NTNU
Innovation and Creativity

Contex-Aware Call Control

Håkon Vestmoen

Master of Science in Communication Technology
Submission date: June 2006
Supervisor: Rolv Bræk, ITEM
Co-supervisor: Arild Herstad, Telenor R&D

Norwegian University of Science and Technology
Department of Telematics

Problem Description

This master assignment is based on the project assignment "Mobile Absentee Marking - Status Based Call Control" that was carried out at the Department of Telematics, NTNU, fall 2005.

The thesis shall discuss solutions for context-aware call control reflecting the location, the calendar, the availability and preferences of a user assuming the PATS lab infrastructure and ServiceFrame as service platform.

A demonstrator using the Ericsson Network Resource Gateway (NRG) shall be designed, implemented and tested. The Ramses modelling environment shall be used to develop the demonstrator.

Assignment given: 23. January 2006
Supervisor: Rolv Bræk, ITEM

Summary

Today, more and more communication between humans is mediated, where communication is offered as services by applications. The problem with this mediation is that the communication richness is decreased because the communicating humans no longer have knowledge of each others' contex. The solution to this is to make the applications context-aware. When subscribers call each other, the application should be enabled to take the context and preference of the subscribers, in addition to the network and the terminal statuses, into account when deciding where to route the call.

Context was defined to be any information that can be used to characterize the situation of an entity, where an entity is a person, place, or object that is considered relevant to the communication between two users, including the users themselves.

A context-computing demonstrator service that took the context of the subscribers into account when performing routing decisions was implemented and tested using the NRG Simulator.

To be able to compute the context of humans, it is first of all necessary to have a clear and accurate model to describe it with. There are several ways to model context, both in terms of content and the modeling technique that is used. Since context information is temporary, imperfect, highly interrelated and has varying representation formats, ontologies were found to be the best modeling technique. OWL was used to specify the ontology because of its ability to express sophisticated classifications and properties of resources in a formal fashion.

To be able to infer the context of subscribers and deduce what to do, the demonstrator needed an architecture for context management that was responsible for monitoring the modelled and sensed information, perform context reasoning and forward the refined context information to the service logic. The context reasoning was based on five different views resulting in three different (potentially distributed) levels of reasoning: Ontology reasoning, Primary context reasoning and Policy reasoning.

Preface

This master's thesis is written at the end of the study in Master of Technology at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway, and was carried out at the Department of Telematics spring 2006.

I would like to thank my professor Rolv Bræk for all the valuable advices during this semester. I would also like to thank Mazen Malek at the Department of telematics NTNU for helping me out with the work on ontologies and reading through the ontology-related chapters of my thesis. Further on, I'd like to thank Humberto Nicolás Castejón Martínez at the Department of Telematics at NTNU for developing the policy-managing module in ServiceFrame and helping me out when it was included in the demonstrator service. Finally, I'd like to thank my supervisor Arild Herstad at Telenor R&D for reading through my thesis and giving valuable advices during the work on this thesis.

Trondheim, June 11th 2006

Håkon Vestmoen

Contents

1 Introduction

This chapter will give an overall presentation of this thesis related to why it was performed and what it addresses. In addition, a presentation of related research is given so that the reader may relate this master's thesis to other ongoing research projects. Finally, an outline for the rest of this thesis is given.

1.1 Motivation

Humans are quite successful at conveying ideas to each other and reacting appropriately. There are many reasons to this and among the most important ones are the richness of the language they share, the common understanding of how the world works and an implicit understanding of everyday situations. The quality of the communication between humans can further be improved when the communicating entities have knowledge of each other's context. This is possible due to the fact that the contextual information is interpreted (mostly) in the same way by all participating parties.

However, a lot of communication does not take place on a face to face basis anymore. When we want to get hold of somebody instantly, we typically give them a call. This is especially true in a work setting and contacting people is typically done by calling them on their cell phone, alternatively by sending them an SMS or emailing them.

The problem with mediated communication is that it gets difficult for users to utilize the context information of each other. Communication is offered as services by applications owned by service providers. The only context information that is available for use is the context of the application (e.g. network status), no information about the context of the users is available. The applications should be enabled to make use of the users' contextual information to increase the communication richness [4, 1]. This is especially true when this information changes frequently, as is the case for mobile services. When subscribers call each other, contextual information about them should be utilized in addition to the contextual information of the network and terminals to decide where to route the call. In the project assignment "Mobile Absentee Marking - Status Based Call Control" performed at the Department of Telematics fall 2005, a call control demonstrator service was developed using Ericsson's Network Resource Gateway. The routing decision was based on pre-defined status variables that was set by the user. This master's thesis will investigate how to extend this routing decision by taking the contextual information of the users into account, thus making it more flexible.

Although context-aware computing have been researched since 1992 [5], there are still some debate concerning a number of things. First of all, to be able to utilize context information in applications, it is important to have a clear model of it. There have been a number of attempts to model context [45, 36, 43, 44, 38, 41], that differ both in representation (how context is defined and represented) and representation technique (e.g. ontologies, UML). In addition,

context-aware applications need a number of functionalities. These need to be identified and an architecture that supports them is needed [25, 2]. Finally, applications need to be able to perform reasoning on a number of levels [26], based on the context information. Central to these issues is the context information. A clear definition of context is needed and the characteristics of it needs to be identified.

1.2 Objective

This master's thesis shall investigate solutions for context-aware call control reflecting:

1. The location of the user
2. The calendar and agenda of the user
3. The availability and presence of the user
4. The preferences of the user

A demonstrator service shall be designed and implemented using the Ramses modelling environment and simulated assuming the PATS lab infrastructure with Ericsson's NRG Simulator simulating the underlying network and ServiceFrame as service platform.

1.3 Scope

Central to the above-mentioned goals is to investigate:

1. Context information characteristics and definitions
2. Context modelling techniques
3. Principles of context-reasoning

Based on a literature study of context-aware computing, context will be discussed and defined as a term. In addition, context-aware functions will be identified, some chosen architectures for context management will be presented and context-reasoning will be presented and discussed.

A context model, that takes the discussion of context into account, shall be developed. A discussion about modelling techniques will be given where different alternatives will be presented. One will be chosen to develop the context model. In addition, a presentation of different context models that have been developed in other research projects will be given.

An architecture for managing context shall be developed. The architecture shall take the context-model that was developed into account. In addition, it

needs to support the necessary context-aware functions to be able to facilitate context-aware call control.

The demonstrator service shall illustrate context-aware call control by using different context-sensors developed at the ServiceFrame platform. Further on, it shall make use of the architecture that was developed for managing context and implement context reasoning at different levels. Based on the implementation, a discussion of the context reasoning will be given.

The emphasis of this master's thesis will be on the representation of context information and how to facilitate reasoning based on this, and not on how to actually receive or fetch the context information using real sensors.

1.4 Related research

This section will provide a brief introduction to other related research projects.

DAIDALOS

DAIDALOS[1] (Designing Advanced network Interfaces for the Delivery and Administration of Location independent, Optimised personal Services) is a EU Framework Programme 6 Integrated Project. Its overall goal is to seamlessly integrate heterogeneous network technologies that allow network operators and service providers to offer new and profitable services. Among its four technical work packages are context-aware service provisioning (developing an architecture for context-awareness that is flexible, scalable, robust and optimised) and pervasive service enabling (developing a service platform for pervasive services, where emphasis is put on user centered, flexible, and adaptable service management).

SPICE

SPICE[2] (Service Platform for Innovative Communication Environment) is a European IST-FP6 project. Its vision is to design, develop, evaluate and prototype an extendable overlay architecture and framework supporting easy and quick service creation of intelligent and ambient-aware services, cooperation of multiple heterogeneous execution environments and Pan-European seamless delivery of services across operator domains, networks and terminals. Among the work packages defined are Intelligent Service Enablers which aims at providing intelligent service platform solutions for user profile and context information management and for pro-active service adaptation (anticipatory and attentive middleware functionality).

[1] `http://www.ist-daidalos.org/default.htm`
[2] `http://www.ist-spice.org/`

Akogrimo

Akogrimo[3] (Access to Knowledge through the Grid in a mobile World) is funded by the EC under the FP6-IST programme. It is aiming to radically advance the pervasiveness of Grid computing across Europe. Among the features covered by the Akogrimo framework are mobility and context awareness and the network middleware layer of the Akogrimo architecture specifically offers functionality for presence and context management [64].

1.5 Outline

Chapter 2 will provide a discussion of context-aware computing and communication. In addition, contextual information will be characterized and defined and some architectures for managing context will be discussed. Finally, context reasoning will be described.

Chapter 3 will present what ontologies are and what they can be used for. In addition, several ontology languages are described and finally a discussion about other context ontologies that have been developed will be given.

