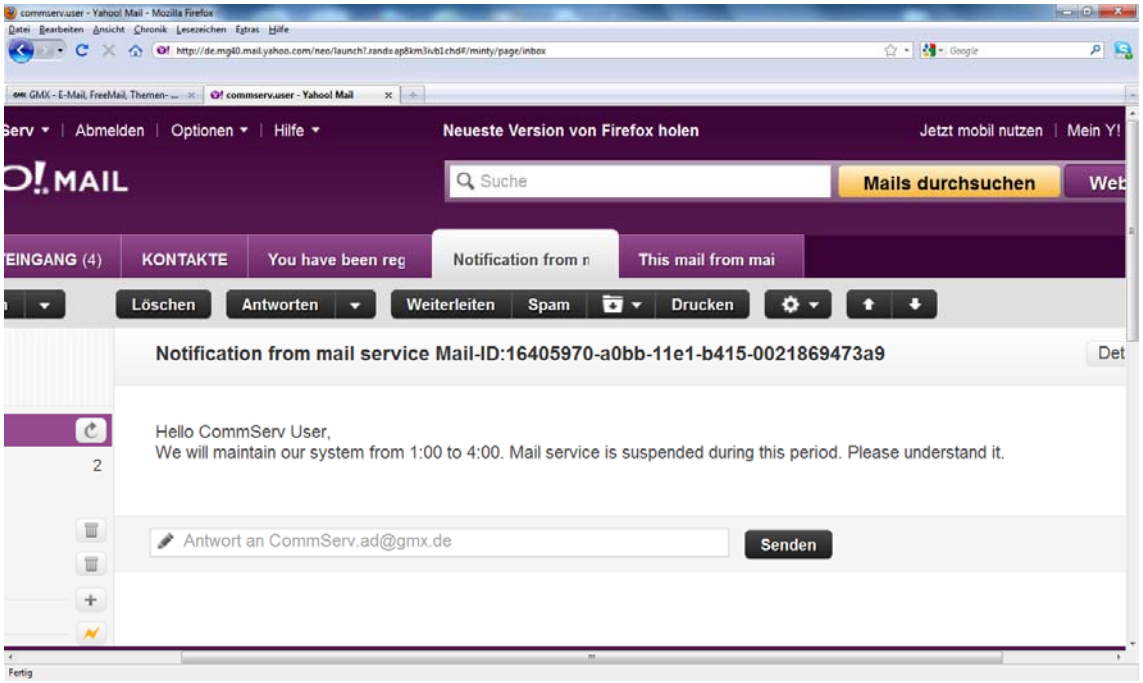

Figure 40. The user commserv.user@yahoo.de has recieved a notification from Mail Service.

6.3.3.2 A Request with Attachment

Attachment is an important component of a mail. A mail can contain a piece of text and some attachment files. A request message containing an attachment which is a piece of data is shown in Listing 26. The content of this attachment is added into element<!CDATA[[ ]]>. This element is marked for the parser to interpret as only character data, not markup.

```
<ser:getRequest>
    <ser:recipient>commserv.user@yahoo.de</ser:recipient>
   <ser:subject>This mail from mail service.</ser:subject>
    <ser:message>We send a report to you.</ser:message>
   <ser:attachments> <!CDATA[[data 6 45 0000000016 00000 n 0000001413 00000 n
   0000001490 00000 n 0000001668 00000 n 0000002028 00000 n 0000002517 00000 n
   0000003005 00000 n 0000003040 00000 n 0000003085 00000 n 0000003130 00000 n
   0000003175 00000 n 0000004950 00000 n 0000006706 00000 n 0000008357 00000 n
   0000010149 00000 n 0000011902 00000 n 0000012110 00000 n 0000012312 00000 n
   0000014429 00000 n 0000016349 00000 n 0000018546 00000 n 0000021239 00000 n
   0000021939 00000 n 0000022592 00000 n 0000023807 00000 n 0000025041 00000 n
   0000025660 00000 n 0000026218 00000 n 0000027412 00000 n 0000028705 00000 n
   0000029959 00000 n 0000031201 00000 n 0000031850 00000 n 0000032589 00000 n
   0000033844 00000 n 0000035139 00000 n 0000036414 00000 n 0000036642 00000 n
   0000036864 00000 n 0000037098 00000 n 0000037337 00000 n 0000037414 00000 n
   0000037550 00000 n 0000037667 00000 n 0000001196 00000]]>
   </ser:attachments>
   <ser:timeout></ser:timeout>
    <ser:response>no</ser:response>
    <ser:responseTemplate></ser:responseTemplate>
</ser:getRequest>
```

Listing 26. An example of a request containing an attachment file

GetRequest service loads this piece of data as a string in an `InputStream`, and generates a filename “fc9c8d50-a451-11e1-b793-0021869473a9.txt” for this `InputStream`, then puts the `InputStream` into the field “content” of the table `maildb.mailattachments`. Figure 41 shows contents of this table at this time.

id	mail_id	filename	content
1	fc9c8d50-a451-11e1-b793-0021869473a9	fc9c8d50-a451-11e1-b793-0021869473a9.txt	BLOB
2	c1319070-a452-11e1-9c94-0021869473a9	c1319070-a452-11e1-9c94-0021869473a9.txt	BLOB

Figure 41. A file is added into the table maildb.mailattachments.

