NTNU
Norwegian University of
Science and Technology

Composing end-user services

Jens Einar Heide Vaskinn

Master of Science in Communication Technology
Submission date: June 2009
Supervisor: Mazen Malek Shiaa, ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem Description

The task is to develop a simple-to-use infrastructure for end-user service composition. Services that we are focusing on are the services aimed for all kinds of end-users who are in the city. Example services can be: appointment organizer, personal manager, calendar, parking assistant, reservation service, shopping helper, traveller agent, notification handler, etc. Such services can be composed to provide an added-value service to tourists in the city, patients looking for doctors, youngsters going out to sport events, club searchers, shopping planners, car drivers, etc. The vision is that ordinary people can easily compose a set of available services and run them on their handheld devices while they are on the run.
The student will first investigate the possibilities for exploiting existing service composition mechanisms and adapt them to the specific requirements for end-users. Second, a set of simple service composition scenarios will be worked out and specified. Third, the student will design a prototype for composition using either a graphical editor or a programming interface. Last, the student will select certain scenarios for execution in real execution environment on end-user devices – here we will use the infrastructure of Wireless Trondheim in the down town area.

Assignment given: 15. January 2009
Supervisor: Mazen Malek Shiaa, ITEM

Abstract

Service composition is currently a very hot topic in the Service Oriented Computing area. End-user programming is one aspect of this. This thesis proposes one such end-user programming environment for telecom services where a user can create, edit and set up a self defined behaviours when e.g. receiving calls or sms. The environment consists of services which can be used to program the desired behaviour. Some useful service is defined and a xml representation of them has been worked out.

The thesis takes a scenario based approach to this and uses different real life composition scenarios to shed light on several aspects of the programming interface and service composition e.g. creating compositions, combining compositions and constraints.

Preface

This master thesis is written at the Department of Telematics at the Norwegian University of Science and Technology (NTNU) in the spring semester of 2009. The master thesis is submitted as a fulfilment of the graduation requirements of my Master degree in Communication Technology.

I would like the opportunity to thank my supervisor for giving me good advice and pushing me forwards.

I would also like to thank my mother and father for helping me keep the motivation to finish and for proofreading during the final week.

A great thank you also goes to my fiancée for holding out through my frustration, mood swings and inability to take part in social activities in the final period. Supporting me, and keeping my spirits up. I could not have done this without you.

Contents

List of Tables

List of Figures

1. Introduction

1.1. Motivation

From the plain old telephone service until to today there has been an enormous development in available services. The most important being that one now is available 24 hours a day and one can expect to be called by mostly anyone at any time and at any place. Being so available can often be more annoying than it is practical. One may often want to add restrictions to which calls are accepted at certain times and maybe also treat the calls from various sources differently.

On the other side, now it is not uncommon to have 2 or 3 mobile phones with a stationary work phone and home phone on top of that. How should a caller know which phone number to try at the different times of day. Surely it is okay that there is some people that only know one specific number, but there is some people that you would want to reach you any where you are. Would it not be easier for your child to only remember one number? Or even better for the little child that has just got his first phone, that any number dialled goes to one of the family members?

1.2. Problem to be solved

Early on in the thesis period it was decided to alter the direction of the project text a bit. Gintel has a complete interface for composing Telecom related services for enterprises. This application (called Easy Designer) was created with technical persons and application experts in mind, not for end-users in other words. From this it was decided to also have a scenario driven approach that was generic enough, however linked to the specific service composition platform used in Telecommunication networks (an existing execution environment in a real telcom operator's network). With the Easy designer in mind a new application directed at end-user composing of the telecom services was proposed. So a new task was agreed upon: Develop a equivalent solution for end-users. Enabling the end-users to set up their telecom services in their one time at their based on their own preferences.

1.3. Project Outline

This thesis report is organized into 7 parts. **Chapter 1:** Introduction, presents the motivation and the problem to be solved. **Chapter 2:** Background, describes the core technologies that this thesis builds on. **Chapter 3:** End-User service composition, narrows down the problem to a set of tasks, an approach for the thesis is chosen and a method of execution is presented. **Chapter 4:** Implementation, presents the results and the work that has been executed. **Chapter 5:** Discussion, a discussion of the accomplishments and the scope of the thesis and also some future development is suggested. **Chapter 6:** Conclusion of the thesis. **Appendix:** The resulting XML representations for all the services is moved here from the XML representation section to make the implementation chapter a bit cleaner.

2. Background

2.1. End-User Service Composition

Finding related background material directly related to end-user service composition has not been easy as most of the existing service compositions solutions are aimed for supporting professional developers, with solutions such as automated service composition, model-driven service composition [BOP], Semantic-Web-enabled composition [RKM], QoS-aware service composition [MS], and businessdriven automated composition [OYP].Combining Service oriented architecture and service composition combined and end-user programming serves well as backgroung information. The service oriented architecture and service composition covers the technological aspect and end-user programming covers the usability for the end-user.

2.2. Service Oriented Architecture

Service Oriented Architecture is the next wave of application development. There is no widely agreed upon definition of Service Oriented Architecture (SOA). This is probably because SOA is much more of a mind set, development philosophy and an architecture, than it is any concrete and definite. But it is agreed upon that it is an architecture that relies on service-orientation as its fundamental design principle. Service Oriented Architecture is an adaptable, flexible style of architecture, that provides the foundation for shorter time-to-market and reduced costs and risks in development and maintenance.

2.2.1. What is a service?

A service is a functionality provided by a service provider to a service user. A service user could be a human end-user an other service or a type of software component. Services are network addressable software components. Services that are addressed in this report will be mainly related to the Telco and the Web-based services, such as online stock quote, weather information, calendar, Instant Messaging, audio conference, etc. In this context, services should have well-defined description and semantics. Description could be a WSDL interface and semantics could be captured in an ontology like description such as OWL-S. In an environment implementing SOA independent services can be accessed without knowledge of their underlying implementation [CHT].

2.2.2. Basic SOA

SOA in its most basic form consists of a collection of services provided by a service provider, and a client, or service consumer, making use of these services. Figure 2.1 illustrates a basic service oriented architecture. On the left we have a Service Consumer sending a service request to a Service Provider on the right. The Service Provider returns a service response to the Service Consumer.

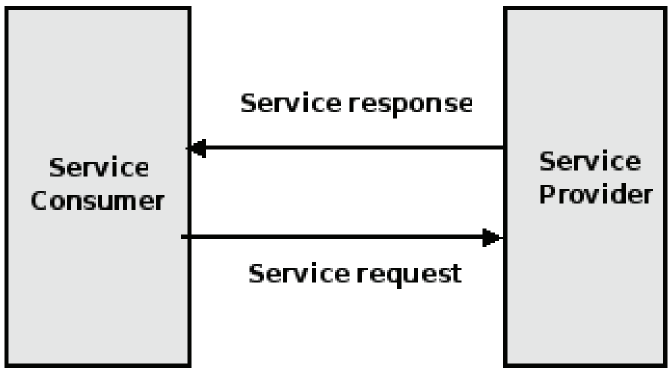

Figure 2.1.: Basic Service Oriented Architecture

These two components alone is not sufficient for SOA as a distributed software model. According to IBM, SOA is comprised of three participants and three fundamental operations, regardless of its implementation [B]. These are shown in figure 2.2.

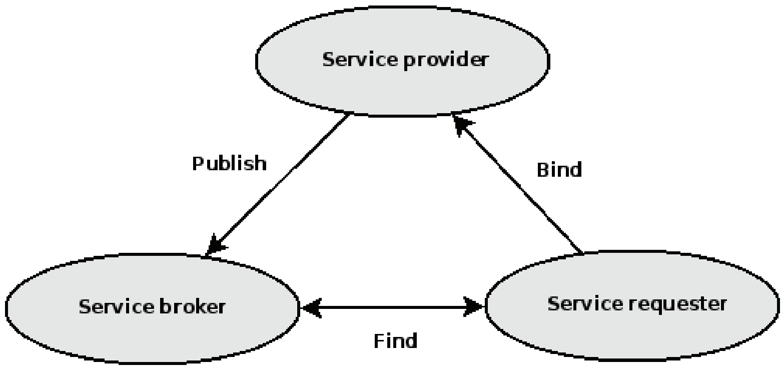

Figure 2.2.: The Service Oriented Architecture Model

Service Provider:

The service provider makes the service available with it's Service Contract and publish it on the Service Broker. A Service Provider is a network node that provides a service interface for a software asset that manages a specific set of tasks. A service provider node can represent the services of a business entity or it can simply represent the service interface for a reusable subsystem.

Service Broker

The Service Consumer find the compatible Service and its Service Contract using the Service Broker. The Service Broker is a Network node that acts as a repository ("yellow pages") for interfaces that are published by service providers. A business entity or an independent operator can represent a service broker.

Service Requester

The Service Requester and the Service Provider interact after the consumer has found an adequate service and its Service Provider. The Service Requester, or Service Consumer, is a Network node that discovers and invokes other software services to provide a business solution. Service Requester nodes will often represent a business application component that performs remote procedure calls to a distributed object, the service provider. In some cases, the provider node may reside locally within an intranet or in other cases it could reside remotely over the Internet. The conceptual nature of SOA leaves the networking, transport protocol, and security details to the specific implementation.

2.3. Service composition

Service composition is currently one of the most hyped and addressed issues in the Service Oriented Computing area. Service composition, in this context, is the combination of existing services to provide an enriched, valueadded composite service with an overall functionality.

Services used in a composition can both come from different providers and different domains. To be able to use such services in a composition there are some preconditions that needs to be filled:

- The services needs to have their interfaces exposed and have operations that can be invoked remotely
- The services needs to be published and discoverable from a service repository
- They need to be described in a formal language.

The formal description of a service component is often annotated semantically and reflect their intentions and semantics. These semantics are captured using ontology.

There are several approaches to servicecomposition. Some of them use the principles of ServiceOriented Computing (SOC), while others are based on Artificial Intelligence (AI). Many approaches use semantic web services, by which web services are annotated with semantics, such as service description, provider details, service operations details, service intentions, service parameters, etc. Annotations follow formal terminologies, which are called ontology. In [M] they identify five different approaches for service composition. These classifications is widely recognized in the research area (It is used in [DS] and [UK] for instance). These will be discussed briefly in the text below.:

- Natural language based composition
- Goal-based composition
- Choreography-based composition
- Functional level composition (FLC)
- Aggregation of non-functional properties

The natural language approach allows the service consumer to specify its service request in an informal way, i.e. in natural language. A formal specification is then derived from this request and can be used as an input to the composition engine. The core idea is to match fragments of the natural language request to semantics known to the composition engine, ontology elements and service descriptions.

The objective of the goal-based approach is to provided a composite service based on a request expressed in a certain format. This request includes a description by goals, i.e. the effects of the service in the "real world". An example for a goal could be "book a flight" or "send a message". This approach uses the description of the goals in different steps. At the service discovery and selection time, potential services are selected based on the goals they achieve. At the composition step, services are assembled together based on the semantics which includes the goals.

Choreography-based composition evaluates if a composite service matches a standard workflow from a problem domain, a choreography. The worklow can be seen as a composite service because it is a set of relations service provider/consumer. The flow of services can be described in a BPEL flow.