Chapter 4 will present the the requirements and different solutions to the context ontology and the context management architecture. In addition, a short discussion of why ontologies are well-suited for modelling context will be given.

Chapter 5 will present the demonstrator service illustrating context-based call control. This includes requirements specification, architecture presentation, implementation details through MSCs and an evaluation of the implementation.

Chapter 6 will provide a presentation of the simulation of the demonstrator service in the NRG Simulator.

Chapter 7 will present a discussion of the most important parts of this master's thesis. These include how the presence and availability information are realised, how the different levels of reasoning and the user preferences map to each other and finally how the context ontology that was developed could be modified during runtime.

Chapter 8 will conclude this master's thesis by summarizing the achievements and future research possibilities.

Appendix A presents the state machines of the actors and the configuration files that are needed in the demonstrator. **Appendix B** provides a guide on how to simulate (in terms of what needs to be installed and configured) the demonstrator. **Appendix C** will present relevant technologies for the master's thesis and **Appendix D** will present the application platforms on which the demonstrator was developed.

[3] `http://www.akogrimo.org/`

2 Context-aware computing

Mobile devices are getting more and more powerful in terms of processing capacity, memory and functional capabilities, making it possible to run customized applications on these. Java stands out to be the number one application development language, and it has proved to succeed in mobile and distributed computing as well. Together with the increasing bandwidth capacity in mobile networks, it is no longer a utopia to make complex context-aware services. According to [5], context-aware computing was pioneered by researchers at Olivetti Research Ltd. (ORL) in 1992 under the vision of ubiquitous computing, by many also called pervasive computing. The goal of pervasive computing was to enhance computer use by making computers available throughout the physical environment, but making them effectively invisible to the user. Several definitions of context has been proposed, of which some of them resemble while others are quite different. The definitions have developed with time and field of research. For instance, context is in sociology defined as "*an actor's space of action as it is understood with basis in a certain situation's opportunities and limitations*". This chapter will present a discussion about context; how it is and have been defined. In addition, the nature of context information, context-aware functions and context-aware communication will be discussed. Further on, different context types and their formats will be presented, and finally a presentation of how to acquire, manage and perform reasoning on context information will be given.

2.1 Defining context

Most people have an idea of what context is. But when they are asked to define it, few can provide a good answer. Most often, enumerations of examples or synonyms are given instead of a clear definition of the term. However, most would agree that context describes or complements the description of a situation; it has something to do with the environment of an object. Context is defined by [13] as "*That which surrounds, and gives meaning to, something else*". According to [22] context is "*the interrelated conditions in which something exists or occurs*". General definitions of context have been developed since the birth of context-research, and in this section a summary of various definitions will be given.

In [7], context is defined as "*... the set of all entities that influence human (or system's) cognitive behaviour on a particular occasion*". This is a general and wide definition. The definition claims that context is a set containing entities that will influence humans cognitive behaviour (conscious intellectual activity, e.g. thinking, reasoning, remembering, imagining or learning) in a situation. From a technical point of view, this definition demands too much knowledge about cognitive science. In relevance to mobile computing, a more technical definition is needed. Each situation (or occasion) will have to be analysed in terms of what will influence the user's cognitive behaviour, which for technologists is

not that straight forward.

In [8], it is claimed that when talking about mobile distributed computing systems, there are three important aspects of context: where the user is, who the user is with and what resources are nearby. In [6], it is claimed that "*A primary concern of context-awareness in mobile computing is awareness of the physical environment surrounding a user ...*". Location is probably the most used representation of the user's physical environment in implementations demonstrating context-aware services. However, there are many other physical environments that are of interests, especially if we take the context definition of [7] into account.

Other definitions have been proposed extending the earlier definitions above, many of them simply enumerating examples of context (e.g. the user's emotional state, focus of attention, location and orientation, date and time, objects, and people in the user's environment). However, according to [1] these are difficult to apply, since when trying to determine whether a type of information that is not in the definition is context or not, it is not clear how the definition can be used to solve the dilemma. The definition of context needs to be a lot more generic to be applicable to different situations, applications and users.

Many of the early definitions of context also differ in the way the environment is viewed; the user's or the application's environment. In [15], context is defined to be the elements of the user's environment that the user's computer knows about. In [16], it is seen as the situation of the user, without even taking the situation or environment of the application or terminal into account. In [21], the other approach is taken by viewing it as the state of the application's surroundings. Similarly, [20] sees it as the setting of the application. In [18], the definition is taken to a new level by defining it as aspects of the current situation thereby including the entire environment. According to [1], these are all difficult to apply, since they all merely define context by synonyms.

As noted above, [8] claimed that there were 3 important aspects of context when relating it to context-aware services in a distributed environment. Context was defined to be the constantly changing execution environment dividing it into:

- Computing environment (e.g. available processors, devices accessible for user input and display, network capacity, connectivity, and costs of computing)
- User environment (e.g. location, collection of nearby people, and social situation)
- Physical environment (e.g. lighting and noise level)

In [17] context is defined to be the user's physical, social, emotional or informational state. In [19], it is defined to be the subset of physical and conceptual states of interest to a particular entity.

According to the authors of [1], two of the most referred-to researchers within context aware computing, these definitions are good but too specific. They see

context as the whole situation that is relevant to *the application and all of its users*. Their definition of context is as follows:

> "*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*"

The definition above has been well-acknowledged and referred to by other researchers and seems thus to be state-of-the-art. However, it has been criticized by [2] for not providing a way to view the information at a meta-level, that is information about the actors common understanding of situation (e.g. I know that you know that I know...). In addition, according to [5] the definition does not deliver the difference between context as determining the behaviour of mobile applications and context as relevant to the application but not critical. They believe that context, in mobile computing, has these two aspects and the definition of context must take this into account. Their definition of context is as follows:

> *"Context is the set of environmental states and settings that either determines an application's behaviour or in which an application event occurs and is interesting to the user."*

They define the first part as the active context and the second part as the passive context. It is entirely up to how a context class is used in an application that determines whether it is passive or active. They think that this classification of two types of context is valuable when trying to understand the use of context in mobile applications.

Although this is valuable when designing applications and trying to understand the use of context in mobile applications, it does not have to be part of the general definition of context. This divide belongs at a later stage of the application design. It is easier to differ between active and passive contextual information when the purpose of the application is known. Therefore, the definition should be more general and wide since this is the true nature of context.

2.2 Categories of context

Although it is important to provide a general definition of context to relate to, it is perhaps even more important to have a classification of categories of context. In [6], a context feature space is defined, see Figure 1. For each context, a set of features is relevant and for each relevant feature, a range of values is determined (implicitly or explicitly) by the context. It is important to note that (when trying to establish context models) one of the greatest contribution by this context feature space is the hierarchical divide; no matter what kind of contextual information is provided it can be classified as characterizing a human factor or physical environment.

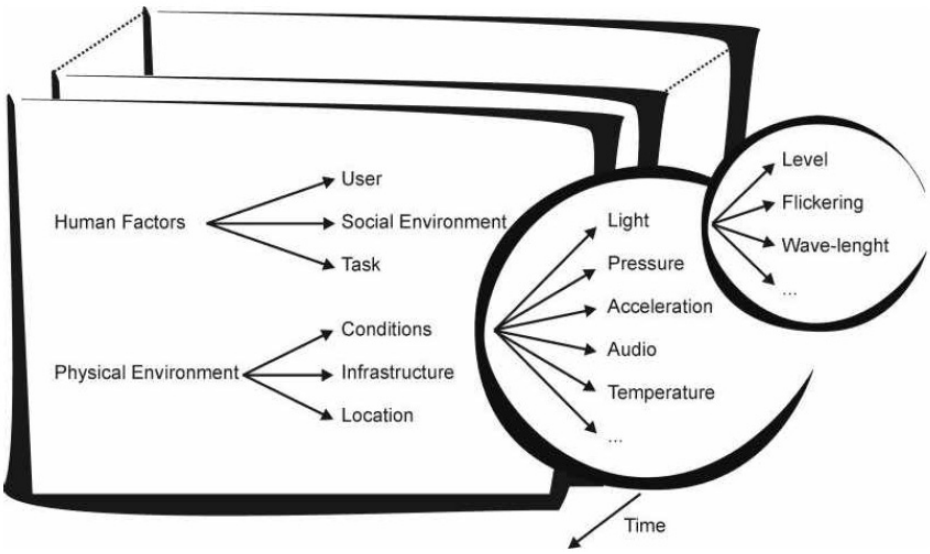

Figure 1: Context feature space, taken from [6]

In [23] location, environment, identity and time are suggested to be the 4 main context types. In general, context-aware applications look at the who's, where's, when's and what's (what the user is doing) of entities and use this information to determine what actions to perform. According to [1], there are certain types of context that are, in practice, more important than others. They are defined as the primary context types for characterizing the situation of a particular entity and are defined as follows:

- Location
- Identity
- Activity

- Time

The primary context types can help finding secondary context, e.g. given the identity of a user his/her email address can be found. The primary context types are considered to be at the first level of contextual information while all other context is considered to be on the second level, thus constituting a two-tiered system of contextual information.