After that, SendMail service reloads the data from the database into a new file with the name “fc9c8d50-a451-11e1-b793-0021869473a9.txt” and this file is saved in local disk. The specific path is defined in configuration.conf. Then SendMail service adds the attachment into a new `messageBodyPart` of `message`. Finally, SendMail service sends a mail containing an attachment fc9c8d50-a451-11e1-b793-0021869473a9.txt to the recipient.

6.3.4 Test Case4—Mail Service Gets a Request Needing a Response

For Mail Service a request can contain one or a few questions, which the recipient has to answer them in a response. These two cases are shown in the following tests.

6.3.4.1 Simple Question in a Request

GetRequest service receives the following request from SoapUI. Listing 27 shows a request message which contains one question. This request needs a response in 10 hours (line 8) and this response must confirm to a template (line 10-17). In this template only an answer for “Question1” is defined. This answer should be a string with the values “yes” or “no”. In element <!CDATA[[ ]]> character “<” is represented by “<”, character “>” is represented by “>”, character “&” is represented by “&” as syntax rule. In fact, the template looks like this:

<![CDATA[[<wsResponse><name>Question1</name><type>string</type><enumeration >yes|no</enumeration><min></min><max></max><wsResponse>]]>

```
<ser:getRequest>
   <ser:recipient>commserv.user@yahoo.de</ser:recipient>
   <ser:subject>
      This mail from mail service. Questionnaire.
   </ser:subject>
   <ser:message>Question1: Do you like mail service&#63;</ser:message>
   <ser:attachments></ser:attachments>
   <ser:timeout>00:10:00:00</ser:timeout>
   <ser:response>yes</ser:response>
   <ser:responseTemplate><![CDATA[&lt;wsResponse&gt;
       &lt;name&gt;Question1&lt;/name&gt;
       &lt;type&gt;string&lt;/type&gt;
       &lt;enumeration&gt;yes|no&lt;/enumeration&gt;
       &lt;min&gt;&lt;/min&gt;
       &lt;max&gt;&lt;/max&gt;
       &lt;/wsResponse&gt;]]>
   </ser:responseTemplate>
</ser:getRequest>
```

Listing 27. An example of a one-question request message

SendMail service sends a mail and the user will receive a mail which is shown in Figure 42. `mail_state` in the table `maildb.sentmails` for this request is changed to SENT.

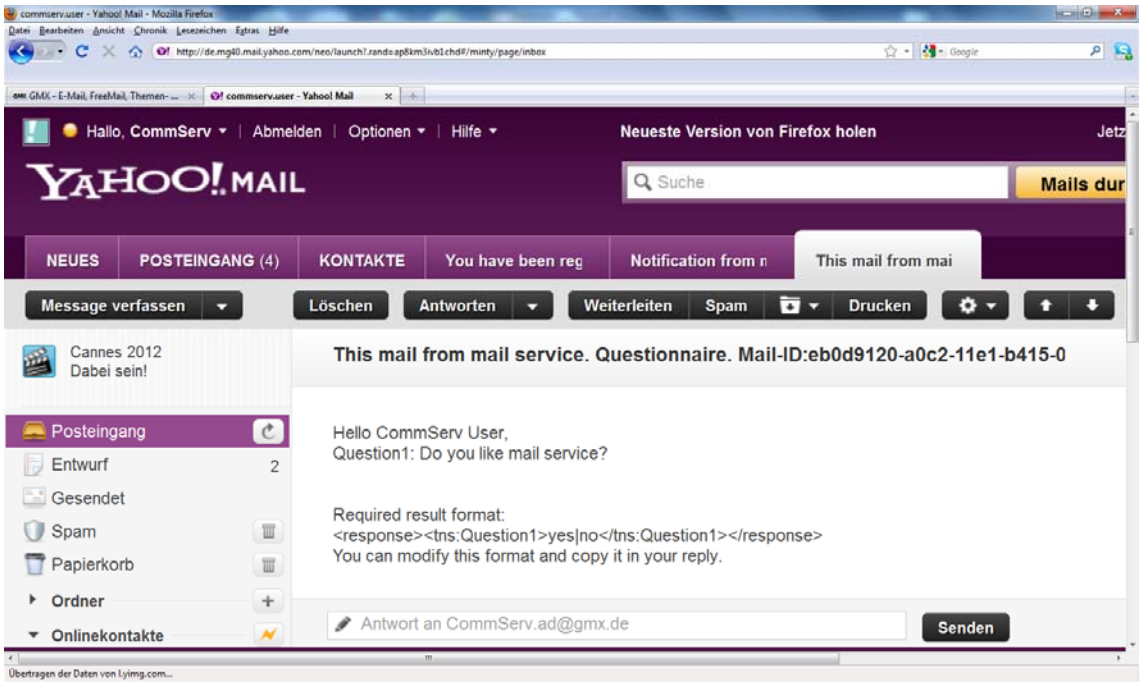

Figure 42. The user commserv.user@yahoo.de has received a mail which contains one question from Mail Service.