The focus of FLC is on selecting a set of web services that, combined in a suitable way, are able to perform a service composition request, i.e. an abstract service described by its Inputs, Outputs, Preconditions and Effects. Since web service discovery and FLC are very close processes, they are usually combined together to find a chain of suitable services. The service composition request defines the overall functionality that the composed service should implement.

The aggregation of non-functional service properties checks that a given composite service matches the non-functional properties specified in the service request. Additionally, the analysis of these properties allows to establish a ranking of different composite services. Such properties can be cost, response time or reliability.

The service composition process is triggered by a service need. When a service is needed[1], a service requester specifies a service request containing the desired properties of the service. The service requester sends the service request to a repository that contains descriptions issued by service providers of the services they provide. A search is then executed to discover a service with the desired properties in the repository. If a

[1]Needed can here mean both needed at design time and needed to use at run time

service with the desired properties could not be discovered, some algorithm is executed that will construct a service composition automatically.

2.4. What is End-User Programming

End-User programming is closely related to End-User service composition. Both subjects wants to achieve the same goal. Let the user create an application or service specially for his or her needs.

Most people experience computers as end-users of packaged programs. Unfortunately the writers of these programs can't know the details of the job you are trying to do. Trying to meet the needs of diverse users, they bloat their programs with hundreds of features most people never use. Life (and programs) would be much simpler if each user could add the functions she wanted.

End-User Programming is a subset of the term programming. Programming is the process of writing, testing, debugging/troubleshooting, and maintaining the source code of computer programs. What most people associate with programming is the writing of source code in a programming language. The code may be a modification of an existing source or something completely new. The purpose of programming is to create a program that exhibits a certain desired behaviour (customization). The process of writing source code often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms and formal logic. End-User programming differ from programming in that it is simplified and specialized for a specific domain so that a person with less knowledge in programming languages and techniques can effectively achieve his or her desired results. But how much can one simplify and limit the possibilities before it is just altering options and settings. [N] addresses this: "The objective of programming is to create an application that serves some function for the user. From the end user's point of view, the particular behaviour involved is not important, so long as application development is easy and relative rapid. In this respect, we can include automatic programming system, programming by example systems, and form-filling dialogues in which applications can be customized."

In traditional object-oriented software development where professional programmers are put on the task much of the focus is on developing the collaboration among the classes, organizing class hierarchies, and implementing the classes. The focus of end-user programming environments is often more directed at the instances: creating object instances and setting their properties, and defining the special cases of behaviour. The added behaviour may affect only a single object, or in some cases a group of objects.

Providing a end-user programmable interface is not trivial. The programs must be designed to to accept user-written components in appropriate places. There must be a way to store and manage them. Most important, since most users do not have the time or inclination to learn the tools and skills of a professional programmer, reasonable compromises are required. The expressiveness and generality of full-fledged programming languages are traded for usability by a variety of metaphors and tricks. Programming can be done much more easily within the metaphor – a desktop with file cabinets and wastebaskets; a formula of spreadsheet locations or mathematical symbols; a sequence of GUI actions; a circuit diagram; an application-specific language – than with conventional programming.

Because the appropriate metaphors, with their capabilities and limitations, differ widely depending on the users and their purposes, there is no one method of end-user programming. Instead there is a variety of techniques, such as Programming by Demonstration, visual programming, and many domain-specific languages and formalisms. Ideally there is a smooth progression from simple but limited metaphors, to more complex and powerful techniques as the user-programmer advances.

2.5. End-User Programming Approaches

In this section a series of different End-User Programming approaches is described. Some of which has to some degree been an inspiration to the environment developed.

2.5.1. General Purpose Programming Languages

This is the earliest form of attempted end-user programming approaches. There have been many attempts to create such languages but most of them has failed to be useful as a end-user programming tool. Common for most of these is that they tent to be to complicated for a normal person and better suited for programmers and developers. Well known examples of this is Visual Basic, SmallTalk [S_1] and ToonTalk [K].

Smalltalk was one of the first systems to pioneer the WIMP (Windows, Icons, Menus and Pointers) interface. Compared to conventional programming languages such as C or Pascal, Smalltalk has an unusual syntax. Objects are employed to represent everything in Smalltalk, including all the conventional data types that may be found in any programming language: such as integers, Booleans, floating-point numbers, characters, strings, arrays. In addition to this, objects are used to represent display items such as menus, windows and even the compiler itself. Smalltalk is therefore described as a uniformly object-oriented language. Smalltalk language was developed at the Xerox Palo Alto Research Center in 1970s, the programming environment greatly influenced the development of Apple Macintosh and Microsoft Windows.

ToonTalk [MKK] is a child-oriented programming language whose environment is an animated virtual world, with objects that children can pick up and use as in a game, such as birds, trucks, and robots, providing direct child-oriented metaphors for programming constructs. Actions performed by a programmer's avatar with these objects are both code and coding. ToonTalk is a powerful system, not just a "toy" system: it is based upon concurrent constraint programming languages, and programs written in languages such as Flat Guarded Horn Clauses and Flat Concurrent Prolog can be straightforwardly constructed in ToonTalk. However, there is not a specification of ToonTalk, for ready implementation in other environments. They propose that the ToonTalk language lies not in the animations displayed by the current environment, but on the actions performed by the programmer with virtual world objects. They present a description and analysis of the methods the ToonTalk language provides to programmers for expressing programs.

2.5.2. Scripting

A lot of programs has included some sort of scripting possibilities. Scripting languages is supposed to be easier to learn than general programming languages. These languages tend to be less rigid and more domain specific. Being simpler scripting languages usually have some limitations compared to general purpose languages. With the growth of the internet scripting languages like JavaScript and VBScript are used in the web pages and and interpreted by the client's browser.

2.5.3. Macro Recording

The idea of macro recording is that the user can record a series of events done during a task and simply replay the recording. Enabling the user to do repetitive tasks much faster. The problem with macros is that they are usually to specific. So when the user wants to modify a value in some way the macro just replaces the value with the same as in the recording. The macros are recorded as scripts, so if the user wants to modify the macro he or she has to understand that scripting language [RP].

2.5.4. Programming by Example

In programming by example the user creates one or more examples of what the program is supposed to do. The program is then derived from the users demonstration. Programming by example is very similar to macro recording, but was created to try to solve the problem of the macro recording being to specific. In programming by example (or by demonstration as it is also called) the system derives a generalisation of the problem from one or more examples. AI techniques are often applied in this area, and most of the research in on AI problems.

2.5.5. Visual Programming

Sanscript from Northwoods [S_2]Software is a good example of this. Sanscript has an programming environment based on dataflow. Programs are created by adding objects to a visual function called a flowgram. Objects can be constants, control structures, forms or functions from a component library. Each object has inputs and outputs. This can be connected to other objects inputs and outputs. Prograph from Pictorius is another example of an environment based on dataflow. Prograph is much the same as Sanscript, but is more object oriented.

Lego MindStorms is a programming environment for programming Lego robots. A program is created by placing program pieces into one or more sequences. Each sequence will become a process. Each sequence can be connected to a triggering condition, for instance that a touch sensor was pushed or a light sensor reached a threshold value. Other program pieces are commands that control the speed and direction of connected motors and timing. The system is however a closed world and is not created to be used to any thing other than controlling Lego robots and such.

3. End-User Service Composition

3.1. Problem Description

Develop a infrastructure for end-user service composition. The service composing infrastructure should be directed at composing services for administering incoming and outgoing calls, logging functions, SMS, instant messaging etc. Give the user near total control over his telecom services through specifying behavioural responses to events in communication, user activities and time. Explore existing approaches and use or adapt them to fit the needs in this project. The application is to be used by end-users and hence some considerations should be taken concerning that.

3.2. Related/Existing Approaches

3.2.1. Spread Sheet

A spreadsheet is a computing application that displays a rectangular table (or grid) of information, consisting of text and numbers, where values sit in cells. Spreadsheets allow defining the type of data for each cell (usually limited to texts and numbers) and defining how different cells depend on each other. The relationships between cells are defined through formulas. Users can interactively change the data and formulas and immediately see the effects of their actions. When using spreadsheets as a service composing tool one need a frame work around the spreadsheet to send requests and retrieve results from various local and remote services. In [OG] they show a solution to this and suggests tools to support different composition patterns and show how the style of declarative dependencies of spreadsheets can facilitate service composition.

In [OG] they argue that the spreadsheet functions proved to be easy to understand for most end users and powerful enough to enable usage of most of the complex services. AMICO:CALC proved to be a very good tool for rapid prototyping as it allowed easy and real-time service compositions and demonstrations by inexperienced developers and end users.

With regard to performance, the spreadsheet environment introduces a delay that is a consequence of calculating formulas and cell dependencies, as well as a communication overhead between the middleware and the spreadsheet extension.

3.3. Selected Approach

[LHM] The complexity of specification and infrastructure is the main obstacle for end users when using current professional composition technologies. It is thus necessary to provide an end user oriented composition environment which makes them feel like using desktop applications. The environment should be as simple as and as familiar as possible, where the end-users will enjoy high usability and personality.

The spread sheet approach seemed quite interesting from a prototype point of view enabling a quick way to test the proposed prototype, but from a end-user point of view It could arguable be boring however simple it would be to use.

So the chosen approach was to use one or some of service composition approaches and combine this with with one or some of the approaches to end-user programming. The services composition approaches that looked interesting were:

The natural language based service composition, using this could in some way could let the user specify the desired behaviour with simply typing sentences. Using the natural language approach would also impose using some goal based or semantic based approach and or ACE [UK] and [M]. Using a natural language based approach would however not give the use any real control over the created behaviour, which would be a requirement of the problem description.

To give full control over how the service interact a choreography based approach would be much more suited. Combining this with some of the key elements from visual programming would give the use an interesting an easy to learn interface.

3.4. Method

Do a search for related/existing approaches to the problem. Study a existing service composing application for composing Telecom services for enterprises developed by Gintel. Create a sett of example services suitable for end-users to be used in the services composing interface. Work out and specify a set of service composition scenarios. Define a XML representation for the services and a way to represent compositions when after they are designed.

4. Implementation

4.1. Example services

4.1.1. Size of services

One important issue is the size of the services and components in the user interface. Large and composed services are often easier to use as one need less services to achieve the results one wants. The user interface gets less cluttered and gives a better overview of the resulting behaviour. Smaller components gives better re-usability and custumizability. A disadvantage of smaller components is that the complexity increases a more components is needed for the same result. Smaller components also gives a lot more possible end results and a more care is needed to ensure no conflicts, infinite loops and other errors in the user-programmed result.

4.1.2. Services that are not necessary building blocks in the behaviour designer

To be able to create a behaviour designer one need to have a foundation of working services that the other services can depend on and be supported by. The following subsections describe some of the important ones.

Calender

As the name states, this is a calender. It can be any calender or appointment handler that has a interface compatible with the Activity handler. A calender services is necessary to keep track of appointments and activities so that this can be utilized to provide a special behaviour during such activities.

Directory Service

This is a service that enables uploading of files, saving settings. Also it would be beneficial for the users to have the possibility to publish their designed behaviour with a description and also have the possibility to download the designs of other users.