As can be seen, the primary context types defined in [1] are not so different from how [23] defined the different categories of context, the only difference is that environment was replaced by activity. Although primary and secondary context types are not mentioned in the context feature space provided in Figure 1, one might think that the primary context types were considered to be human factors and physical environments. In [1], it is claimed that the primary context type "environment" is just a synonym for context, which does not help application designers to develop context-aware applications nor adds anything to the investigation of context and should therefore be replaced by activity. Similarly, one might think that they would say the same about the two primary context types presented in the context feature space [6], see Figure 1.

The primary context types are not the only ones that can be used in context-aware applications, e.g. temperature and lux conditions can be used in addition to time to determine whether or not the user is outside. Although considerable information is required to establish useful context, [9] points out that it is impossible to represent, much less store, it all. Different applications demand different context types, and so each needs to select the most important ones to use.

2.3 The nature of contextual information

When developing models for representing context, it is important to take the nature of contextual information into account. Below is a short description of each of the characteristics as identified in [25].

2.3.1 Contextual information has temporal characteristics

Contextual information is both static and dynamic, depending on what type of context you're looking at. For instance, the date of birth of a user does not vary while the age of the user will. Since mobile context-aware system's context will change frequently, it is important to note that the focus of context-awareness should not only be on the state of the context but also on the past experience of context to pro-actively predict future tasks. Consequently, part of the context description should be formed by the context history.

2.3.2 Contextual information is imperfect

Because of the frequently changing and dynamic environment of context-aware computing, information can quickly become out of date. This issue scales up

dramatically by the fact that the sources, repositories and consumers of context are distributed, making the time it takes to collect contextual information increase. In addition, processing of the raw contextual data is necessary because of the ill-suited formats (see section 2.8). These problems can produce differences between the present context and the context that is used in the system. In addition, the information provided may be faulty. Finally, part of the contextual information may be missing because of failures or disconnections.

2.3.3 Contextual information has many alternative representations

Different applications demand different requirements of the contextual information. Consequently, a context model must support multiple representations of the same contextual information in different formats and at different levels of abstraction. The model must also be able to catch the relationships between the alternative representations.

2.3.4 Contextual information is highly interrelated

Several relationships are evident between people, their devices and communication channels. For instance, the ownership is a relationship between a user and a device. The contextual information may be derived from these dependencies and relationships. Consequently, the properties of the dependencies are important for the representation of context, since the contextual information is intimately linked with these.

2.4 Context-aware functions

One of the earliest characterization of context-aware applications were given in [8]. Four categories are described with respect to two orthogonal dimensions; whether the task at hand is to obtain information or execute a command and whether the task is effected manually or automatically. This is illustrated in Table 1.

Task	Effected manually	Effected automatically
Obtaining information	*Proximate selection & contextual information*	*Automatic contextual reconfiguration*
Executing a command	*Contextual commands*	*Context-triggered actions*

Table 1: Context-Aware Software Dimensions as defined in [8]

Proximate selection is a user interface technique where the located objects that are nearby are emphasized or otherwise made easier to choose (e.g. when

location is used to emphasize the nearest printer). In the case of context-aware systems, the interesting about automatic contextual reconfiguration is how context of use might bring about different system configurations and what these adaptations are, for example a virtual project white board can become active when the project group is meeting. Contextual information and commands seek to exploit the fact that there are certain things we regularly do in certain situations. Finally, context-triggered actions are simple rules (e.g. IF-THEN) used to specify how context-aware systems should adapt.

In [19], a taxonomy of context-aware functions is proposed. The two approaches differ in that [8] identified classes of context-aware applications while [19] identified the main features of context-aware applications. However, they still do resemble each other. The features are defined as follows:

- Contextual sensing: the ability to detect contextual information and present it to the user, augmenting the user's sensory system (maps to proximate selection)
- Contextual adaptation: the ability to execute or modify a service automatically based on the current context (maps to context-triggered actions)
- Contextual resource discovery: the ability to locate and exploit resources and services that are relevant to the user's context (maps to automatic contextual reconfiguration)
- Contextual augmentation: the ability to associate digital data with the user's context (e.g. reminders with respect to location)

As can be seen, the taxonomy defined in [8] support the presentation of commands relevant to the user's context (contextual commands) which is not supported by the taxonomy defined in [19]. Similarly, contextual information is not supported by the taxonomy defined by [8].

In [1], a taxonomy that combines the ideas from the two taxonomies above is proposed that also takes the major differences into account. It is a list of features that context-aware applications may support that is defined as follows:

- Presentation of information and services to a user (maps to [19]'s contextual sensing and a combination of [8]'s proximate selection and contextual commands)
- Automatic execution of a service (maps to [8]'s context triggered actions and [19]'s contextual adaptation)
- Tagging of context to information for later retrieval (maps to [19]'s contextual augmentation)

Two distinguishing characteristics stand out from this taxonomy: the integration of information and service and the removal of the contextual resource discovery (called automatic contextual reconfiguration in [8]) as a feature. According to [1], it is usually too difficult to distinguish between a presentation of information and a presentation of services, since this is entirely up to how the user actually uses the presentation. If the user simply looks at the presentation, it is merely a presentation of information. However, if the user chooses to use part of the presentation, it is considered to be a presentation of services. Further on, [1] chooses to remove the resource discovery as a feature. They see this as part of the first two categories and not as a separate feature category, since resource discovery is nothing more than locating new services according to the user's context.

[5] has also defined a taxonomy for context-aware applications. Remember that they defined context slightly differently than [1]; they differed between active and passive context. Consequently, they took this into account when defining their taxonomy:

- Active context awareness: an application automatically adapts to discovered context, by changing the application's behaviour
- Passive context awareness: an application presents the new or updated context to an interested user or makes the context persistent for the user to retrieve later

The active context awareness demands more infrastructure and may, according to [5], help to eliminate the unnecessary user cooperation and make technology as ubiquitous and pervasive as possible. As can be seen, [5] has combined [1]'s Presentation of information and services to a user and Tagging of context for later retrieval into Passive context awareness. In addition, [5] has removed the term, and thus divide, between information and service.

2.5 Context-aware Communication

Communication is by [22] defined as "a process by which information is exchanged between individuals through a common system of symbols, signs, or behavior" while interaction is defined as "mutual or reciprocal action or influence".

Communication is more focused on the process of exchanging information: how the information is exchanged, between which entities the information is exchanged and how the information is represented. The information that is exchanged is considered to be the (semantically) same at the receiver and the sender. How the individuals react or are affected is not a part of the communication. Interaction is about the activities being performed and how the involved entities affect each other as a response to the activity that is performed. Communication (at some level) is a prerequisite for interaction. The involved entities act based on what they register that the other part does.

Context aware communication takes place when one or more of the communicating individuals has knowledge of one or more of the other communicating individuals' context, and this knowledge affects the way in which the communication takes place. The individuals of interest could be terminals, humans, agents and/or services.

As is described in section 2.4, there are three different type of functions (services) that can be provided by a context-aware application: presentation of information and services, automatic execution of a service and tagging of context for later use. For this master's thesis, the service that is offered is automatic execution of call control. The application needs to compute the context of a user and perform automatic call control that is based on this whenever two or more users would like to communicate through telephones. The call control could involve deciding whether or not the communication is to take place, choosing which medium the communication shall go through or deciding between which entities the communication shall take place. Since the application never presents any information nor services to the user, the communication is never really motivated by the context of any of the parties (e.g. the caller is not motivated to communicate with the callee because of the context of the callee, nor is the callee motivated by the caller's context). Thus, context-aware communication never really takes place.

The definition of context, as it is defined in [1], is focused on the interaction between users and applications. In context-aware computing, the interaction between the user and application is necessarily important. However, this is not the only important interaction - it can take place between users too. In this master's thesis, the important aspect of the context-aware computing is the communication between two users, where the communication is mediated through the PSTN and/or cellular (GSM) network.

The definition of context that will be used i this master's thesis will be:

"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the communication between two users, including the users themselves."

2.6 Representing context

This section presents a short discussion about the most important context types and how they could be represented.