6.3.4.2 Multi-Questions in a Request

A request message can contain more than one question. The name of a question is a correlation between question and answer. Therefore the name of a question must be unique defined in a request. Listing 28 shows a request message which contains two questions, one question is `Question1: Do you like mail service?` , another is `Question2: How old are you?` Under element `<ser:responseTemplate>` there are two elements `<wsResponse>` for each question (line 12-25). The answer of question “Question1” should be a string with the values “yes” or “no”. The answer of question “Question2” should be an integer between 1 and 100.

```
<ser:getRequest>
    <ser:recipient>commserv.user@yahoo.de</ser:recipient>
    <ser:subject>
       This mail from mail service. Questionnaire.
    </ser:subject>
    <ser:message>Question1: Do you like mail service&#63;&#10;
     Question2: How old are you&#63;</ser:message>
    <ser:attachments></ser:attachments>
    <ser:timeout>00:10:00:00</ser:timeout>
    <ser:response>yes</ser:response>
    <ser:responseTemplate>
        <![CDATA[&lt;wsResponse&gt;
        &lt;name&gt;Question1&lt;/name&gt;
        &lt;type&gt;string&lt;/type&gt;
        &lt;enumeration&gt;yes|no&lt;/enumeration&gt;
        &lt;min&gt;&lt;/min&gt;
        &lt;max&gt;&lt;/max&gt;
        &lt;/wsResponse&gt;
        &lt;wsResponse&gt;
        &lt;name&gt;Question2&lt;/name&gt;
        &lt;type&gt;integer&lt;/type&gt;
        &lt;enumeration&gt;&lt;/enumeration&gt;
        &lt;min&gt;1&lt;/min&gt;
        &lt;max&gt;&lt;/max&gt;
       &lt;/wsResponse&gt;]]>
</ser:responseTemplate>
</ser:getRequest>
```

Listing 28. An example of a multi-questions request message

SendMail service sends a mail and the user will receive a mail which is shown Figure 43. `mail_state` in the table `maildb.sentmails` for this request is changed to SENT.

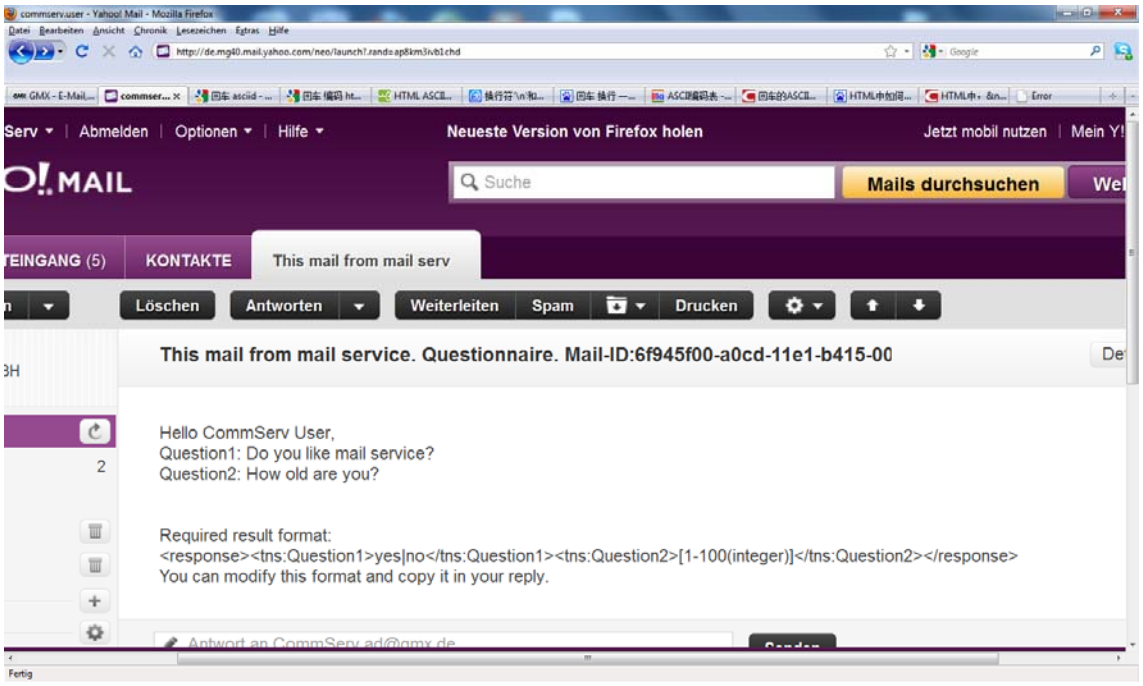

Figure 43. The user commserv.user@yahoo.de has recieved a mail which contain two questions from mail service.

6.3.4.3 Get a Valid Response

In the background ProcessMail service checks the value `mail_response` for the request with `mail_id` eb0d9120-a0c2-11e1-b415-0021869473a9. Because the value is “yes”, then ProcessMail service changes the value of `mail_state` from SENT to AWAIT_REPLY. Now ReceiveMail service gets a new mail, the content of this new mail is shown in Figure 44. After, ProcessMail service finds a response for this request in the table `maildb.mailbox`, and then it changes the value of `mail_state` from AWAIT_REPLY to REPLIED. Next, it checks the validation for this response. Because this response is a valid response, and then it changes value of `mail_state` from REPLIED to FINISHED.