Help and Documentation

For the user to be able to understand the flow of the program and understand how to create satisfactory behaviour there has to be a good documentation.

Composition Validity Checker

When the user is finished designing his behaviour, the validity of the composition needs to be checked. This service goes through the designed composition and checks for discrepancies. This should preferably be done automatically and incrementally when the composition is designed so that the user can be notified of any conflicts at when designing. A check should also be done right before deploying the designed behaviour so the composition as a whole is checked.

4.1.3. Trigger Services

In this domain it is two main categories of events that can start a behaviour. As in ordinary programming this is either a external or internal event. External events are here interactions from either a caller or the callee. Caller events are incoming calls, incoming messages, phone hang up, key presses (DTMF), and voice responses to a voice recognition service. Callee events are reject call, answer call, disconnect call, key presses (DTMF). It might not be all that naturally to think that disconnecting a call is the start of a behaviour, it would be easier to think of it as the end of a behaviour (as it of course can be). But it can also be used as a trigger or starting point. For example a user wants a SMS or e-mail sent some were with call details after the phone is hung up. Internal events are mostly timers or services used in combination with timed triggers. This could be the start of an appointment/task or event in the calender or a periodic task on for example a hourly or daily basis.

Activity Handler

Description

This service interacts with the calender. In this service one can define what behaviour one wishes when participating in various activities. By default it should come with a set of standard activities, but it should also have the possibility for the user to define his or her own. For each type of activity one can attach a desired activity.

Settings and options

- Define the next service for each activity.
- Set up a connection to the calender in use.

Periodic Tasks

Description

This service lets the user trigger a behaviour on a timed basis. The behaviour can have on periods and of periods and also have a interval between the triggers in the on period.

Settings and options

- Active from day/time to day/time.
- Weekdays, times for each day.
- And an interval for the trigger.
- Define the service to be triggered.

SOS Short Codes

Description

SOS or call for help list. Intended to be used when there is a need for help, but not necessary a need to call 113 or 911.

Settings and options

- Define the short code that triggers the behaviour.
- Define the call list and time-out before trying the next number on the list.
- Next service.
- Customizable short codes.

Custom Trigger

This service is a container for specialized features. This could allow an advanced user to program and define a custom trigger for a behaviour. E.g. use the information from a weather service so that you can trigger a behaviour for good weather.

Call Handling

This is a group of triggers that relates to the handling of calls. They do not have any settings but the ability to define the next service.

Answer Call Connect a behaviour to when calls is answered.

Disconnect Add behaviour to when a call is ended.

Reject Call Start a behaviour to when calls are rejected.

No answer Start a behaviour to when calls go unanswered.

Outgoing Call Connect a behaviour to outgoing calls. (e.g. Blacklist)

Incoming SMS Start a behaviour when receiving an SMS.

Outgoing SMS Start a behaviour when sending an SMS.

4.1.4. Filter Services

Caller analyser

Description

Defines custom behaviour of the incoming calls or messages based on the number of the caller. The user can define a list of numbers and assign a next service to those numbers.

Settings and options

- Define next service for each number or assign a next service to a set of numbers.
- Next service if none of the numbers match.

Blacklist

Description

A list of caller-ids that are blocked. Originally intended to be used as a "‘spam"’ blocker. Blocking telephone salesmen and other annoying callers. Normally 2 lists (incoming / outgoing) would suffice, but it should be no problem defining several lists which can be used and reused in the different instances of the blacklist services.

Settings and options

- Add, Edit and Remove numbers in the blacklist.
- Next service if blacklisted.
- Next service if not blacklisted.

White list

Description

A list of numbers that are important. VIP – list could be a other name of this service. A number entered in this list is thought to be a important person, authority, relatives or other persons that the subscriber wants to talk to or does not mind being disturbed by at any given time. The white-list service is not much use by it self. To create a useful composition one needs to combine white-list with at least one other service providing some sort of blocking and the white-list functions as a bypass of this blocking. Normally 2 lists (incoming/outgoing) would suffice, but it should be no problem defining several lists which can be used and reused in the different instances of the whitelist services.

Settings and options

- Add, Edit and Remove numbers in the whitelist.
- Next service if whitelisted.
- Next service if not whitelisted.

Location Finder

A service that finds the location of terminals. This could be the users location but also the location of family members or persons that has accepted to be located by the user.

Settings and options

- Which terminal to locate.
- Possibility to match a location or to report the current location to the next service.
- What area or location should be accepted as a match and trigger a behaviour.

Time-filter

Description

This service enable behaviour to be enabled at certain times of the day/week/month/year. Calender with activity handler can serve the same purpose, but this is a simpler version that works on a regular basis.

Settings and options

- Define the time of day, days of week, months of the year for when the succeeding behaviour is active.
- Define the next service if none of the time filters match.

Caller Choice

Description

Present a menu to the caller and let the caller choose what to do.

Settings and options

- Define a greeting message.
- Define the menu items.
- Define the voice message for each menu item.

Callee Choice

Description

Gives the callee a choice of what to do. This menu works in a similar manner as the caller choice menu, except this menu is presented to the callee.

Settings and options

- Define a greeting message.
- Define the menu items.
- Define the voice message for each menu item.

Text filters/triggers

Description

This service scans through incoming text messages for defined text strings. Upon match the programmer can define to trigger a behaviour.

Settings and options

- Define text strings to search for and the next service to use if match is found.
- Define a next service if no match is found.

Define text strings to search for and the next service to use if match is found.

Roaming handler

This is a simple service that determines if the user is roaming or not and enables defining special behaviour when roaming abroad. The roaming cut-off is much related to the Location Finder 4.1.4 service but as this will be an often used special case of location finder it is practical to have a separate service for this.

Settings and options

- Next service when roaming.
- Next service when not roaming.

Custom Filter

This service is a container for specialized features. This could allow an advanced user to program and define a custom filter for a behaviour. E.g. use the information from a weather service so that you can have one behaviour for good weather and one behaviour for bad weather.

4.1.5. Function adding services

In this section services that enables some end functionality to the end-user will be described.

Voice-mail

Description

As the name of the service states, this is a basic voice-mail. It consists of a voice-message presented to the caller and a voice message recorder. By having it as a service in the designer the user is able to set up the voice mail to his of her preferences and also to set up other behaviour prior or after the voice-mail and hence enabling advanced notification and different treatment of different numbers.

Settings and options

- Upload custom sound file as the voice message.
- Use text message as a parameter from previous services to synthesize a voice message.
- Next service after the voice-mail session has finished no new message.
- Next service when new message in voice-mail.

SMS

Description

This is a simple service that sends a SMS. It can be used to notify the user of some event or in any other situation the user finds it useful.

Settings and options

- Define a custom text message.
- Use a text message as a parameter from a previous services to compose a message.
- Define the next service after the SMS has been sent.

E-mail

Description

This is a simple service that sends a E-mail. It can be used to notify the user of some event or in any other situation the user finds it useful.

Settings and options

- Define a custom text message and subject.
- Use a text message as a parameter from a previous services to compose a message or the subject.
- Next service after the E-mail has been sent.

Collect and postpone

Description

This service collects and postpones messages. All incoming messages will be collected and on a time specified by the user the messages will be sent to the receiving device. To define a starting point for when to start collecting this service needs to be combined with other services. Time filter of

Settings and options

- Set a time for when the messages will be forwarded with the ability to set up recurring events.
- Next service after message has been collected.
- Next service after messages has been forwarded.

Chat

Description

This is a service that triggers a chat session between the subscriber and the caller. The chat session can be on any chat client based on the users preferences and the capability of the device used.

Settings and options

- Which type of chat client to use.
- Chat client specific settings.
- Next service when chat has started.
- Next service if chat initiation fail.
- Next service when chat ends.

Transfer Call

Description

This service gives the user the possibility to set up a transfer of the incoming call.

Settings and options

- Which caller id is shown to the new callee (caller or current callee).
- Next service after transfer is done.

Play voice message to caller

Description

A service that simply plays a sound message to the caller.

Combined with other services

the voice message service can play a message that is defined by the resulting outputs from the preceding services. E. g. by combining the location finder and the voice message, the voice message can describe the location found. The voice message service therefore needs to able to process a text string as a representing the voice message. The text is then translated using a synthesizer to a voice message.

Settings and options

- Define a sound file as the voice message.
- Define a text to be synthesized to a voice-message.
- Define which text string from the preceding services to be used as a sound message.

Queueing

Description

A service for the busy user. This service gives the user the possibility of having and managing calling queues. The incoming calls is placed in this queue and connected to the user in turn.

Settings and options

- Define a waiting sound.
- Set how/if the user should be informed about queue position.
- Set the priority of this queue. This can be done by using the numbers from 0 to 7. Where 0 is the highest priority.
- Next service after queuing.

Follow me

Description

A redirecting service. Redirects the call to one or several other numbers if no answer for a given number of seconds. The call attempts can be done either in sequence or in parallel.

Usage example

A person has several phones. A mobile phone, home phone, work phone. When there is a incoming call to the home phone it rings. The call goes unanswered. After 20s the call is redirected to the work-phone and after 20s more to the mobile phone. Or this can be done in parallel if this is what the user wants.

Settings and options

- Define a list of call attempts.
- Each call attempt can contain one or more numbers to enable parallel calling.
- Define the time-out before going to the next attempt.

Department hunt

Description

Intended to be used at work when there is several people that can answer the same questions. But could also be used in volunteer work or party planning or other situations were a similar situation would arise

Settings and options

- Define list of colleagues.
- Option to enable load sharing.

Custom Effect

This service is a container for specialized features. This could allow an advanced user to program and define a custom effect. E.g. post some text on a web page each time this service is triggered. The idea of this service is that it can take some code in java or php or other widespread programming language for maximum configurability to the user.

Call Handling

This is a group of services that relates to the handling of calls. They do not perform any other task that What is described in the text. They also do not take any parameters or have any settings to configure.

Connect Call this will forward the incoming call to the phone and make it ring.

Disconnect will disconnect the on going call phone.

Reject Call gives the caller the busy signal.

No answer gives the caller the impression that there is no answer.

4.1.6. Logging services

Logging services does not act as the other services. They does not need to be connected to the composition to work. If logging is enabled, all the other service should add a entry to the log describing the occurred event. All of the logging functions can then later scan through the log to fetch the interesting entries and creates views of the log of digests for the different time periods.

Periodic digest

Description

Can produce several different digests: Daily digest which shows a summery of the day, weekly of the week and monthly of the month. All these services has much the same settings and properties. They all have settings for the what to show and when and how to publish the digest.

Settings and options

- What information to be collected and showed.
- Enable and disable buttons different categories of information.
- Define the time that one want the digest to be collected and presented.
- Define how to present the digest, E-mail, Web-page etc.

Logging functions

Description

Defines what to log and how one want the report to be. This is much the same as daily and weekly digests, but is meant to treat all the information from start to for ever.

Settings and options

- What to log, how to display it.

Notification

Description

Defines a threshold of an event that creates an notification.

Settings and options

- The service to monitor
- The event of that service to monitor
- The threshold of to trigger notification

4.2. Use case scenarios

4.2.1. Trekking expedition

Summary

Battery saving while being available on the mobile phone. Created by combining black-list, white-list, signal strength, menu and voice message.