Location

When working with location information, it is usual to differ between absolute and relative locations, where relative locations are often represented with an address (e.g. Innherredsveien 13) or name (e.g. office) while an absolute location

is unique within a predefined range (e.g. the earth). A geodetical datum [63] is a framework for representing absolute locations and it includes the size, shape and reference points (using a coordinate system) related to a mathematical model of the surface of the earth. Most of the models use an ellipsoide as the model of the surface of the earth. There are many different ellipsoids that represent the surface of the earth, and even more datums (with different ranges) that are based on these. NGO1948 (Norges Geografiske Oppmåling 1948) is a datum that can be used within Norway and ED1950 (European Datum 1950) can be used within Europe. With the development of satellite-technology, a datum that could be used for the entire earth was developed. It is called WGS84 (World Geodetic System 1984) and is supported by both GALILEO[4] and GPS[5].

A position is represented with coordinates (in the coordinate system of the datum) having a longitude, latitude and altitude relative to the ellipsoide. WGS84 supports both. If the position is represented using latitude and longitude, a coordinate can be represented using decimal or degree/minute/second representation, where 1 degree equals 60' (minutes) and 1 minute equals 60" (seconds).

In addition to datums, relative and absolute locations, there are other interesting location-related attributes that exist when performing context-reasoning. An example is the distance to other entities, e.g. the distance to the office.

Time

Time is an important dimension of our everyday lives. Our perception of the world and ourselves is dominated by time, e.g. a big part of the identity of a person used by the society is the date of birth of that person. Our lives are divided into years which are further divided into months. Time can be viewed to be cyclic (and thus repetitive) or linear (something occurs once) in our everyday lives. In addition, time can also be measured in intervals and thus describe the duration of something.

Time is also an important aspect of context computing. This is due to two reasons. The first reason is that context is by nature constantly changing and dynamic. Thus, time is a dimension of the context information. However, this doesn't mean that context information needs to be characterized by time (as in values and units) to be meaningful, e.g. location information is (when isolated) independent of time. The second reason is that people use time to describe their context - if a person were to describe his/her situation, time would definitively be a part of it. Thus time needs to be able to be measured (using values and units) in a context-computing application. Finally, other context information can be depending on time.

[4]The Galileo positioning system is a proposed satellite navigation system, to be built by the European Union (EU) as an alternative to the GPS (which is controlled by the military of the United States) and the Russian GLONASS.

[5]The Global Positioning System, usually called GPS, is (as of 2006) the only fully-functional satellite navigation system in the world

Activity

The activity of a person is by far what most automated context-computing is all about. Trying to find out what the user is supposed to do versus what the user is actually doing. There are thus two different forms of activities - planned (or scheduled) activity and current (or from the application's perspective - deduced) activity. Important attributes when describing an activity are start-time, duration, (relative) location and participants. In addition, category, summary, description and other information could be added. In general, vCalendar and iCalendar are well-acknowledged standard formats for describing activities, and so these should be followed.

Identification

To be able to know which entity a given set of context information applies to, it is important to have a way to identify it. In addition, it is important to identify the profile information of the entity, e.g. a person most probably has a social and/or professional profile. If the entity is a user, he/she may have some preferences (at several levels, e.g. policy-preferences) that are stored in a profile. Finally, a user may also have different roles in different situations. All of this can be deduced based on the identity and additional context information.

Presence and Availability

IETF has specified a Presence Information Data Format (PIDF) [61] for presence information (to be used to exchange presence information) in addition to developing a conceptual model of presence services. The entity who's presence is described is defined as a Presentity, and the PIDF contains the following information: Presentity URL (the current URL of the Presentity), one or more Presence tuples and a Presentity human readable comment. The Presence tuples contain information about the status, communication address (medium and address), a relative priority of this communication address and a time stamp.

3GPP has also defined a conceptual model of presence services [62] where presence information is defined as "... the user's ability and willingness to be reached for communication ...". The Presentity is used (by 3GPP too) to represent the omitting owner of the presence information, and contains a set of access rules that define who can have access to his/her presence information and what part of this information they should have access to (in addition to the presence information). Presence is viewed as a composition of the user status and location of the user.

2.7 Acquiring context

According to [6], context can be acquired either explicitly (by requiring the user to specify it) or implicitly (by monitoring the user and his/her activity). An

example of explicit context acquisition is the specification of current location as required by personal digital assistants to adapt standard location-dependent applications such as clock, tone-dialing and "world". Implicit context acquisition is based on monitoring the entire human-computer system, e.g. monitoring the user interaction to determine whether the device is idle or not.

In [8], a problem with building context-triggered actions is identified; balancing the accuracy with predictable behaviour. When the context changes often, the application designer has to make a choice; let the application display the information (or executing the command) in accordance with the updated context or simply ignore the newly updated context to make the application behave predictably. This problem is also referred to by [3]. In the worst case, the context might change so frequently that the application (assuming that it adapts to the constantly changing context) will be perceived as unstable and unreliable by the user.

There are many sources for contextual information about a certain entity (remember the definition of context in 2.1). As noted in section 2.1, context information can be used immediately or stored for later retrieval (denoted as active and passive context in [5]), depending on the relevancy for the application. Examples of types of sources for contextual information are:

- Sensors (hardware or software)
- Existing information (weather forecast, stock prices etc...)
- User- or task-models
- The state of user devices together with the user interaction with the device
- Explicit user-defined states (e.g. "I am out for lunch")

2.8 Managing context

According to [2], there are two extremes when it comes to managing context; context engine or tight coupling. When dividing the context management from the applications that are going to use the context and thus protect the context in an isolated and autonomous system, we call it a context engine. The other extreme is to let all context that an application needs be an integrated part of the application.

The context engine approach follows the central engineering principle divide and conquer, making it possible to hide details about the collection, interpreting, storing and updating of contextual information from the application developer. In that way, the developer may concentrate on the service logic and simply define what kind of contextual information is needed without concerning about how this information is provided. In addition, letting each application handle all the context managing on its own reduces the reusability and inter-operable capabilities of this functionality.

Below is a presentation of 4 implemented architectures for context management; the Context Toolkit, SOCAM, CoBrA, CoolTown and Stick-e Notes. Due to relevancy issues, the Context Toolkit, SOCAM and CoBrA will be described more detailed than the latter two.

2.8.1 Context Toolkit (1999):

In [24], a layered framework for the developing of context-aware applications is described. An implementation, called *the Context Toolkit,* of the framework was also provided and is presented in [12, 10, 11]. One of the most fundamental goals for the framework was to hide details about the collection, interpretation, storing and collocation of the context for later use by applications. The Context Toolkit can be seen as a complete and autonomous context engine. The framework has been used in the Telenor R&D projects Sm@rt.Travel and Sm@rt.FUNK when developing the context-aware services CAMA and GAID, see [2]. An illustration of the layered framework is shown in Figure 2.

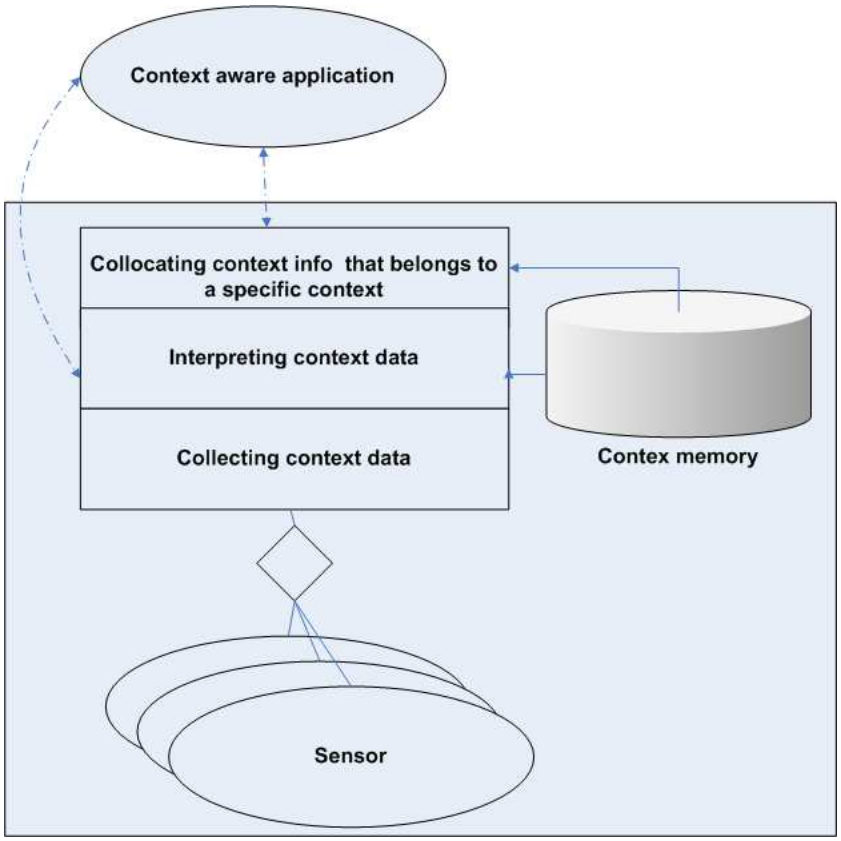

Figure 2: A layered framework for context management, taken from [2].