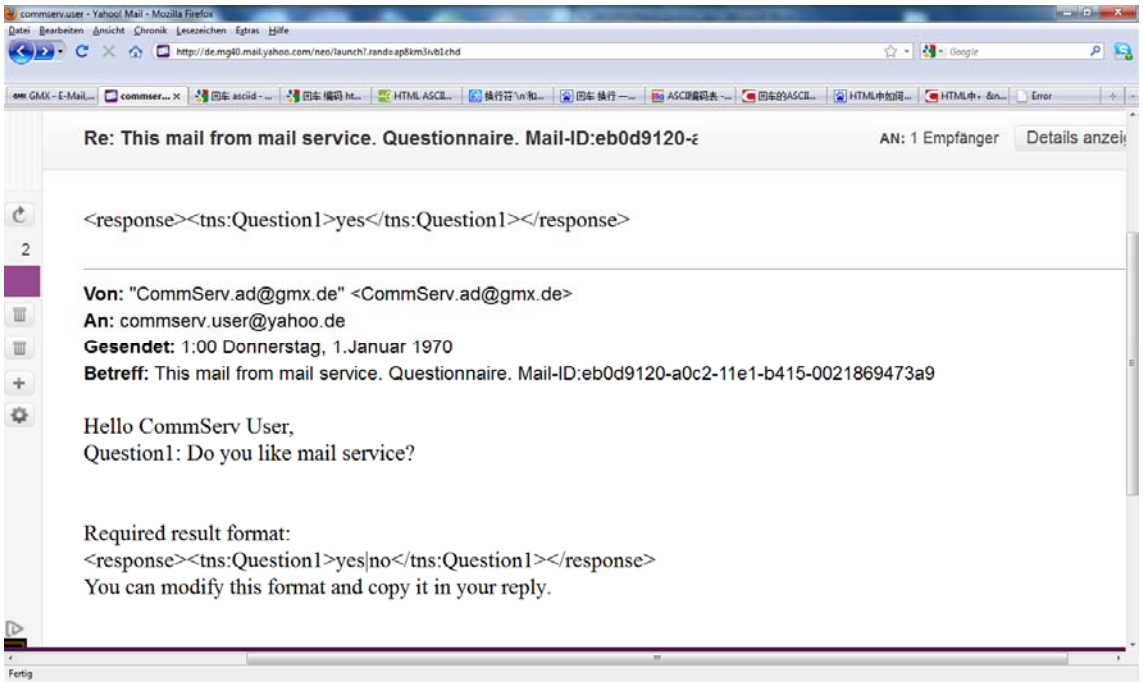

Figure 44. An example of a valid response for the request with mail_id eb0d9120-a0c2-11e1-b415-0021869473a9

6.3.4.4 Get an Invalid Response

After ProcessMail service changes the value of `mail_state` for the request with mail_id 6f945f00-a0cd-11e1-b415-0021869473a9 to AWAIT_REPLY, ReceiveMail service gets a new mail and the content of this new mail is shown in Figure 45. This mail is an invalid response, because the answer for the second question is 0 which is out of the range between 1 and 100.

After, ProcessMail service finds a response for this request in the table `maildb.mailbox`, changes the value of `mail_state` in the request from AWAIT_REPLY to REPLIED. Next, it checks the validation for this response. Because this response is an invalid response and the current time is not beyond the time limit, it adds “Invalid response(s)!” into the value of `mail_message` of the request as a prompt, changes the value of `mail_state` from REPLIED to NEW and the value of `mail_sentDate`.

This changed request will be resent. This process is a cycle, until the request arrives at an end state, FAILED, FINISHED or TIMEOUT.

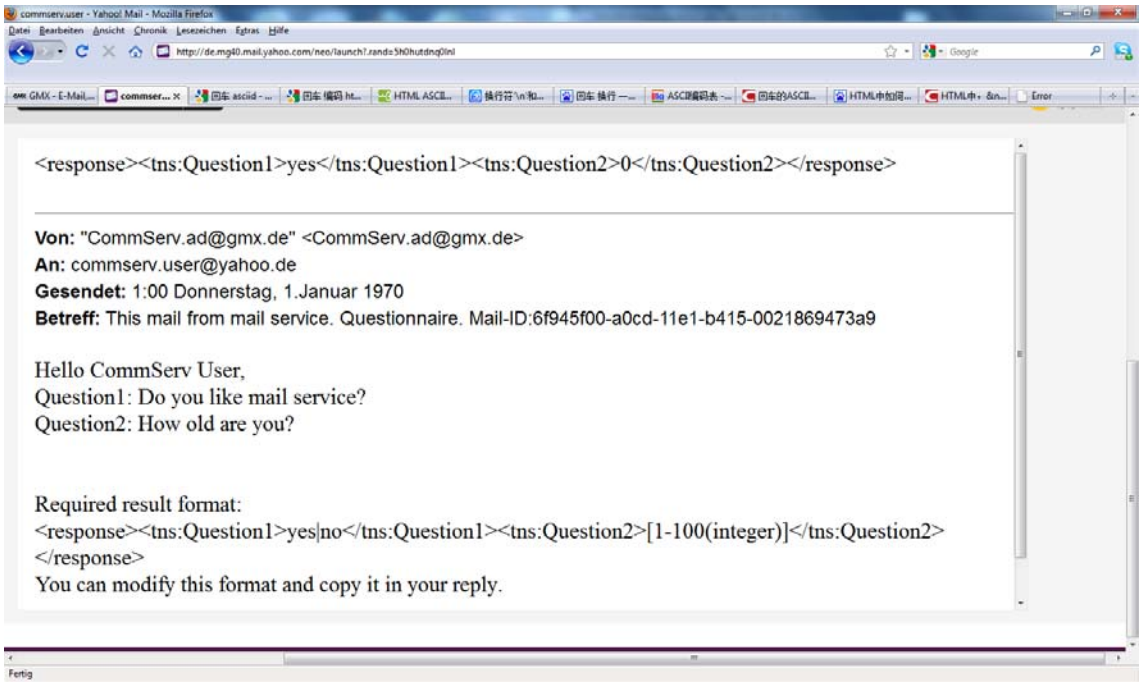

Figure 45. An example of a valid response for the request with mail_id 6f945f00-a0cd-11e1-b415-0021869473a9