Problem description

Create a composition that enables battery conservation.

Main actor(s)

User and potential callers

Activity scenario

A person is on a trekking expedition. Being away from charging possibilities he wants to conserve the battery as much as possible while still being available on the phone.

Composition scenario

Press button for blacklist service. The dialogue for blacklist appear. The user adds numbers to blacklist if needed. To select a behaviour for blacklisted number, block call is selected from the drop-down list. For all other calls Signal Strength is selected from a drop-down list. In the signal-strength Dialogue there is three drop-down lists: One for low, one for medium and one for high signal strength. In low voice-message is selected, in medium white-list is selected and in high connect is selected. Voice message dialogue appears. A check box stating: use text message from previous service to create voice message. The behaviour after voice message is chosen as Hang Up. In the white-list dialogue a possibility to add numbers to the white-list is presented and drop-down list for both white-listed and non white-listed is presented. In white-listed connect is selected, in non white-listed voice-message is selected. In voice-message dialogue the options available is use white-list message, use signal-strength message and upload custom message. Use signal-strength is selected. The behaviour after voice message is selected as Hang Up. A illustration of the composition can be seen in figure 4.1

Alternative stories

A person is fed up with bad reception and wants to only receive calls when in an area with adequate signal.

Properties

Shows how a battery savings mode composition can be created.

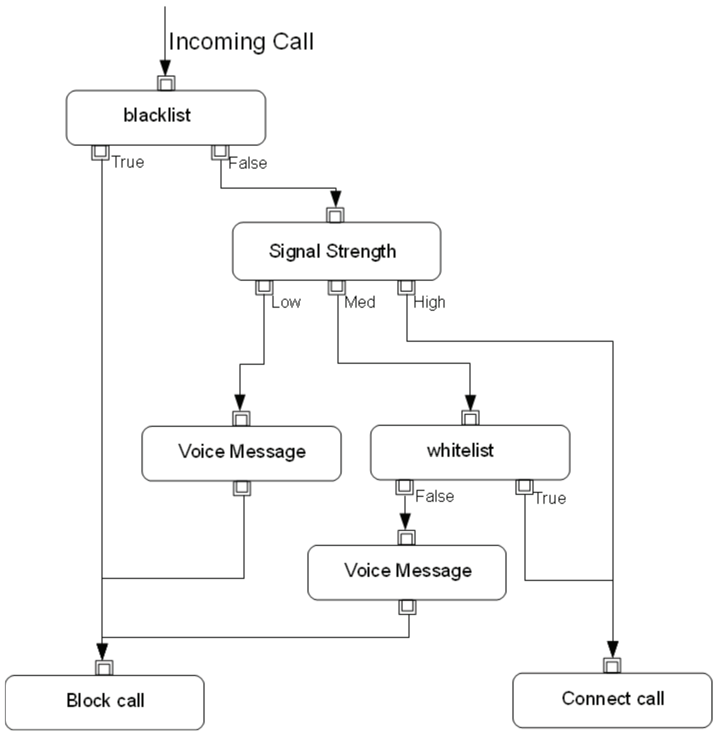

Figure 4.1.: Battery Saving

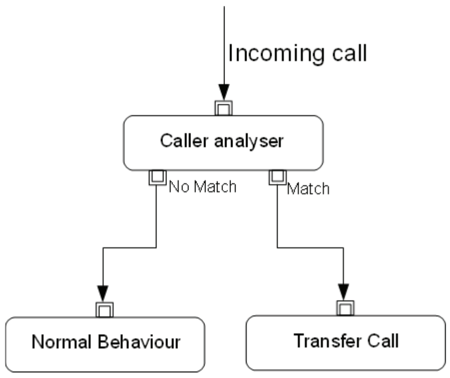

Figure 4.2.: Family Care

4.2.2. Family call

Summary

Call screening and transfer of certain numbers.

Problem description

Creates a composition that transfers calls from certain numbers to other member of family.

Main actor(s)

“Old person”, legal guardian.

Activity scenario

A demented elderly woman is no longer able to organize her financial situation. Her son therefore wants all calls from the bank to go directly to him instead of his mother. He has already got authorization to administer his mothers phone subscription.

Composition scenario

Select to administer other phone subscription, choose the number from list of numbers of which one is authorized to administer. Press button for Caller-Analyser. Dialogue for Caller-Analyser appears. The user chooses to create a new list. The user adds the prefix of the banks number to the list of numbers he wants to transfer. To select behaviour for the matched numbers, transfer call is selected from the drop-down list. In the transfer call dialogue the desired number to transfer to is selected. He start by selecting family call from the predefined compositions. He selects to administer his mothers phone from the list of numbers available. He then adds the number prefix of the bank. A illustration of the composition can be seen in figure 4.2.

Alternative stories

Parent wants to answer some of the incoming calls to their children.

Properties

Shows how some incoming calls can be transferred to a other user

4.2.3. Roaming Cut-off

Summary

Special call screening when roaming

Problem description

Creates a composition that automatically rejects some incoming calls. This

Main actor(s)

subscriber

Activity scenario

A person is on a trip abroad. Due to the added cost of receiving calls when roaming, he wants to automatically filter some of the incoming calls. He already has enabled a black-list that removes the normally annoying calls, but wants to be even more restrictive when abroad.

Composition scenario

Press the button for Roaming Cut off. The Roaming Cut off dialogue appears. The check box by “White-list bypass” is already checked by default so the user leaves that as it was. Behaviour for other callers is chosen by a drop down list of available services. The user chooses voice message from the drop down list. In voice-message dialogue the options available is use roaming message and upload custom message. Use signal-strength is selected. Behaviour after voice message is selected as Hang Up. A illustration of the composition can be seen in figure 4.3.

Properties

Shows how a incoming call filter that activates automatically when going abroad.

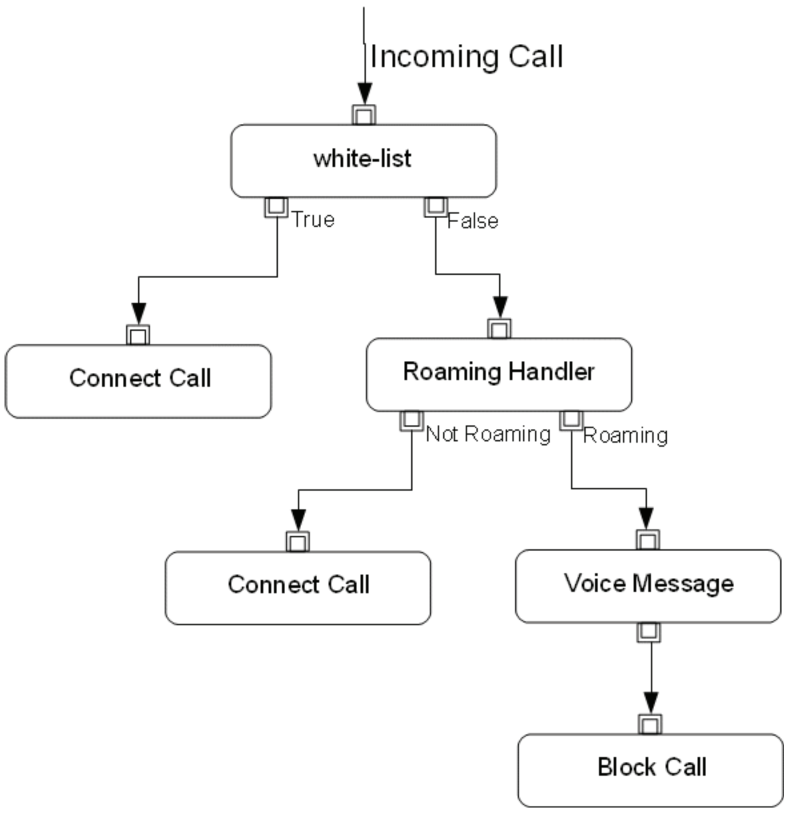

Figure 4.3.: Roaming Cutoff

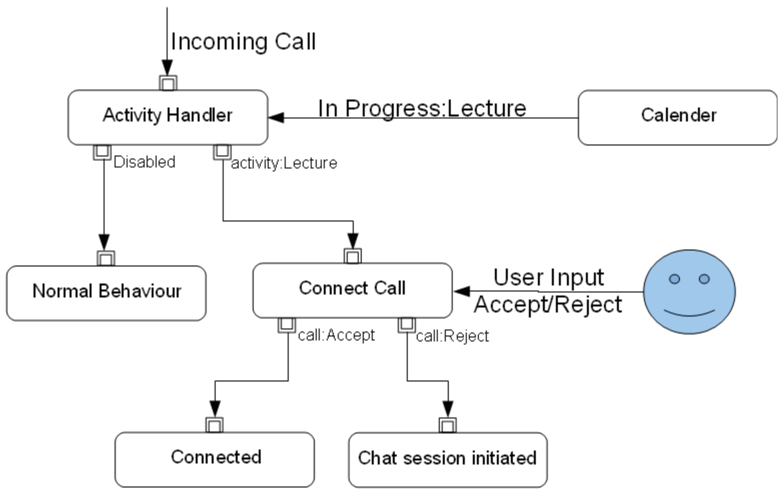

Figure 4.4.: Automatic Chat Initiation

4.2.4. Automatic chat initiation

Summary

A composition that enables semi-automatic chat initiation for some specific callers.

Problem description

Create a composition that enables chat initiation.

Main actor(s)

Subscriber and friends

Activity scenario

A student is in the middle of a lecture. Being in a lecture he can answer the phone, but he wishes to be available and likes using chat clients. He wishes that when any of his “chat” friends calls during a lecture, a chat is initiated between him and the calling party. He chooses to have the call reject act as a trigger. In that way he can decide to answer the phone, let the phone ring unanswered or press reject and start the chat.

Composition scenario

First the lecture schedule has to be entered in the calender in the same manner as in any other calender program. In the composition designer press the button for Activity Handler. In the activity handler one need to find the lecture activity or define it if it has not been defined yet. Next one needs to define the next service for the lecture activity. Here call reject handling is selected. In the call rejected handling dialogue initiate chat is selected as behaviour. The initiate chat dialogue presents the settings and options needed to set up the chat.

Properties

Use of the calender and activity handler to define a behaviour for a planned activity. Use of call reject to trigger a chat session.

4.2.5. Parental Care

Summary

Parents are worried their child shirk school so they use the designer to get notified.

Problem description

Create a service that lets notifies the subscriber if their children shirk school.

Main actor(s)

Parents, Child

Activity scenario

Two parents has a teenager. They have lately grown suspicious of his lack of effort in school and suspects him for shirking school and hanging out down town. They therefore decide to keep an eye on him by tracking their sons mobile phone during school hours. The parents has already gained authorisation from the phone company to track their sons phone.

Composition scenario

Press the button for periodic tasks. This dialogue has settings and options for defining when a trigger is sent. They enter the school hours and an interval of 60 minutes. In the task to trigger they choose location finder. In the location finder dialogue there are three drop down boxes and a location to match setting. One drop down box is for selecting which mobile phone to track and the two other is to set the next service when location match and miss for else. In the mobile they select their sons mobile. For location match sms is selected and for location miss do nothing is selected. They set the location to match as down town Trondheim. In the SMS dialogue they enter the both their numbers and a custom text message. “Our son is down town during school hours”. A illustration of the composition can be seen in figure 4.5

Properties

Uses the a periodic task to trigger a location find and send a SMS on location match.