The layer at the bottom consists of different sensors that automatically registers contextual data from the environment. Above this layer, we find the layer that is responsible for the collection of the contextual data. One might have claimed that the registration and collection could be integrated by making the sensors responsible for both registering and collecting contextual data. However, since sensors often are designed to handle one task only and lacks processing power and memory, this divide is necessary. Since the raw contextual data is typically not suited for presentation in applications and services (for instance the GPS coordinates is often not suited for presentation to an application), it needs to be interpreted (for instance interpret GPS coordinated into an address). Finally, the interpreted data can be stored for later retrieval or used by the application instantly. The application can either poll the interpreted data or a collocated representation of the interpreted data. For instance, some applications may need information from several sensors and so the contextual information needed is a collocation of the interpreted data provided by all the sensors it needs. This can be done by having the application poll the information from the different sensors itself. Alternatively, it can be done by having the application subscribe to the contextual information from a context engine that will determine the terms on which context will be delivered.

2.8.2 SOCAM (2004):

The Service Oriented Context-Aware Middleware (SOCAM) architecture aims to help software engineers design context aware services more efficiently, by dividing the domain of the service into context providers, context interpreters, context aware services and a service locating service [36, 37]. This is illustrated in Figure 3.

The context providers abstract contexts from different sources and can be divided into external and internal ones. The architecture is based on an ontology-based context model (which is presented in section 3.4.3) implemented using OWL, so the context providers further convert the different contexts into OWL representations so that they can be shared and reused by other SOCAM components.

The context interpreter consists of the Context Reasoning Engines, providing reasoning services, and Context KB (Knowledge Base), providing database services. The reasoning services provided by the Context Reasoning Engines include deduced context, resolving context conflicts and maintaining context consistency of Context KB. The database services provided by the Context KB includes adding, deleting, modifying and querying stored context knowledge.

The Service locating service provides the mechanism for how the context interpreters and providers advertise their presences so that users and applications can access and locate their services.

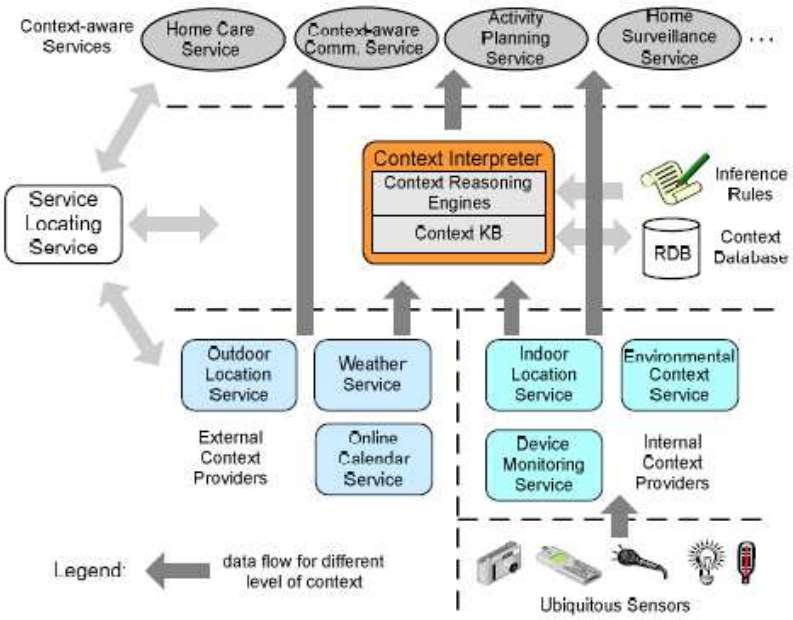

Figure 3: Overview of the SOCAM architecture, taken from [36]

2.8.3 CoBrA (2003):

In [42], an agent based pervasive context-aware computing infrastructure called the Context Broker Architecture (CoBrA) is presented. It includes a context broker that is responsible for accepting context related information from internal (in the same environment) devices and agents in addition to external sources (e.g. semantic web pages, information servers, databases). The context broker integrates and reasons over this information to maintain a coherent model of the space, devices, agents and people in it, and their associated services and activities. To be able to do all this, a set of common ontologies that undergird the communication and representation was developed.

As can be seen in Figure 4, the architecture takes a centralized approach of the broker. This is motivated by the growing demand of context-aware agents and devices that operate on network enabled computing devices (e.g. cell phones, Bluetooth enabled PDAs). Since these devices have very little processing capacity, it is evident that the precessing-demanding operations should be given to a resource-rich server agent. The four functional components of the CoBrA architecture are the Context Knowledge base, Context Reasoning engine, Context Acquisition Module and Privacy Management Module.

The knowledge base defines the ontologies of the intelligent meeting domain and heuristics domain knowledge (e.g. no one person can be physically present in two different meeting rooms at the same time). The reasoning engine is respon-

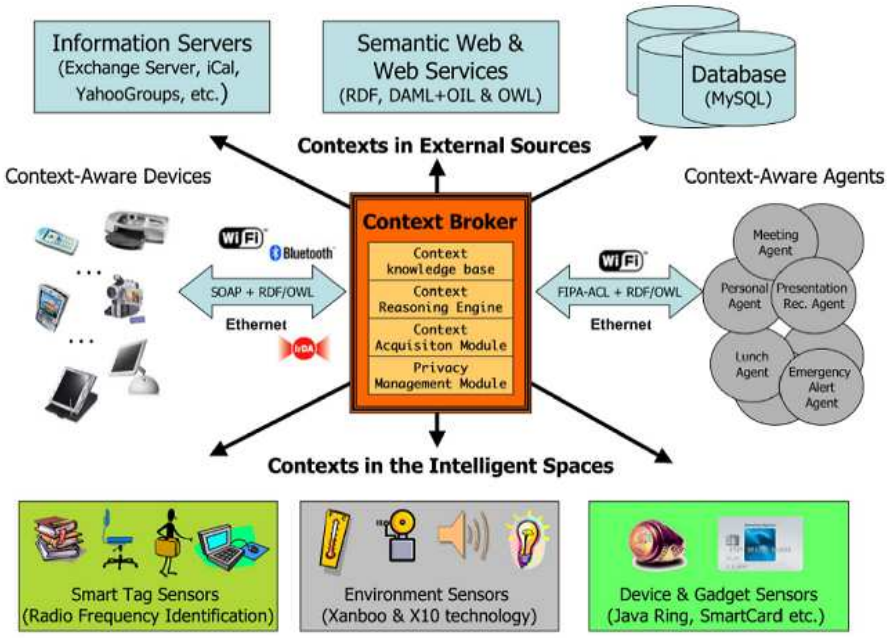

Figure 4: The CoBrA architecture, taken from the CoBrA web site [46]

sible for reasoning with both static and dynamic information by detecting and resolving inconsistent knowledge using domain heuristics and apply learning algorithms and pattern recognition mechanisms to learn about inconsistent user behaviours. The context acquisition module is a collection of pre-defined programming modules for acquiring contextual information from different sources (heterogeneous).

2.8.4 CoolTown (2000):

CoolTown [14] is an infrastructure that supports context-aware services by representing physical objects (people, places, equipment) using web pages. The infrastructure is developed at HP Labs CoolTown-project. Each object is represented as a web site that dynamically updates when new information about the object is found. CoolTown is primarily suited for applications that present context and services to end-users. It is not that well-suited for interpreting and storing sensory (low level) data nor automatically execution of services based on context.

2.8.5 Stick-e Notes (1996):

Stick-e Notes is a general framework for certain context-aware services. The purpose of the framework is to provide non-programmers with a way to define rules for those services. For a given activity, the user can describe, through individual rules (Stick-e Notes), which context elements that will apply, what values they should have and which actions the service should perform based on this. The services are up to the software engineers to develop, they are not specified. How the context is collected, stored and interpreted is not specified and neither is the description of how applications can request context. More information about Stick-e Notes can be found in[15].

2.9 Context reasoning

"The task of using context data in an intelligent way is one of the most challenging contemporary research tasks and is often referred to as context reasoning."[26]

The most important part of a context-aware system is the reasoning, i.e. how the context is deduced based on the raw context data. This is the most important part, because the most severe issue and challenge with context-aware and ubiquitous computing is that the users have to learn to use these systems, too. [9] goes as far as saying that *"Learning to work in a world of increasingly context-aware applications is one of the greatest challenges that we face"* . Consequently, the reasoning in context-aware systems is important, especially when mappings (forming policies) between contextual information and actions are used. According to [26], a more precise definition of context reasoning is *deducing new and relevant information to the use of application(s) and user(s) from the various sources of context-data.* The authors further claim that context is by nature hierarchical, where low-level context is raw data and the higher level contexts are combinations of lower level data sources. For instance, GPS location can be mapped into abstract locations such as OFFICE. Reasoning in context-aware computing can be approached from 4 perspectives and below is a description of each [26].