6.3.5 Test Case5—BPEL Process invokes Mail Service and Skype Service

In this test case a BPEL process shown in Figure 48 is created to invoke Mail Service and Skype Service. According to input parameters the BPEL process can choose to invoke Mail Service or Skype Service. Input parameters for the BPEL process combine two input parameters of Mail Service and Skype Service, shown in Figure 46. Only parameter channel is new and it is an indicator for the BPEL process, which of Web Services should be invoked. If the value of channel is “skype”, then the process will invoke Skype Service to send a request. Otherwise, the process will invoke Mail Service.

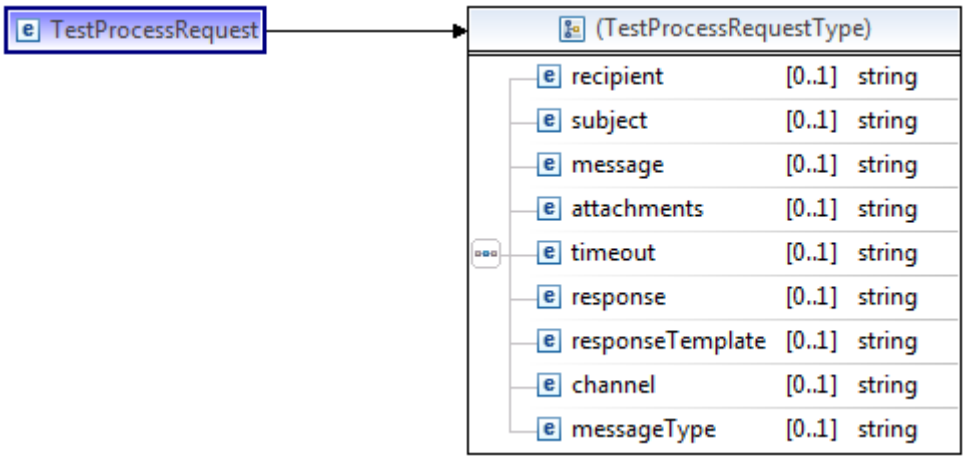

Figure 46. Input parameters for BPEL process

To send a message to a human user the BPEL process can invoke either GetRequest service of Mail Service or SkypeGetRequest service of Skype Service. Because Skype Service can only send a notification at present, so after invoking SkypeGetRequest service a string message will be returned to the BPEL process. For Mail Service a request can need a response. After invoking GetRequest service the BPEL process can receive the id for this request and can invoke GetResponse service of Mail Service with this id to query, whether Mail Service has received a response for this request. If Mail Service returns a string “noResponse”, the BPEL process will invoke GetResponse service again. Until Mail Service returns a valid response, or a string “TIMEOUT”, or a string “FAILED” to the BPEL process, the process will receive the output parameter and run completely. Output parameter for the BPEL process is a string, shown in Figure 47.