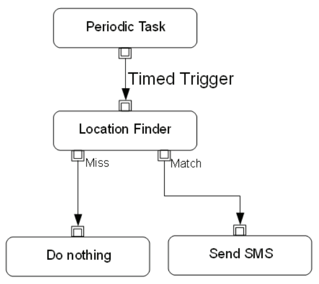

Figure 4.5.: Parental Care

4.2.6. Follow Me

Summary

A person having several phones wants to have her phones ring in a certain pattern, some of then simultaneously.

Problem description

Setting up a behaviour so that several phones will ring in a specified order.

Main actor(s)

Lisa, a person that has several phone subscriptions.

Activity scenario

Lisa has 4 phones. A IP-phone at the office, a work mobile phone, a private mobile phone and a private IP-phone at home. She wants to be available at all times. Having so many phones makes it difficult for friends, co-workers and others to know which phone to call on. So when she receives a call she wants the phones to ring in a specific order when there is no answer on the phone which number is called.

Composition scenario

First she selects the phone subscription to administer. She chooses her private mobile. This defines which number triggers this behaviour, incoming calls on the other phones will behave as normal. Select Follow Me. In the Follow me dialogue there is a time-out value that defines how long before trying a other sett of numbers. There is also a list to which one can add numbers and a time out value before going to the next on the list. She first setts the time out value to zero as she wants her home phone and mobile to ring at the same time. As the first on the list she puts her home phone and her mobile phone and sets the time-out to 25 seconds. Second on the list she puts her ip-phone at work and her work mobile.

Properties

Shows the use of the follow me service

More details

If she wanted the same behaviour on all her phones, then the behaviour has to be defined for all of the phone numbers. The process to do this will be very similar and hence is omitted.

4.2.7. Prioritized Queues

Summary

A busy person needs several queues and want the queues to have different priority.

Problem description

Setting up a composition for queuing incoming phone calls and with some callers getting higher priority.

Main actor(s)

Arne, the subscriber. Calling persons and Calling VIPs.

Activity scenario

Arne is a busy person. He receives a lot of phone calls. The persons calling him is often rejected because Arne is busy talking on the phone and the Call Waiting place is already occupied. For Arne, some persons are more important than others, he therefore want these persons to be prioritized when they call. Arne has already used the designer for some time so he has the blacklist and a queue already working and now wants to add a prioritized queue. Figure 4.6 shows the composition already created by Arne.

Composition scenario

First click on the whitelist service. In the whitelist service dialogue for the "default next service" the blacklist service already applied is selected from the drop down box. For the whitelisted next service queue is selected. In a the queue dialogue values for queue length, queue time out, how to inform user can be set. For the next service when first in line connect is selected. For the next service when kicked out voicemessage is selected. When next is clicked in the queue dialogue it is detected that there is more than one queue that go to the same component (connect call to the user). A queue priority dialogue is presented. In this dialogue it is possible to define which queue has the higher priority and how to distribute the servicing of the queues. A illustration of the final composition with priority queues can be seen in figure 4.7

Properties

Queue priority, adding components to existing composition.

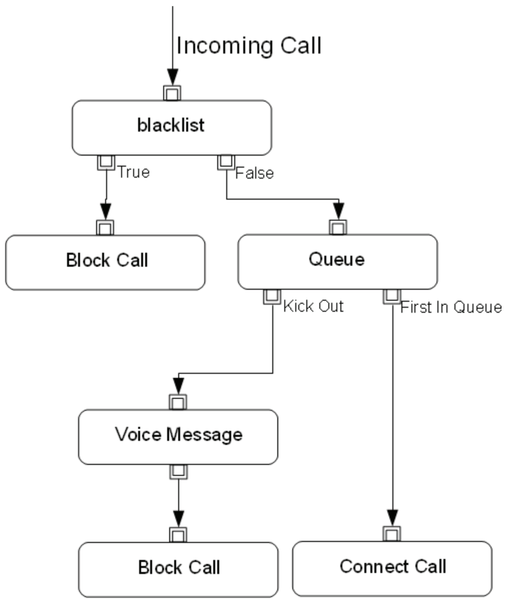

Figure 4.6.: Basic Queue Composition

4.3. Service Compositions

In this section some proposed services composition is defined. This could be used alongside the services as a quick way to add a lot of behaviour. The service compositions can also be used as examples of what is possible to create with the current services. The settings and options when setting up a pre-made services composition will be much the same as with the services, but in addition the user can be presented with the possibility to select which services to use or be guided to which services they want to use by selecting what type of behaviour they want. Each of the services selected should be pre set with usable values, but by selecting advanced options the user should be able to customize the settings of each service as when composing the services manually. For some of these compositions added details for special cases can be found among the usecase scenarios in Section 4.2.

4.3.1. Do not disturb

Description

A service composition that blocks unwanted phone calls, sms, mms and other things that would disturb. Can be used in meetings and other any other situation that has a certain duration where one does not want to be disturbed.

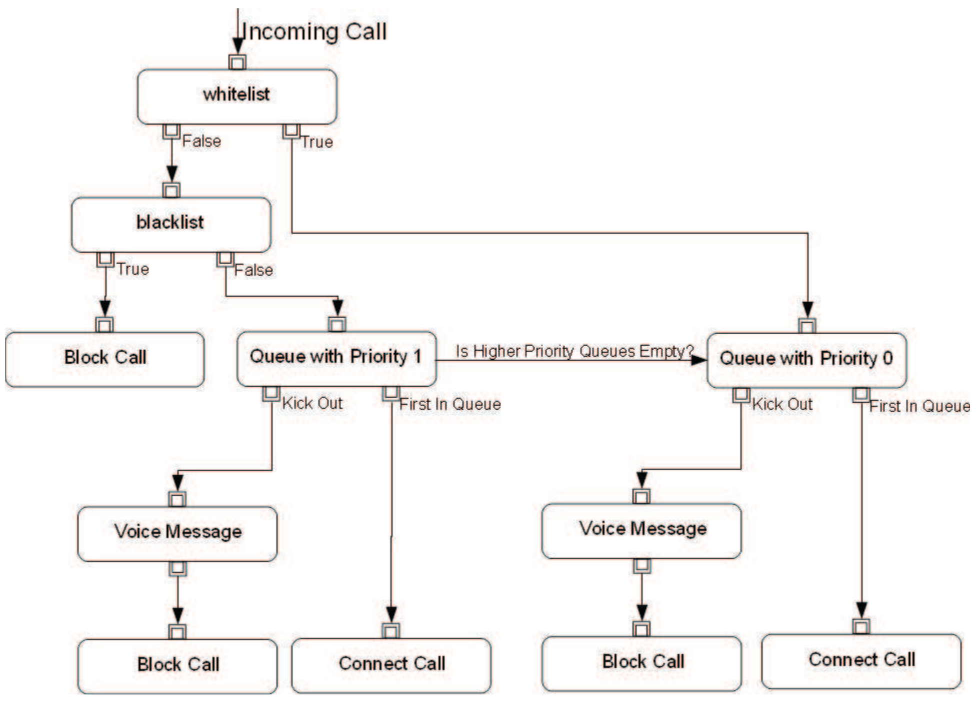

Figure 4.7.: Queues With Priority

Settings and options

The users can select a regular schedule or based on entries in a calender to activate the behaviour. A possibility to enable or disable informing the callers of why they are being blocked and also a possibility to inform the callers of the callee's situation regardless of them being blocked so they can decide not to disturb with unimportant matters. Here properties of the Time-Filter or the Activity-Handler could be used to inform the caller when the user will be available again. Enable or disable the whitelist to bypass the blocking. Option to enable or disable postponing sms.

Services

- Time filter or activity handler to define a starting point, a end and eventually recurrence of the behaviour.
- Voice-Message to inform the callers about why they are being blocked.
- Collect and postpone
- Whitelist to bypass the blocking

Grouping

Compositions, Practical, Telephony

4.3.2. Roaming Cut-Off

Description

When abroad and roaming, the serving distributor adds a additional cost for receiving calls. Because of this some users would like to automatically reject some of the calls. This service composition is meant to handle this. This is done by presenting a option to inform the caller about your situation and the ability to use lists of numbers and or time of day to block or let a call come through.

Settings and options

Option to enable or disable the whitelist bypass. Option to enable periodic digests and voice-mail.

Services

- Roaming Handler
- Voice-Message
- Voice-Mail
- Whitelist
- Daily/Weekly/Monthly Digest

Grouping

Medium, Practical, Location, Telephony

4.3.3. Battery saving mode – Expedition mode

Description

A service composition to reduce the traffic to the phone. This is done by blocking calls, informing the caller about your situation and when you will be available, enabling «opening hours» and collecting information to a daily digest. Intended for use in situations when you have no possibility of charging your phone and are not that interested in «chatting on the phone all the time». More details can be found in 4.2.1.

Settings and options

A list of times, from - to when one plan to be available. Daily digest - enable/disable. Option for white list only. Define behaviour for the different signal levels. Define a voice-message. Setting of how to handle calls that are blocked, send to voice mail, send sms only log for daily digest or combinations.

Services

- Signal Strength
- Periodic Digest
- Blacklist
- Whitelist
- Voice-Mail
- Voice-Message

Grouping

Medium – Advanced, Practical, All

4.3.4. Family call

Description

A service for use in families with small children. The child can call the number of any parent, if number is busy or no answer, the call is transferred to other parent/older siblings etc.

Settings and options

Entering and editing the list of numbers to the parents. Time of no answer that before next number is tried. Possibility to have either parallel calling or calling in sequence. For young kids that just has got their first phone, there is an option to enable that any outgoing call goes to this call-ring Any number dialled (except 911, 112,113,110) goes to the first parent on the list. When calling in parallel, an option to enable a do not worry» SMS to the called numbers that did not answer if a connection of «some duration» is made to any of the other numbers on the list.

Services

- Transfer call
- Hunt Group
- Follow Me

Grouping

Medium, Control, Telephony

4.3.5. Famous person

Description

A service composition that applies a lot of restrictions on the incoming calls. This is intended to be used by public persons or persons that could receive way to many incoming calls and sms. This service would function as a white list only. Only persons on the white list would come through with calls and messages. All incoming traffic that are blocked will be logged and viewable by daily/weekly digest and by logging.

Settings and options

Customizable greeting message. Intended to inform the user about why the call is blocked. Disagree to blocking notification enable/disable. Enabling this will give the blocked caller a possibility to notify that he or she feels that they know this person. Possibility to set the handling of blocked calls send to voice-mail, store voice message on daily digest etc.

Services

- Blacklist
- Whitelist
- Voice-Message
- Periodic Digest
- Caller Choice
- Logging

Grouping

Medium, Control/Practical, All

4.3.6. Parental – block incoming – outgoing

Description

Intended to be used by parents to restrict which numbers to receive calls from and call to. Ability to use predefined lists of suspicious numbers.