Low level view

This view is concerned about forming a view of the (user's) current context. This includes: context data pre-processing, sensor data fusion and context inference. It is called the low level view because most of these tasks should be supported either by the hardware (e.g. sensors) itself or by middleware. The most important questions to be answered in this view are:

1. How can the requested context snapshot be formed (the current values of relevant context parameters)?

2. How should data, that is possibly erroneous or missing, coming from multiple sources be dealt with?

The low level view can further be divided into 3 tasks:

1. The *pre-processing of context data* aims to make later processing easier by recognizing the relevant context attributes, handling missing attributes and cleaning the data (e.g. removing non-relevant data). It is in other words concerned about answering the first question.
2. *Sensor data fusion* is concerned about integrating data from multiple sensors (sources) in a reliable way. There has been a lot of research into this, and the prime concern has been to reduce the communication costs by the integration of similar data sources.
3. *Context-inference* is perhaps one of the most challenging tasks. There must be some underlying mechanism that makes it possible to map the lower level context to higher level contexts. One way is to use ontologies in the process and use logic reasoning for the mapping phase. Another approach is to use probabilistic reasoning, where variants of Bayesian networks are used to produce extensible probabilistic models that are then used in the mapping phase.

Application view

This view considers how the application can use the context in an intelligent way, and it is assumed that the raw context data is already obtained. The task of recognizing relevant attributes is also important in this view. In addition, the applications can use a wide variety of reasoning methods to use the context data. The view on this [26], is to consider reasoning components that allow the application (or user) to make agreements (policies) with the underlying systems on how to use the data, e.g. using the user feedback learning methodology *reinforcement learning* (taken from the machine learning community). Originally, the authors don't mention this in the application view, but in the model monitoring. It is without doubt important when updating the model. However, it could be applied with great profit in the application view as well. In reinforcement learning, user feedback does not necessarily have to be provided explicitly (e.g. the user presses yes or no), it can easily be provided implicitly. For instance, if the user is not satisfied by the choice made by the application, he/she will likely overrun this choice and make their own. This can be interpreted as a negative feedback to the application, and based on this it is possible to learn not to make the same choice under the same circumstances again.

Context monitoring

This approach attempts to detect changes in the user's and application's context and to respond to the changes. In addition, it would be favorable to be able

to predict when the context is likely to change (under which conditions) and use the prediction results to pro-actively perform actions, e.g. switching the type of available services for the user. Context monitoring requires sequential prediction methods and [26] suggests Kalman filtering and sequential Monte-Carlo sampling.

Model monitoring

This approach aims at keeping the learned models in a consistent state (in accordance with the monitored context). If some classifier that uses some particular context is learned, it is likely that the context classes will change at some point. These changes need to be recognized by the system, and so the models need to be updated. Decisions made by the system should also be monitored and user feedback should be used to modify the behaviour of the system (e.g. through reinforcement learning).

3 Ontologies

This chapter will present different context ontologies that have been developed in other context-aware applications. Section 3.1 describes what ontologies are and how they are used. Since ontologies have been used and referred to a lot in the semantic web, a short presentation of it will be given in section 3.2. Section 3.3 present some languages that can be used to specify ontologies, and finally section 3.4 presents a language for specifying context ontologies and three context ontologies that have been developed in other research projects.

3.1 About ontology

Ontology is a term that is defined differently, depending on what field of research it is applied. It was first applied in philosophy by Plato and Aristotle who wanted to derive the general structure of the world. Since then, ontology has evolved and been used in various fields of research [65]. It is defined by [13] as "*(From philosophy) An explicit formal specification of how to represent the objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that hold among them*".

When people, organisations, and software systems communicate between and among themselves, there can be widely varying viewpoints and assumptions regarding *what is essentially the same subject of matter* [35]. The different entities collaborate to develop a shared understanding of the areas of interest. The same happens when independently developed software components and sub-systems have to interact with each other. They need a shared understanding of how the area of interest is defined and how it works. If they use different modelling methods, paradigms and languages, the potential for re-use and sharing is severely reduced. Overall, this leads to much wasted effort in re-inventing the wheel.

The above-mentioned problems can, according to [35], be solved by reducing the conceptual and terminological confusion by introducing a framework for shared understanding functioning as the basis for communication and interaction between different entities. There are several benefits to this scheme, of which the most important ones are:

- Inter-operability (among systems): facilitated by translating between different modelling methods, paradigms, languages and software tools
- Re-usability: the formal specification of the system (including its entities, attributes, parameters, processes and their inter relationships) may serve as the reusable software component in the system
- Reliability: the formal specification makes it easier to apply and develop good consistency checking, resulting in more reliable software

The above-mentioned framework is referred to as the ontology of a system, and it entails some sort of world view with respect to a given domain. The world

view is often realized as a set of concepts (entities, attributes, processes), their definitions, and their inter-relationships (often referred to as a conceptualisation). An ontology includes at a minimum a vocabulary of terms and some specification of their meaning (e.g. definitions).

3.2 The Semantic Web

"The semantic web is a web of data"[68]

The problem with the original web is that data is controlled and owned by applications, and each application keeps its data to itself. It is not possible to collocate information from two different services/applications without the services/applications actively collaborating to make it happen or involving a human. The semantic web is about two things; common formats for interchanging data (only interchanging of documents is possible in the original web) and languages for recording how the data relates to real world objects. Put generally, the semantic web is about developing languages for expressing information in a machine-processable form [66] and provide rules for reasoning about data.

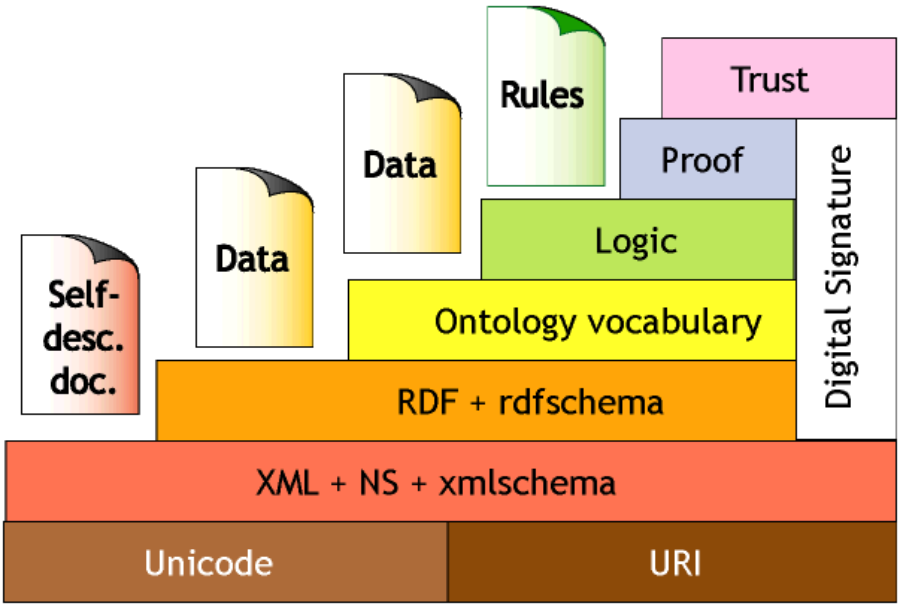

Figure 5: The semantic web stack, obtained from [67].

The architecture of the semantic web will be based on a hierarchy of languages (see Figure 5), where each language exploits the features and extends the capabilities of the layers below it. The URI is the foundation of the semantic web by facilitating global identification of entities and resources. RDF is concerned

with describing resources in documents, while the Ontology layer contains more meta information. After the Ontology layer was instantiated with the Web Ontology Language (OWL), the attention has turned towards specifying logic and rules [69]. The Logic layer is concerned about providing mechanisms for writing logic into documents to allow reasoning that can be validated against rules in the Proof layer, moving the semantic web towards a technology of trust [66].

Although the different languages have been developed for customized purposes, section 3.3 discusses how well-suited the lower-level languages (XML and XML Schemas, RDF and RDF Schemas and OWL) are for specifying ontologies.

3.3 Ontology languages

This section presents some of the languages that are relevant when developing ontologies. A short discussion of how well-suited they are for developing ontologies will be given.