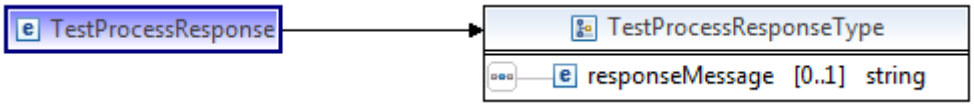

Figure 47. Output parameter for BPEL process

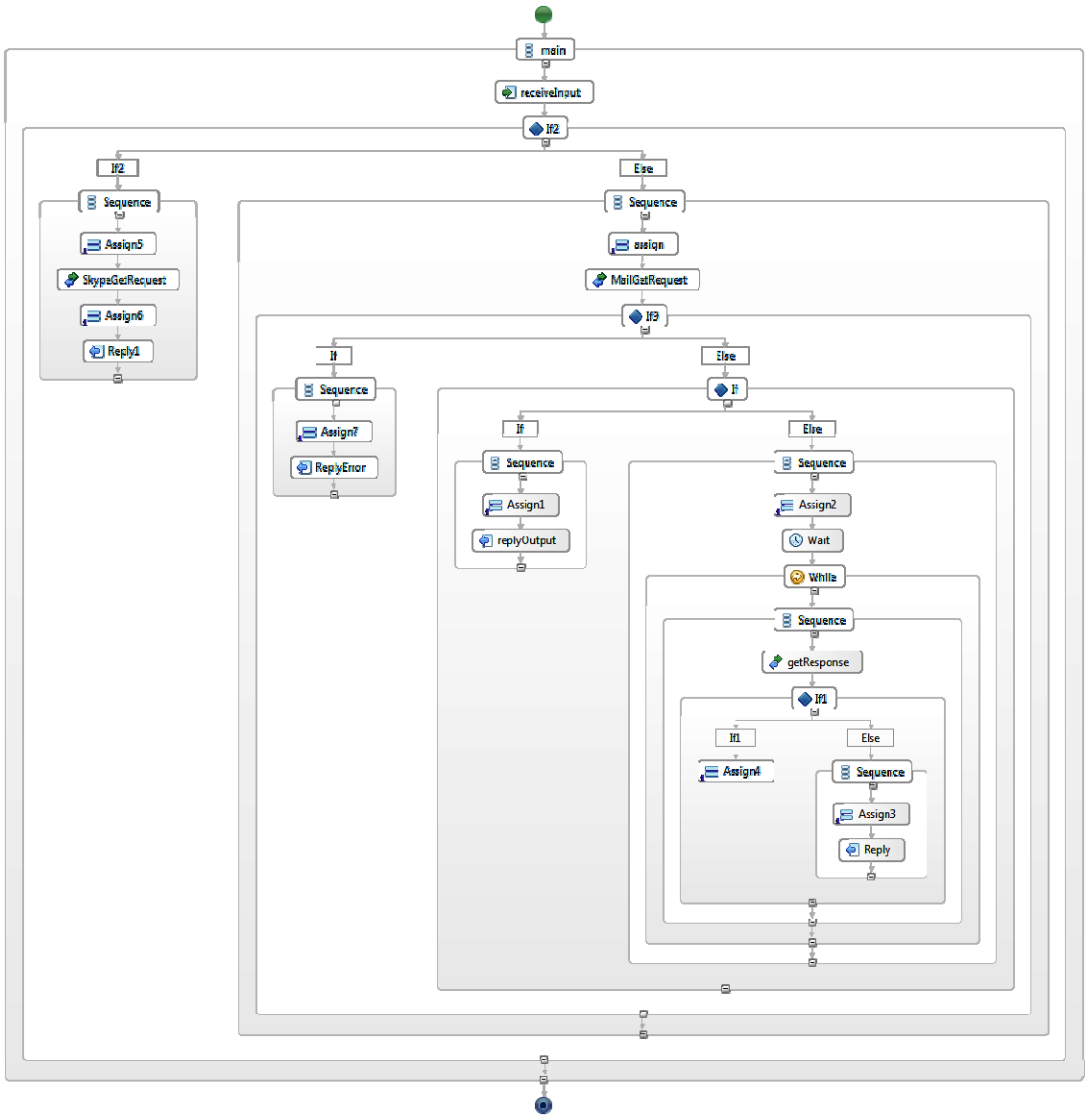

Figure 48. Test BPEL Process

6.3.5.1 Invoke Skype Service

The BPEL process receives the following request shown in Listing 29. According to this request the BPEL process will invoke Skype Service and send a text message to Skype user commservuser. In this case Skype user commservuser is a registered user for Skype Service.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sam="http://sample.bpel.org/bpel/sample">
   <soapenv:Header/>
   <soapenv:Body>
      <sam:TestProcessRequest>
         <sam:recipient>commservuser</sam:recipient>
         <sam:subject></sam:subject>
         <sam:message>test case1:skype notification message</sam:message>
         <sam:attachments></sam:attachments>
         <sam:timeout></sam:timeout>
         <sam:response>no</sam:response>
         <sam:responseTemplate></sam:responseTemplate>
         <sam:channel>skype</sam:channel>
         <sam:messageType>text</sam:messageType>
      </sam:TestProcessRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 29. A request to invoke Skype Service

After the BPEL process ran, it received the following response:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">

<soapenv:Body>

<TestProcessResponse xmlns="http://sample.bpel.org/bpel/sample">

<tns:responseMessage xmlns:tns="http://sample.bpel.org/bpel/sample">

The request is accepted.

</tns:responseMessage>

</TestProcessResponse>

</soapenv:Body>

</soapenv:Envelope>
```

6.3.5.2 Invoke Mail Service

The BPEL process receives the following request shown in Listing 30. According to this request the BPEL process will invoke Mail Service and send a message to commserv.user@yahoo.de and wait for a response in 3 minutes.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sam="http://sample.bpel.org/bpel/sample">
   <soapenv:Header/>
   <soapenv:Body>
      <sam:TestProcessRequest>
         <sam:recipient>commserv.user@yahoo.de</sam:recipient>
         <sam:subject>test case6:timeout test</sam:subject>
         <sam:message>question1: Is this request timeout&#63;</sam:message>
         <sam:attachments></sam:attachments>
         <sam:timeout>00:00:03:00</sam:timeout>
         <sam:response>yes</sam:response>
         <sam:responseTemplate>
            <![CDATA[&lt;wsResponse&gt;&lt;name&gt;question1&lt;/name&gt;&lt;ty
            pe&gt;string&lt;/type&gt;&lt;enumeration&gt;yes|no&lt;/enumeration&
            gt;&lt;min&gt;&lt;/min&gt;&lt;max&gt;&lt;/max&gt;&lt;/wsResponse&gt
            ;]]>
         </sam:responseTemplate>
         <sam:channel>mail</sam:channel>
         <sam:messageType></sam:messageType>
      </sam:TestProcessRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 30. A timeout request to invoke Mail Service

After the BPEL process ran and the recipient did not reply the mail in 3 minutes, then it received the following response:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">

<soapenv:Body>

<TestProcessResponse xmlns="http://sample.bpel.org/bpel/sample">

<tns:responseMessage xmlns:tns="http://sample.bpel.org/bpel/sample">

TIMEOUT

</tns:responseMessage>

</TestProcessResponse>

</soapenv:Body>

</soapenv:Envelope>
```