Settings and options

Block all incoming, allow special numbers or allow all incoming, block certain numbers Block all outgoing, allow certain numbers or allow all outgoing, block certain numbers Option to define different behaviour for SMS/MMS and for voice calls. Possibility to define custom behaviour when a call is blocked.

Services

- 2 x Blacklist (One for incoming and one for outgoing)
- 2 x Whitelist (One for incoming and one for outgoing)
- Send SMS

Grouping

Medium, Control, Telephony, IMS

4.4. Service Composition Interface

4.4.1. Viewpoint of the proposed designer

When creating a end-user programming interface, it is important that the interface is intuitive. Hence in this thesis the viewpoint is of the incoming call/message and how this call is routed between the services. Each behaviour in the designer has a starting point. This starting point can be any of the triggering services the behaviour is then routed and directed to the end services through the filter services. Some of the end-services can also filter the behaviour of have additional behaviour succeeding it self.

4.4.2. Service representation in the designer

In the designer each service is represented by a icon that illustrates the service. Every service and hence icon has a set of incoming connecting points and outgoing connecting points. How many is defined by the type of service and the settings chosen inside the service. A outgoing connection point is possible to connect to a incoming connection point of a other service. None of the services in the example service set has the need (or possibility) to connect to it self. One way arrows represent the directional connection between the different services. Two services connected by an arrow indicates that the first of those services has the second service as one of its possible next services.

4.4.3. How to create a composition

To create a composition that gives the desired behaviour one first needs to identify which components to use. Ideally this should be as easy as possible. At best one should be able to type a sentence and the appropriate components should be selected for the user and so the user would end up only having to fill out the details. How to achieve that capacity is out of the scope of this thesis. The next best is to have intuitive names on the services and components. The name should in a good way represent the capability and possibilities of that service. The components should also be placed and organized in a intuitive way.

When the user has decided upon a desired behaviour and has found some of the main services needed for that behaviour the programming of that composition can begin. The user first selects the main component or the one that he wishes to be first in the behaviour. When that service is selected a dialogue for that service should open. This dialogue has options and settings for that service and also gives the user the possibility to select which service or component should be executed after the current one. Some service has multiple next services and hence creates branches of behaviour which develops to a behaviour tree. Technically this can be done in many ways but intuitively traversing the tree depth first would be best. When a end point of a branch is reached and either terminating services or no new services is connected to that leaf node in the branch, one would then go back one level and complete that branch. Figure 4.8 shows this.

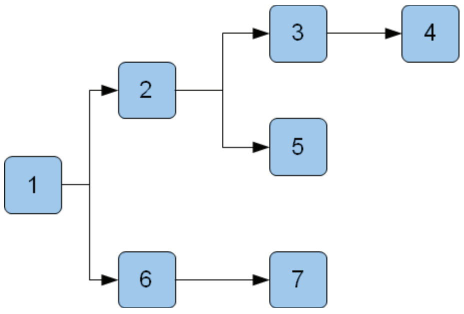

Figure 4.8.: Flow of the composition process

When the first behaviour tree is defined. The user can then later enter into each of the services by clicking on their icons and edit and change the settings. When adding new behaviour, the user can always attach this to the current behaviour by selecting the a new service and in the dialogue of this service select one of the already used services as a succeeding service. Or the user can enter a already used service and change or add a new service to the already used one.

A side result of being able to go into the composition tree afterwards is that the user can create the compositions in a second way: By selecting the services from the list of services and using drag and drop onto the service composing area and then connecting the services with arrows and or entering the services and specifying with service is next. For the user that likes to be in total control, this would be the selected approach. Figure 4.9 shows an example of how the interface could be designed.

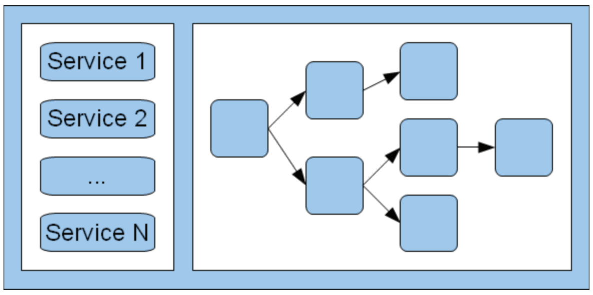

Figure 4.9.: Composition user interface

4.4.4. Composition Logic

Taking a given composition scenario where there exist several services (assuming these services are generic enough, self-contained/atomic, composable, and may be presented to/used by end-user in various compositions), a composition logic is both the structural and behavioural arrangement of these elements (i.e. the service components). In other words, selecting service components, arranging them (for example defining the sequence of invoking or executing them) will give a specific composition logic.

The following rules have been found necessary for the composition logic

- Order of the services determine the order they are executed when deployed. For instance location finder and caller analyser. If caller analyser is first then the result would be find location of user for some callers. If they were the other way around the result a result could be find the location of user and when user is in some locations check the number of the caller.

- Which other services they are connected to impacts the behaviour of the composition, but does not impact the decision making of the service, but can be used to enhance the service.

- Where the service is placed in the designer does not influence the behaviour of either the service or the composition, only how the services are connected to each other.
- Only some of the services produce triggers that initiate a behaviour. Some services route the behaviour, and some produces an end-result visible to the user.

Priority of services

The starting point before any services is deployed is the normal behaviour of any phone or device. When applying the designed behaviour this changes only the minimum of what one could expect. If it is not defined, then the behaviour is as normal. So when applying a service that is active for a given time, at any other time the behaviour is as normal. This can be used to when deciding upon which service has the higher priority. If a service is in the same "conflict group" and they have overlapping filter criteria then the one with the fewest entries is the most specialised one and hence should get the highest priority. But never the less, if there is detected a conflict between two or more services then the user should be informed an allowed to either give the services priority or be able to create a other composition without this conflict.

4.4.5. Composition Constraints

Composition constraints mean that arranging certain service components in certain composition logic would impose certain conditions and thus requirements. For instance using a service component that defines screening for black list and white list would encounter that a number that exists in the white list must not be in the black list if, arguably, the black list to be executed/checked before the white list. This situation does not necessarily impose similar constrain if the sequence of executing these lists (black and white) is done the other way around, i.e. white list first and then black list. Given a list of service components, there must be defined a set of characteristics for each service component that by combining it with another service component would provide all the necessary information regarding any potential constraint. Similarly, consequences of service composition of certain service components may be worked out. What are the consequences of composing service 1 and service 2? What do we need to configure in addition to the set of configurations for each service individually?

Constraints to prevent abuse

The end user programming environment gives the user a lot of possibilities. This could potentially be exploited and used to abuse other people or create unnecessary load on the system. Especially the periodic task service needs some restriction on how often a periodic task can be sent. Take the case were there is no restriction on the interval. If then 1000 users set the interval to 1 second and use this to check the location of their mobile, this would create very much unneeded data traffic and system load. In this application there is no need for intervals of less than 1 minute. Very seldom under 15 minutes as well so a lower limit some where between 1 and 15 minutes should suffice.

Similarly there needs to be some kind of restriction on which devices one could locate with the location finder. The right measures need to be taken here so that one only can find the location of devices that one has properly through the right channels achieved authorisation to locate.

Constraints on reuse of services. Some of the service is designed to be used more than once. But there should be some restrictions on how many of each service one could deploy.

Constraints to the behaviour tree

No loops can bee allowed in any service tree. Allowing loops in a composition will dramatically increase the difficulty of reasoning on the compositions. It will also induce problems concerning performance and system load.

Constraints to prevent conflicts

Some combinations of the different services would create conflicts. This can arise because one could add many different compositions to the same triggering event. This can be omitted if one restrict each triggering service to only be used once. Doing so would prevent the user from defining two compositions that would be running in parallel and causing unpredictable behaviour. One could think that sometimes a parallel behaviour would be desirable, but to argue against this: All, but connecting a call, can be done in a serial pattern and show no external difference from a parallel pattern. The connecting of calls in parallel can be done with the hunt-group service and hence there should be no need for parallel patterns.

To consider the case where the user creates a lot of different behaviour compositions the resulting behaviour tree would then grow very wide. By allowing virtual parallel behaviour this could be remedied. A virtual parallel behaviour would here mean allowing parallel trees where there exists a clear definition of which tree to use. This more closely discussed in 4.4.6.

4.4.6. Dividing the services into groups to reason on them

The services that is described in 4.1 all have some properties that can be used to reason on. Firstly the services can be divided into four groups with regard to how they interact with the other services. These groups are:

- Stand alone services
- Trigger services

- End-Behaviour services
- Filter Services

Stand alone services

Stand alone services is services that will work on their own. What is meant by this is that they do not influence the other services, they never cause conflict with other services and have no dependencies that could cause conflicts. Logging and daily, weekly or monthly digests fit into this category. These services do not have to be a direct part of the behaviour designer. They can be set up and administered with no regard to the rest of the system. However the logging services may benefit of being aware of the active behaviour as it can be logged and provide more details to the log.

Trigger services

Trigger services is services that trigger or start a behaviour. These services either produces a trigger by it self based on timers, or reacts on user interaction and external events. Thus giving the user possibilities to start a behaviour at that event. Periodic tasks, Answer Call and Reject Call are examples of this. These services can not directly create conflicts with each other because the all work as a starting point of a behaviour. If two behaviours were started at the same time this would still create no conflict as two behaviour could run simultaneously.

End-behaviour services

End-behaviour is services that either produce the service that the user experiences or ends a branch in the behaviour tree. Voice-mail, Connect Call, Reject Call and Send SMS are good examples or end behaviour services. Most of these services can operate in parallel or cope with parallel execution in an satisfactory manner and therefore does not need to much consideration with regard to conflict. Even though they are called end-behaviour services it does not mean that they have to stop the behaviour. Other services can still be attached to these services. This will however not impact the

Filtering services

Filtering services is services that filter the behaviour. These services does one or more checks on an value and decides which of the connected services should be used next. They all have at least two possible outcomes. In the designer the can end up being connected only to one other service when the second outcome is block or end behaviour. These are the services that determine which end-behaviour is finally selected. The filtering services can again be divided into several smaller groups. One way to do this is to look at what the services use for filtering. Is it the caller id? Is it the time of day? Is it the location of the callee? Below is a list of the filter criteria used in the the services used in this thesis:

- Time
- Signal
- Activity
- Location
- Caller Number
- Incoming text

These filtering criteria can then be reduced to these groups (see table 4.1). These groups can then be populated by the proposed services in the following manner (see table 4.1)

Group	**Services**
Time	Activity Handler, Time filter
Location	Location finder, Roaming Handler, Signal Strength
Caller Number	Caller Analyser, Black List, Whitelist
Other/Unrelated	Incoming text

Table 4.1.: Filter Criteria Groups

Services that could fit into two or more groups: One could also let services be part of two or more groups with out very severe consequenses. It will however add some additional computations to check for conflicts. It was identified two ways to cope with this. The first is to check all entries of that services in with regard to possible conflicts from all the groups that it is a part of. For some services that is part of several groups it is possible to reduce the number of groups it is part of. Activity Handler is an example of this. An activity can be set as active either based on time or based on location and could hence be in only one of those groups.