XML and XML Schemas

The eXtensible Markup Language (XML) [55] version 1.0 was published the 10th of February 1998. It is the de-facto knowledge representation language used to describe metadata. XML provides mechanisms for structuring documents (into a root-element, children-elements and attributes of the elements), but has no mechanism for restricting the semantics of the documents. To restrict the structure of an XML document, the XML schemas [56] are used. Although it would be possible to define an ontology using the XML schema, XML and XML schemas are not really optimized for ontologies. There are several reasons to this, of which the most important ones are that XML and XML schemas don't include functionality to represent the inter-relations between different concepts nor provide the semantically meaning to the document that is needed if it contains an ontology. To be able to do this, the developer would have to define these functionalities on his/her own. Thus, new languages based on XML were developed that included more functionality to represent these things.

RDF and RDF Schemas

The Resource Description Framework (RDF) [58] was released as a W3C recommendation in February 1999. It is built on top of XML and is intended for describing resources and the relations that exist between them. In addition, it provides semantics for these concepts making it possible to reason on the information model contained in the document. RDF is better suited than XML to create ontologies, because it has more built-in functionality for describing the semantics of concepts and the interrelations between them in a formal fashion. The RDF Schema [57] provides a vocabulary for describing classes of resources and the properties that exists between them in addition to providing semantics of these hierarchies of classes and properties.

OWL

The Web Ontology Language (OWL) [52] was released as a proposed W3C recommendation in December 2003. It is built on top of RDF and is the successor of DAML+OIL, another language for creating ontologies. OWL deals with the same issues as DAML+OIL: to express far more sophisticated classifications and properties of resources than RDFS. It adds more vocabulary for describing properties and classes. OWL has been used in many context-aware computing projects (CoOL, CoDAMoS, CoBrA, SOCAM - presented in section 3.4) and has been reported to be well-suited for creating context ontologies. This is the main reason as to why OWL was chosen as the language for developing the context ontology in this master's thesis. For a more detailed presentation of OWL, please refer to Appendix D.3.

WSMO and WSML

The Web Service Modeling Ontology (WSMO) [59] was published as a W3C Member Submission 3 June 2005, and aims at describing all relevant aspects related to general services which are accessible through a Web service interface. It is a part of the Web Service Modeling Framework (WSMF) and the goal is to automate (fully or partial) the tasks (e.g. discovery) involved in both intra- and inter-enterprise integration of Web services. To achieve this, WSMO provides a meta-model for semantic web services related aspects, where the top-level elements are Ontologies, Web services, Goals, and Mediators. Ontologies can be described using the Web Service Modeling Language (WSML), a language designed to write, store and communicate ontologies. WSML was specifically designed to express semantic descriptions that are in accordance with the meta-model, however it can also be used to express general ontologies since it is able to specify concepts and their inter-related structure. In addition to axioms and the regular relations, there is a special case called a Function that has a unary range and a n-ary domain, where the range specifies the return value. WSMO and WSML are more targeted towards Web services than specifying general ontologies and will therefore not be used in this master's thesis.

SWSO AND SWSL

The Semantic Web Services Framework [60] contains the Semantic Web Services Ontology (SWSO) and the Semantic Web Services Language (SWSL) and was published as a W3C Member Submission 9 September 2005. SWSO provides a conceptual model in which Web Services can be described and a formal characterization of that model. SWSL is used to specify Web service concepts and create descriptions of individual services in a formal fashion. It is a general-purpose language (its features are not service-specific), but is designed to support the needs of Semantic Web Services. WSMO has focused more on the meta-model and OWL compatibility than SWSO. Although SWSL focuses more on extending the functionality of the rule language than WSML, and is thus better suited

as a general-ontology description language, it is still targeted at semantic web services. In addition, it hasn't been reported to be used much.

3.4 Context ontologies

Context ontologies can be specified using different languages. This section presents a language for developing context ontologies followed by a presentation of a general ontology for ubiquitous and pervasive computing. Finally, 2 context ontologies are presented. The main features of each ontology will be presented, followed by a short evaluation of it.

3.4.1 CoOL - a Context Ontology Language

This presentation of CoOL is based on the contents of [38, 39]. CoOL is a context ontology language that is derived from the *Aspect-Scale-Context* (ASC) model ([39, 38]) and it can be used to enable context-awareness and contextual interoperability during service discovery and execution in a distributed system architecture.

Context information is defined as "*any information which can be used to characterize the state of an entity concerning a specific aspect*", where an *entity* is a person, place or in general an object. This definition of context resembles the definitions provided in [4, 6, 8] a lot, however by introducing the term *aspect,* it refines the expressiveness of the view on context.

The ASC model is named after the core concepts of the model, which are aspect, scale and context information, see Figure 6. An aspect is a set of one or more related scales, it is a *dimension* of the situation space being used as a collective term for information objects having the same semantic type. A scale is an unordered set of objects defining the *range* of valid context information. Thus, a valid context information with respect to an aspect is one of the elements of the aspect's scales. For instance, the aspect "GeographicCoordinateAspect" may have two scales, "WGS84Scale" and "GaussKruegerScale", and a valid context information may be an object instance of one of these (e.g. valid_context_information = new GaussKruegerCoordinate(x, y)).

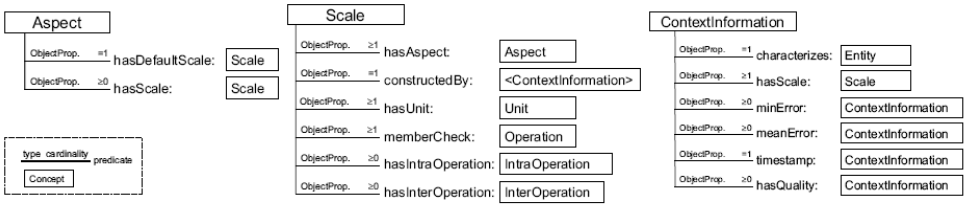

Figure 6: The Aspect-Scale-Context model, from [38]

Although CoOL is a language for describing context ontologies (and one might therefore claim that it is not an ontology itself), it is built on a general, high level context ontology, namely the ASC model. Due to the purpose of the ASC model (provide a basis for a context ontology language), it is the context model that supports the characteristics of context information (see section 2.3) the best. Although several aspects and scales have been defined [39], it is still a bit too general and vague to serve as a context ontology for applications. The context ontology shall serve as the context model (the learned context), which will be the basis for the reasoning. The ASC model is too abstract and general for this purpose, instead a *new* model (ontology) that is based on the ASC model can be developed (which is what CoOL is made for).

3.4.2 SOUPA - Standard Ontology for Ubiquitous and Pervasive Computing

In [45], a shared ontology for pervasive computing called Standard Ontology for Ubiquitous and Pervasive Computing (SOUPA) is presented, and is specified using OWL.

As can be seen in Figure 7, SOUPA consists of two ontologies; the SOUPA Core and the SOUPA Extension. They are distinct, but related. The core ontologies define general vocabularies that are common for different pervasive computing applications, while the extension ontologies extend from the core ontologies and define additional vocabularies for supporting specific applications.

SOUPA Core provides vocabularies for expressing concepts about:

- Person: contact information, social profile, professional profile
- Agent: properties representing what the agent believes, desires and intends
- Belief-Desire-Intention: fact, desire and intention classes that is used in the agent document[6]
- Action: vocabularies for describing the entity that performs the action (actor), the entity that receives the effect of the action (recipient), the object that the action applies to (target), the location at where the action is performed (location), the time at which the action is performed (time), the thing that the actor uses to perform the action (instrument)
- Policy: security and privacy issues related to actions and reasoning functionality about this
- Time: time, time-interval and temporal properties
- Space: symbolic representation of space and spatial relations

[6]When ontologies are expressed in OWL, they are usually placed on the web servers as web documents, which can be referenced by other ontologies and downloaded by applications

- Geo-Measurement: longitude, latitude, altitude, distance, surface area
- Event: what, where, when

For a reference to the SOUPA Extension, please refer to [45].

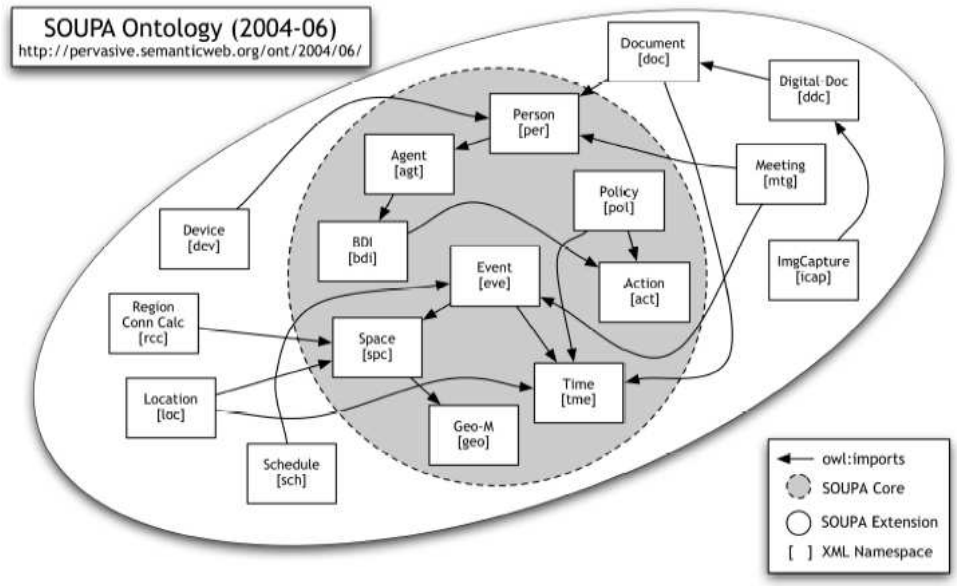

Figure 7: The SOUPA ontology, taken from [45].