The BPEL process receives the following request shown in Listing 31. According to this request the BPEL process will invoke Mail Service and send a message to commserv.user@yahoo.de and wait for a response in 10 hours.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sam="http://sample.bpel.org/bpel/sample">
   <soapenv:Header/>
   <soapenv:Body>
      <sam:TestProcessRequest>
         <sam:recipient>commserv.user@yahoo.de</sam:recipient>
         <sam:subject>test case3: please answer this question</sam:subject>
         <sam:message>question1: How old are you&#63;</sam:message>
         <sam:attachments></sam:attachments>
         <sam:timeout>00:10:00:00</sam:timeout>
         <sam:response>yes</sam:response>
         <sam:responseTemplate>
            <![CDATA[&lt;wsResponse&gt;&lt;name&gt;Question1&lt;/name&gt;&lt;ty
            pe&gt;integer&lt;/type&gt;&lt;min&gt;1&lt;/min&gt;&lt;max&gt;100&lt
            ;/max&gt;&lt;/wsResponse&gt;]]>
         </sam:responseTemplate>
         <sam:channel>mail</sam:channel>
         <sam:messageType></sam:messageType>
      </sam:TestProcessRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 31. A request with response = yes

After the BPEL process ran and the recipient replied the mail in 10 hours, then it received the following response:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">