The Signal Strength service is placed in the location group because the signal strength is strongly related to position. One can have the medium signal in many places, but if one consider the scenario were the user arrives to his cabin. There the signal strength is low. The user has created two sets of behaviour. One that activates when arriving at the cabin and one that activates when having low signal. This would then create a conflict of the two behaviour compositions that would need to be addressed. Hence the signal strength service should be placed in the location group. Activity Handler is placed in the time group as the activities is related to the calender, which is timebased.

Combining two compositions

When a user has created two compositions or if the user has his behaviour composition and get a other behaviour composition from his employer then there is the issue of combining these. This is not a trivial task and no ultimate solution exists. Consider the case of combining the two compositions Battery Saving (Section 4.2.1 and figure 4.1) and Roaming Cut-off (Section 4.2.3 and figure 4.3). It is many different ways these compositions could be combined. There is several possible way to resolve this. The first is to have the application identify the conflict notify the user about the conflict and let him solve the conflict manually with no help. This would be the easy way out for the application designer. How ever it is possible to aid the user a bit more without over complicating things.

Composition priority table

For each composition a key service should be identified. This services can then serve as an enabler or disabler of the whole behaviour in the composition. If we consider the example described above: Signal Strength and Roaming Handler could be identified as key services. The roaming handler has two possible outcomes: Either one is roaming or one is not roaming. The signal strength services has three possible outcomes. low signal, medium signal and high signal. From this one can create the following table 4.2 which tells what composition to use depending on the different outcomes. This way of combining the compositions work very well for a few compositions, but when the number of compositions grow larger the table grows larger and will soon become unsuitable.

Table 4.2.: Prioritizing Compositions

Outcomes	**Roaming**	**Not Roaming**
Low Signal	Signal Strength	Signal Strength
Med Signal	Roaming Cut-off	Signal Strength
High Signal	Roaming Cut-off	Signal Strength

Proposing composition combinations

When a conflict is detected, the user is presented with a set of possible combinations of the services. The most likely first. Again considering the above example, the following combinations of the compositions could be suggested:

1. Finding the first services in one of the compositions in the other composition. Here we can find whitelist, so the whole roaming cut-off composition could be placed after the whitelist in the battery saving composition.

2. For both compositions show possible combinations by connecting the other composition to the connect call parts of the composition.

3. Try to suggest a service that can be placed above both of the other service composition to bind the compositions together. In the used example here it could be suggested to use the timefilter and other services that is not in use in any of the compositions.

4.4.7. Enabling advanced features

If one just use the services as they are combining them together without passing any parameters one would need a lot of services and often complicated services to get any advanced features. But passing parameters will also complicate the services. A lot of considerations needs to be made. For instance, how many parameters should the a service accept and what parameters should be accepted. One way to simplify this is the use of a billboard. Every service that has been used in the current behaviour branch can post its outputs on the billboard. These outputs can be every thing or anything related to the service. The succeeding services can then later go through this list and select the the parameters needed to achieve the added functionality. Voice-message and voice-mail is services that could be used in this manner. If one consider the location finder service. Every time this service checks the location of a device it posts a text string describing the which device is found on what location. Later in the same behaviour branch the voice-message service is run. The user has defined this service to use the text parameter from the location finder to create the voice-message. The voice-message service the searches the billboard for the text from the location finder and uses it to create the sound message. For this to work some effort has to be done on creating appropriate text strings for all the services that can produce information possibly interesting to the user or caller.

4.5. XML representation of the services

There has been an early decision during the implementation study process of this master thesis proposal to have a scenario driven approach that is generic enough, however linked to a specific service composition platform that is used in Telecommunication networks (an existing execution environment in a real Telecom operator's network). Based on that it was decided to give two levels of abstraction. A Generic level and a platform specific specification, Easy Designer level.

4.5.1. Generic Level

This specification is aimed as a implementation-neutral generic representation of the service components and service compositions and their characteristics independently of the platform that will be used to realize and execute them.

Generic term:

Component

elements is used to represent each service used. A component has the following mandatory attributes: Id of the component, Type of the component. To be able to connect the services to each other every component also needs a attribute telling which component is the next in this behaviour: nextId. This attribute can be omitted if this service is the last in the behaviour branch. Some services depend on external services and therefore could be prone to errors. These services need a nextIdIfError attribute as well. Type is the type of the service, telling which of the services described in 4.1 is being used here. Most of the services has one or more settings that need to be described. These settings can be any thing from a link to a sound-file or a telephone number or phone id.

Hence the a component would look like figure 4.10:

```
<Component id="unique_id" type="example_service" [component specific attributes]>
  Service Specific Content
</Component>
```

Figure 4.10.: Component XML representation

Each service has its own component type. The type of the component dictates which attributes should be used. Below attributes for most of the services has been defined:

For many of the services it is sufficient with this element. But in some cases, especially with the filter services there is a need for list of either numbers, times, locations, etc. These can not be described by attributes in a proper manner. Entries can be used for this. An entry has the following mandatory attributes: Type and Next Component Id. The type tells how to interpret the entry and also dictates which attributes is needed. Figure 4.11 shows a basic entry with what is common for all types of entries.

```
<Entry type="setting_type" [entry specific attributes] nextId="next_component_id"/>
```

Figure 4.11.: Component Entry

XML example of a service: In the example here the voice mail service is used. The voice-mail service has as described in "TODO ref Voice-mail service" the following settings: Voice-message and how to notify about new message. Two versions of the voice-mail services is shown here. The first is a standard voice-mail. The second uses a uploaded file for the voice message and sends notification both to SMS and E-mail.

```
<Component id="0" type="voicemail" voiceMessageType="standard"/>
```

Figure 4.12.: Simple Voicemail example

To show how what a service with entries looks like as a whole an example of a Blacklist service is shown here.

```
<Component id="1" type="voicemail" voiceMessageType="file" fileURL="location of
file" nextId="2"/>
<Component id="2" type="sendsms" textSource="textparam" textparam="1" />
```

Figure 4.13.: Voicemail and notification composition example

```
<Component Id="1" Type="BlackList" NextId="x" ScreenType="A-Number">
  <Entry Type="ScreenEntry" Prefix="12345678" NextId="y">
  <Entry Type="ScreenEntry" Prefix="12345678" NextId="y">
</Component>
```

Figure 4.14.: Blacklist xml example

XML example of a battery saving service compositions

Figure 4.15 shows an xml example of a battery savings service composition. This xml is closely related to the composition created in the usecase scenario in 4.2.1.

```
<Service Name="TrekkingExpedition">
   <Component Name="DailyDigest" Type="Logging" nextId="1">
      <Entry Type="DailyDigest" ReportHour="21" ReportMin="0"/>
   </Component>
   <Component Id="1" Name="BlackList" Type="Screen" nextId="2">
      <Entry Type="ScreenEntry" Prefix="12345678" NextCid="-1">
      <Entry Type="ScreenEntry" Prefix="12345678" NextCid="-1">
   </Component>
   <Component Id="2" Name="SignalStrengthRouter" Type="Router" nextId="5" nex-
tIdLow="3" nextIdMed="4" nextIdHigh="5"/>
    <Component Id="3" Name="BadSignalMessage" Type="Announcement" Brelease-
Call="True"/>
   <Component Id="4" Name="Whitelist" Type="Screen" nextId="3" />
      <Entry Type="ScreenEntry" Prefix="12345678" NextCid="5">
      <Entry Type="ScreenEntry" Prefix="12345678" NextCid="5">
   </Component>
   <Component ID="5" Type="Termination" Number="12345678"/>
</Service>
```

Figure 4.15.: Trekking Expedition (Battery Savings) Composition XML example

4.5.2. Easy Designer Level

Basically this is a specific platform oriented specification, where both the objects and their properties and attributes are tightly coupled to the given platform implementation. This coupling is in terms of supported features, domain-specific properties, execution environment characteristics, etc. In this level a bit more details is needed to fill the requirements.

4.6. Results Summary

The following results have been produced: A number of different example services has been defined with settings and options. Services to respond to event, services to filter behaviour and services that produce some effect observable by the user. These services was sorted and grouped with regard to their properties. From these services a set of use-case scenarios and services compositions were created. In the use-case scenarios a description of how the user would compose the desired behaviour was described. From these use-cases and compositions a set of composition constraints were worked out. Two ways for the user to compose the desired behaviour has been presented. A interface layout and work flow for the composition interface has also been suggested. XML representation of both the services and the service composition was worked out.

5. Discussion

5.1. Accomplishments

In process of developing the simple-to-use infrastructure for end-user service composition, several issues has arrived that needs to be discussed. A set of services has been worked out. These services should cover a quite varied set of functions. The services should be self contained and modular enough to be used in many different compositions. Early in the project there was some discussion of how large (how comprehensive) the services should be. On one side there was large services with lots of functionality and settings and some of the earliest suggestions of services has quite complex behaviour. This would have made it more difficult to reason on the services with regard to conflicts. On the other side there was smaller services, more atomic and confined which ended up being the chosen route. This was done primarily for the sake of configurability for the end-user, as having many small services would give the user more choice. The cost of this being that the composition trees would be larger and more complex. Looking back, some of the services ended up being so small that they could look more like programming blocks than like services. But still it can be argued that they are services based on that they produce the service of e.g. checking for a number match.

For the flow of the application it was chosen to give the user two ways of creating a composition. First the easy more wizard like way, where the user is guided through the dialogues based on the choices made in the preceding dialogue. This was supposed to be a easy compared combining the services manually. However this could arguably be a bit disorienting for the user as the dialogues would some times jump to a other branch. This could be remedied by showing a composition tree alongside the dialogue showing where in the tree the user was currently. The other way of creating a composition was to drag and drop the service onto the designer and connect the services with arrows to define which service is connected to each other. This way of creating services gives the user a better overview.

The use-case scenarios created followed a template from the ubiquity for all project. These scenarios served as a good source for information and gave more insight into what was needed of the service composition interface.

Some solutions for working out conflicts were also worked out. The validity of those needs to be tested, but they should work as a first line of defence against conflicts. Some of the suggestions made could be quite useful for the user when implemented.

No prototype was completely programmed and run. This could partly be because of the chosen approach to create the prototype. Maybe if the spreadsheet approach were chosen, then it probably would be much quicker to create a complete and finished prototype and hence one would be able to run and test the compositions and use-cases created.

5.2. Scope of the thesis

The scope of the thesis was changes from the original thesis text early on. It was deemed that the original text fitted well enough in some areas so it was kept. No new scope was never clearly defined and written down. The thesis went from being composing end-user services to end-user service composing. This made understanding the scope hard as there was no accurate text to refer to. It is possible that this has influenced effectiveness in the start period of the thesis.

The way this thesis turned out to be a more end-user service composition implicates that there should have been some more end-user involvement. A user usability survey could have provided many useful inputs. However through out the period of the thesis my contact network was used to provide and influence what services to make and what type of service compositions that were desirable. Some of the use-case scenarios came directly from inspiration from conversation with friends and family.

There have been many lessons learned, especially with regard to keeping the motivation up and how important it is to always have some progression. Looking back, the biggest error was not having a accurately defined text for the thesis.

5.3. Future Development

For the end user programming interface to be useful it needs some more work in certain areas. The services and application in this thesis is more a proof of concept so to create a useful application there is still a lot of work that needs to be done. The user interface needs to be programmed and polished.