<soapenv:Body>

<TestProcessResponse xmlns="http://sample.bpel.org/bpel/sample">

<tns:responseMessage xmlns:tns="http://sample.bpel.org/bpel/sample">

&lt;tns:Question1>38&lt;/tns:Question1>

</tns:responseMessage>

</TestProcessResponse>

</soapenv:Body>

</soapenv:Envelope>
```

7 Summary and Outlook

7.1 Summary

Communication between human users and business processes will play an increasingly important role with the development of network. The main task of communication service is to build a stable and smooth way to communicate with human users. Once anything is related with humans, it will become more uncertain and difficult. In this work two channel are chosen for communication services: Mail and Skype.

E-mail is asynchronous way to exchange a message. Sender sends a mail to someone. To receive this mail a recipient does not need to be online at the same time. This mail will be saved on a mail server. When a recipient is online again, he can receive this mail. Skype is another way to send a message to human users. This message can be a text message or a file.

According to the characteristics of the entire communication system, Mail Service and Skype Service, some concepts are proposed for Mail Service and Skype Service. As the first concept of mail service there were a lot of problems in the concept, for example, instability, data loss. To improve this concept, a new concept is proposed. In the new concept a database as a new component is added in the communication services. The database plays an important role to store every message which has been sent or received from the communication services. This way is also very good to solve the problem “data loss”.

After the concepts are determined, all functions of the communication service are implemented by a bottom-up approach. First, the functions are implemented in Java Program. Then, some Java classes are deployed as Web Services, and Eclipse generates the WSDL documents for each Java class. WSDL document describes interface, data type, binding and operations for a service.

In this work Mail Service is implemented, Skype Service are still partially implemented due to the time constraint.

So far, the following functions are implemented in Mail Service:

1) Mail Service can get a valid request from an external process and store this request in the database.

2) Mail Service can send a mail to someone according to a request which is saved in the database.
3) Mail service can receive each new mail from the administrator's mailbox and store it in the database.
4) Mail service can check the validation of a response in order to update the state of the related request.
5) Non-registered user can register per mail in mail service.

The following functions are implemented in Skype Service:

1) Skype Service can get a valid text request from an external process and store this request in the database.
2) Skype Service can send a text notification to some Skype user according to a request which is saved in the database.
3) Skype Service can add new users which are waiting for authentication into the contract list.

Finally, Mail Service and Skype have been tested with the test tool SoapUI and BPEL. SoapUI calls GetRequest service and GetResponse service and sends some request messages on behalf of the different test cases in order to test the functionality of Mail Service. A BPEL process has been created to invoke Mail Service and Skype Service.

Mail Service has the following highlights:

1) In Mail Service all mails are divided into five classes: REQUEST, RESPONSE, REGISTER, NOTIFICATION, and UNKNOWN. All advertising mails which administrator receives belong to the class UNKNOWN. This distinguishes between an advertising mail and the others. These advertising mails will be not processed.
2) In Mail Service each mail has a state variable. The value of this variable can be NEW, SENT, AWAIT_REPLY, REPLIED, FINISHED, TIMEOUT, FAILED. Once a mail has been handled a step successfully, the value of the state variable must be changed. The addition of this variable is convenient for the administrator to manage and inspect all dates in mail service.
3) In Mail Service the response template is defined in XML Schema format. A response is defined in XML format. Using the relationship between XML Schema and XML the validation of a response can be directly checked out through calling the method `validator.validate()`.
4) By configuration in the file web.xml SendMail, Register, ReceiveMail and ProcessMail runs regularly and automatically, when the server starts up.

7.2 Outlook

In addition to the highlights mentioned in the previous chapter, mail service has the following shortages:

1) Callback

In the concept Mail service should return a message asynchronously. Because mail is an asynchronous way to communicate with humans, Mail Service often takes a long time to return a response message. But in the implementation this part has not been realized. To complete the functionality of Mail Service a new Java class GetResponse.java is created. This class has been also deployed as a Web Service. GetRequest service returns a mail id of a valid request message from an external process. The external process can call GetResponse, and the mail id as input parameter is passed in it. GetResponse is responsible for looking for a valid response for the mail id and returns it.

2) Skype Service

In this work the functionality of Skype Service has not been completely implemented. Current Skype Service can get a request containing only a text message and send a text messag to a Skype user. It should also get a request containing a file, send a file to a Skype user, receive a text message or a file from a Skype user, and process all request needing a response.

3) Error Handling

In this work error handling has been defined incompletely. The following errors in Mail Service are defined:

- Error in connection with a database
- SQL error
- Error in GetRequest
- Unregistered user

If the first three errors occur, an error message will be per mail sent to the administrator. For the last error, a warn message will be per mail sent to this unregistered user to notify the user.

Error handling in Skype Service has not been defined.

4) Attachment

Mail Service can add only an attachment in a mail. And the type of this file can only a a text file with extension .txt. Another aspect, Mail Service can also not process graphics files as an attachment.

Communication services can be made much better. At least three above mentioned points are the aspects needing to be improved in the future. In addition, other channels can be developed for communication services, for example, Skype, ftp and twitter. This work is just a beginning for the development of the entire communication services.

8 Table of Figures

9 Table of Tables

10 Table of Listings

11 Bibliography

[1]. National Institute of Open Schooling: Introduction to communication http://download.nos.org/srsec335new/ch1.pdf

[2]. A. Dix, J. Finlay, G. D. Abowd, R. Beale: Human-Computer Interaction, Second Edition (1998)

[3]. W. Schramm: Family Communication Model http://fatherhood.about.com/od/familycommunication/a/A-Model-For-Understanding-Family-Communication.htm

[4]. M. Weske: Business Process Management, Concepts, Languages, Architectures (2007)

[5]. F. Leymann, D. Roller: Production Workflow, Concepts and Techniques (2000)

[6]. E. Christensen, F. Curbera, G. Meredith, S. Weerawarana: Web Services Description Language (WSDL) 1.1, W3C Note (2001) http://www.w3.org/TR/wsdl

[7]. D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard: Web Services Architecture, W3C Working Group Note (2004) http://www.w3.org/TR/ws-arch/#whatis

[8]. M. P. Papazoglou: Web Services: Principles and Technology (2008)

[9]. E. Cerami: Web Services Essentials (2002)

[10].M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, Y. Lafon: SOAP Version 1.2 specification, W3C Recommendation (2007) http://www.w3.org/TR/soap12

[11].S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. F. Ferquson: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More (2005)

[12].M. MacDonald, M. Szpuszta: Pro ASP.NET 3.5 in C# 2008, Chapter 31: Creating Web Service (2008) http://www.c-sharpcorner.com/uploadfile/prvn_131971/chapter-31creating-web-services/

[13].E. Rusty Harold: XML Bible (1999)

[14].D. Schumm, C. Fehling, D. Karastoyanova, F. Leymann, J. Rütschlin: Process for Human Integration in Automated Cloud Application Management (2012) http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=TR-2012-02&mod=0&engl=0&inst=IAAS

[15].G. Hohpe, B. Woolf: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions (2003)

[16].D. K. Barry: Web Services explained http://www.service-architecture.com/web-services/articles/web_services_explained.html

[17].D. Schumm, D. Karastoyanova: Integrating Humans in Scientific Workflow: Integrate, Register & Communicate. Poster, The 4th Simtech Status Seminar (2011)

[18].Skype: http://www.skype.com/intl/de/home

[19].Twitter: http://twitter.com/

[20].L. M. Surhone, M. T. Tennoe, S. F. Henssonow: Robustness (Computer Science) (2011)

[21].H. Störrle: UML 2 für Studenten [Mit UML-Syntax-Poster] (2005)

[22].JavaMail API documentation: http://javamail.kenai.com/nonav/javadocs/overview-summary.html

[23].J. Myers, M. Rose: Post Office Protocol – Version 3 (1996) http://tools.ietf.org/html/rfc1939

[24].MySQL Connector: http://dev.mysql.com/downloads/connector/

[25].API Specification for the Java 2 Platform, Standard Edition, version 1.4.2. http://docs.oracle.com/javase/1.4.2/docs/api/overview-summary.html

[26].ORACLE Document: web.xml Deployment Descriptor Elements http://docs.oracle.com/cd/E13222_01/wls/docs81/webapp/web_xml.html

[27].Apache Axis2: http://axis.apache.org/axis2/java/core/

[28].JavaMail API: http://java.sun.com/products/javamail/downloads/index.html

[29].Skype API: Getting Started with Skype4Java http://graphics.cs.columbia.edu/courses/csw4170/Skype4JavaTutorial.html

[30].SoapUI: http://www.soapui.org/

[31].J. Bolie, M. Cardella, S. Blanvalet, M. Juric, S. Carey, P. Chandran, Y. Coene, K. Geminiuc, M. Zirn H. Gaur: BPEL Cookbook: Best Practices for SOA-based integration and composite applications development (2006)

12 Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ort, Datum

Unterschrift