Combining the service handler with a common language interpreter which can from a simple sentence determine which services is most likely to be used. E.g. user states: “I want to enable a queue for incoming calls.” From that sentence it should be possible to determine that it would be smart to start designing the service composition with the queue service or e.g. when if one writes the headword battery, then all services that in some way could impact the power consumption would be displayed.

A user interface for mobile devices should be developed so that the end-users will be able to change the behaviour on the go and or to manually e.g. update their activity to activate a behaviour already developed.

Service-compositions that are commonly used can be integrated in the environment. This enables a one button setup instead of a series of clicks and setting entries. If the application includes a uploading functionality and a “browse/search on-line services“ functionality the ease of use is increased considerably, while the time spent to create the functionality wanted would be greatly reduced. This of course requires quite some users and that they are willing to publish their compositions. The uploaded compositions will of course have to be stripped of numbers and personal settings.

In the dialogue of all the services it is possible to define the next service. Some research could be done on which services is most likely to be used together. This would be best and most accurately done if one were to have a finished product and collected usage data from that. This information could then be used to populate the list of the next services.

A end-user survey should be carried out to ensure that the user interface and service composition approach is understandable by the common user.

6. Conclusion

Through discussion with friends and contact network it has been clear that a service like the one being developed here has been missed and desired. Even though there has been no user survey, much of the work done here has been inspired by suggestions and wishes from potential users.

A interface for end-user composition of telecom services has been worked out through a series of use-cases. The established example services should cover a range of different uses and can be used to create many different service compositions.

The interface created should be able to aid the users in creating a desired behaviour for their telecom services. Giving the user full control and great customizability. The end-users have not yet been taken into the loop so testing and usability surveys still needs to be done.

The chosen approach for creating a prototype could possibly have inhibited the progress on this field and choosing a "quick but ugly" approach could have resulted in some testing and more concrete results.

Personally I see great potential in this application and would very much like to have one as a part of my phone subscription. I hope the work on this application continues and bears fruits.

Bibliography

[B] Joseph Bih, *Service oriented architecture. a new Paradigm to Implement Dynamic E-Business Solutions*, Ubiquity - and ACM IT magazine and forum (2006).

[BOP] J. Yang Bart Orriens and M.P. Papazoglou, *Model driven service composition*, ICSOC (2003).

[CHT] Channabasavaiah, Holley, and Tuggle, *Migrating to a service-oriented architecture*, IBM DeveloperWorks (2003).

[DS] Schreiner W Dustdar S., *A survey on web services composition*, Int. J. Web and Grid Services 1 (2005).

[K] Ken Kahn, *Toontalk - making programming child's play* [online], 2009, Available from: `http://www.toontalk.com/English/infodesk.htm` [cited June 18, 2009].

[LHM] Xuanzhe Liu, Gang Huang, and Hong Mei, *Towards end user service composition*, COMPSAC '07: Proceedings of the 31st Annual International Computer Software and Applications Conference (Washington, DC, USA), IEEE Computer Society, 2007, pp. 676–678.

[M] Alain Pastor Freddy Lécué Eduardo Silva Luís Ferreira Pires- Mariano Belaunde Mazen Shiaa, Paolo Falcarin, *Simplifying automatic service composition - enduser and service developer perspectives*, IEEE TRANSACTIONS ON SERVICES COMPUTING (2008), Yet to be published dec 2008.

[MKK] Leonel Morgado and Journal Ken Kahn, *Towards a specification of the toontalk language*, Journal of Visual Languages and Computing (2007).

[MS] Michael E. Maximilien and Munindar P. Singh, *Toward autonomic web services trust and selection*, ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing (New York, NY, USA), ACM Press, 2004, pp. 212–221, Available from: `http://dx.doi.org/10.1145/1035167.1035198`.

[N] Bonnie A. Nardi, *A small matter of programming perspectives on end user computer*, The MIT Press (1993).

[OG] Zeljko Obrenovic' and Dragan Gasevic, *End-user service computing: Spreadsheets as a service composition tool*, IEEE Transactions on Services Computing **1** (2008), no. 4, 229–242.

[OYP] Bart Orriëns, Jian Yang, and Mike Papazoglou, *A framework for business rule driven web service composition*, 2003, pp. 52–64, Available from: `http://www.springerlink.com/content/mplfeany3ea2uecw`.

[RKM] Jinghai Rao, Peep Küngas, and Mihhail Matskin, *Composition of semantic web services using linear logic theorem proving*, Inf. Syst. **31** (2006), no. 4, 340–360.

[RP] Alexander Repenning and Corrina Perrone, *Programming by analogous examples*, Lieberman 2001a (2001).

[S_1] Smalltalk.org, *Smalltalk.org webpage* [online], 2006, Available from: `http://www.smalltalk.org/` [cited June 18, 2009].

[S_2] Northwoods Software, *Sanscript homepage* [online], 2008, Available from: `http://www.sanscript.net/` [cited June 18, 2009].

[UK] Birgitta Knig-Ries Ulrich Kster, Mirco Stern, *A classification of issues and approaches in automatic service composition*, First International Workshop on Engineering Service Compositions (WESC05), Amsterdam, Netherlands (2005).

A. XML Representation Tables

A.1. XML Services

Table A.1.: Voice-Mail Service Attributes

ComponentType	voicemail
Entries	None
voiceMessageType	[none,standard,file,text,textparam]
fileURL	location of file
text	Text String
textParam	Service Id of the succeeding service providing the text String
nextId	Service Id of the next service after the voice-mail session is finished

Table A.2.: SMS Service Attributes

ComponentType	sendSMS
Entries	None
textSource	[textparam,text]
textparam	Service Id of the succeeding service providing the text String
text	Text String
NextId	Service Id of the next service after the SMS is sent

Table A.3.: E-Mail Service Attributes

ComponentType	sendmail
Entries	None
textSource	[textparam,text]
textParam	Service Id of the succeeding service providing the text String
text	Text String
subjectSource	[textparam,text]
subjectParam	Service Id of the succeeding service providing the text String
subjectText	Text String
emailadress	name@server.com
nextId	Service Id of the next service after the E-mail is sent

Table A.4.: Chat Service Attributes

ComponentType	initchat
Entries	None
chatClientType	[MSN,Skype,etc.]
client specific settings	Client specific settings
nextIdOnExit	Service Id of the next service when the chat client exits
nextIdOnConnect	Service Id of the next service after the chat client is initialized

Table A.5.: Voice-Message Service Attributes

ComponentType	voicemessage
Entries	no
voiceMessageType	[none,file,text,textparam]
fileURL	location of file
text	Text String
textParam	Service Id of the succeeding service providing the text String
nextId	Service Id of the next service after the voice-message is played

Table A.6.: Transfer Call Service Attributes

ComponentType	transfer
Entries	no
number	Number to transfer to
nextId	Service Id of the next service after the call is transferred

Table A.7.: Activity Handler Service Attributes

ComponentType	activity
Entries	activityEntry
nextId	Service Id of the next service when no activities is active

Table A.8.: Periodic Tasks Service Attributes

ComponentType	periodic
Entries	periodicEntry
nextId	Service Id of the next service when no periodic task is active

Table A.9.: Caller Analyser Service Attributes

ComponentType	calleranalyser
Entries	ScreenEntry
nextId	Service Id of the next service when no number match found

Table A.10.: Time Filter Service Attributes

ComponentType	timefilter
HourDayEntry	HourDayEntry
nextId	Service Id of the next service when no time match found

Table A.11.: Roaming Handler Service Attributes

ComponentType	roamingfilter
Entries	no
nextIdRoaming	Service Id of the next service when roaming
nextId	Service Id of the next service when not roaming

Table A.12.: Location Finder Service Attributes

ComponentType	locationfinder
Entries	locationEntry
method	[match,report]
locationToMatch	identifier of the location to match

Table A.13.: Blacklist Service Attributes

ComponentType	blacklist
Entries	NumberEntry
nextIdMatch	Service Id of the next service for blacklisted numbers
nextId	Service Id of the next service for other numbers

Table A.14.: Whitelist Service Attributes

ComponentType	whitelist
Entries	NumberEntry
nextIdMatch	Service Id of the next service for whitelisted numbers
nextId	Service Id of the next service for other numbers

Table A.15.: Signal Strength Service Attributes

ComponentType	signal
Entries	no
nextIdLow	Service Id of the next service when low signal strength
nextIdMed	Service Id of the next service when medium signal strength
nextIdHigh	Service Id of the next service when high signal strength

Table A.16.: Text Filter Service Attributes

ComponentType	textfilter
Entries	textEntry
nextIdMatch	Service Id of the next service when text is matched

Table A.17.: Caller Choice Service Attributes

ComponentType	callerMenu
Entries	menuEntry
reanouncetimeout	Time in seconds before a re-announcement of the menu
timeout	Timeout of the menu
nextId	Service Id of the next service on timeout

Table A.18.: Callee Choice Service Attributes

ComponentType	calleeMenu
Entries	menuEntry
timeout	Timeout of the menu
nextId	Service Id of the next service on timeout

Table A.19.: Follow Me Service Attributes

ComponentType	followme
Entries	huntEntry
nextId	Service Id of the next service after call is connected

Table A.20.: Hunt Group Service Attributes

ComponentType	huntgroup
Entries	numberEntry
enableLoadShare	[True,False]
timeout	Timeout in number of seconds
nextId	Service Id of the next service after call is connected

Table A.21.: Queue Service Attributes

ComponentType	QueueSequenceEntry
Entries	no
MaxPositions	How many positions is available in the queue
fileURL	Location of the sound file with waiting music
QueuePriority	[0,1,2,3,4,5,6,7]
nextIdQueueFull	Service Id of the next service if the queue is full
nextIdYourTurn	Service Id of the next service when call is to be served
nextIdEnterQueue	Service Id of the next service when call enters queue

A.2. XML Service Entries

Table A.22.: HourDayEntry Attributes

DayOfWeek	[0,1,2,3,4,5,6]
Duration	The duration in number of minutes
StartHour	The start hour
StartMin	When to start in minutes past the start hour
NextId	Service Id of the next service it time matches entry

Table A.23.: ScreenEntry Attributes

Number	The number to match
NextId	Service Id of the next service if number matches

Table A.24.: NumberEntry Attributes

Number	The number to match

Table A.25.: ActivityEntry Attributes

ActivityIdentifier	An identifier of the activity that is common with the identifier used in the calender
NextId	Service Id of the next service if this activity is in progress

Table A.26.: PeriodicEntry Attributes

DayOfWeek	[0,1,2,3,4,5,6]
Duration	The duration in number of minutes
StartHour	The start hour
StartMin	When to start in minutes past the start hour
Interval	The interval of the trigger in number of minutes between.
NextId	Service Id of the next service for this periodic task

Table A.27.: textEntry Attributes

TextToMatch	The text string to match
nextId	Service Id of the next service if match found

Table A.28.: menuEntry Attributes

Digits	The digits to select this menu entry
nextId	Service Id of the next service if digits match

Table A.29.: QueueSequenceEntry Attributes

SeqNo	SeqNo="When to present message"
Message	Message="Message to present"
QueueNumber	QueueNumber="True/False"
QueueTime	QueueTime="True/False"