Although the SOUPA ontology is not customized for context, it can definitively serve as the basis for developing one. In the agent-based pervasive context-aware computing infrastructure called the Context Broker Architecture (CoBrA) [42], it is used as the super-ontology for context knowledge sharing. CoBrA has defined its own context ontology called the COBRA-ONT, targeted at modelling contexts in smart meeting rooms. In addition SOUPA has been used in MoGATU, a framework for handling pro-active peer-to-peer semantic data management in a pervasive computing environment.

The SOUPA ontology has a very flat structure divided in two layers: the core and the extension. Time, Action, Person, Agent, BDI and Space stand out to be the most important ones, since they all are in the core and are the ones that have imported fewest and is imported by most other documents. One might look at these as the primary context types of the ontology. The ontology is also good at taking onto account that contextual information is highly interrelated and has many alternative representations. This is achieved through the divide between the core and the extension, in addition to exploiting the import functionality of OWL. In addition, the temporal characteristics of contextual information is

taken into account by importing the Time document into Event and Location. However, there is no quality property that can help in setting the reliability of the contextual information (remember that contextual information is imperfect, see section 2.3).

3.4.3 SOCAM

In [36, 37], the Service-Oriented Context-Aware Middleware (SOCAM) architecture, which specifies a context ontology using OWL. The context definition that is used is equal to the one defined by [1].

The context ontology models the basic concepts of person, location, computational entity and activity and how these concepts are interrelated. Different contexts are classified and relationships are modelled using dependency tags on the classes' properties. The sensed context is also annotated with quality constraints capturing the quality of the context.

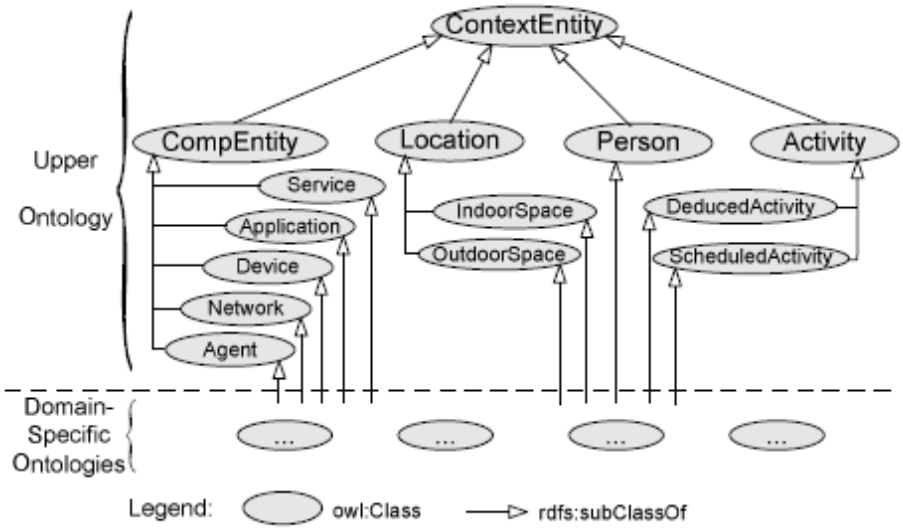

Figure 8: Class hierarchy diagram for the SOCAM context ontology, taken from [36]

Specifying the context in one domain in which a specific range of context is of interest is referred to as *the separation of domain*. As can be seen in Figure 8,

the context ontology is divided into upper ontology and domain specific ontologies. The upper ontology is a high level ontology, capturing general context knowledge about the physical world, whereas the domain specific ontologies are a collection of low-level ontologies which define the details of general concepts and their properties in each sub-domain. With this scheme, the low-level ontologies can be plugged into and unplugged from the upper ontology when the environment changes, making it possible to represent a great variety of context information.

The *ContextEntity* provides an entry point of reference for declaring the upper ontology, where one instance of *ContextEntity* exists for each distinct user, agent or service. Each instance of ContextEntity presents a set of descendant classes of Person, Location, Computational Entity and Activity, which are all detailed in the domain specific ontologies. They have defined all the descendant classes of these basic classes in a smart home environment with a set of properties and relationships that are associated with these classes.

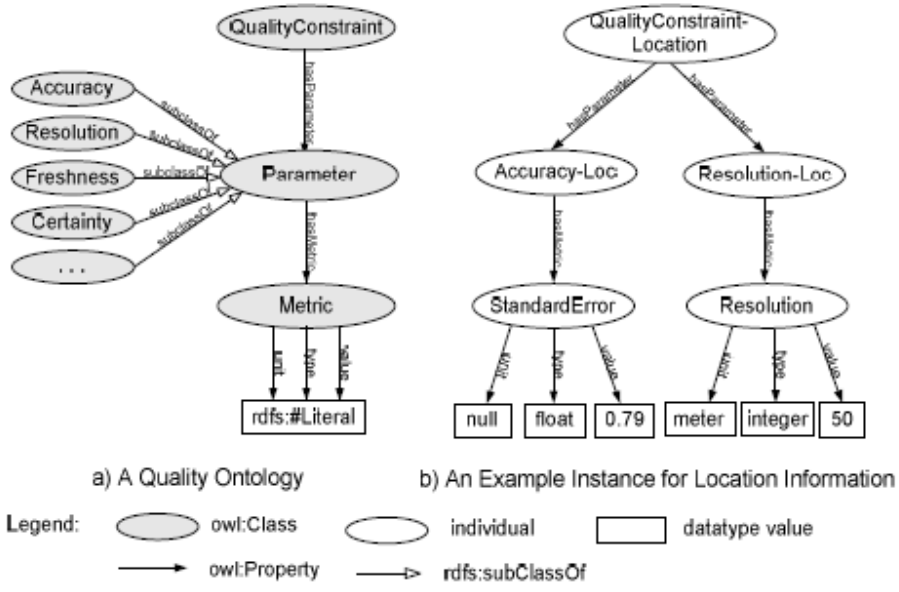

Figure 9: Quality Constraint in the SOCAM ontology, taken from [36]

To set the quality of the context, the OWL properties of entities can be associated with quality constraints. As can be seen from Figure 9a), quality constraints are associated with a number of quality parameters that capture the relevant quality aspects of the attributes of entities and relationships

between them. The parameters are in turn made up of one or more quality metrics, which define how to measure or compute context quality with respect to the parameter. In addition to value, the metric has a type and a unit. The SOCAM ontology has defined 4 types of quality parameters (see Figure 9a)); accuracy, resolution, certainty and freshness. An example is shown in Figure 9b).

The greatest advantage of the SOCAM ontology is that it facilitates *"the sharing of common understanding of the structure of context information among users, devices and services to enable semantic interoperability"* [36]. This is achieved through abstractions when separating the domains. Contextual information has many alternative representations, and so any model of context must support this fact. The SOCAM ontology supports this, through multiple levels of abstraction making it possible to represent the same contextual information in various formats for different applications. In addition, the ontology models the interrelationships and dependencies between different contextual information through OWL properties (using the tag *-rdfs:dependsOn*). The ontology also provides mechanisms to model the temporal and imperfect characteristics of contextual information using the quality constraints, see Figure 9.

The primary context types in the SOCAM ontology are computational entity, location, person and activity. Time is not mentioned, which is (according to [1, 23]) a significant characteristic of a situation of an entity. In other words, time should be, if not a primary context type, a part of the ontology. In addition, user preferences are not part of the ontology. Although these could be a part of the service descriptions, they could with great success be modelled in the ontology as well. This is because user preferences could include not only service preferences, but context reasoning preferences too.

3.4.4 CoDAMoS

In [40], an adaptable and extensible context ontology for creating context-aware computing infrastructures is presented. The ontology is developed for use in Ambient Intelligence, where devices will communicate and interact independently without immediate user interaction making decisions based on a variety of factors (including user preferences and the presence of other users).

Four main entities (denoted as primary context types by[1]) were considered to be the most important aspects in context information, and so the ontology is built around these (see Figure 10). The main entities are User, Service, Platform and Environment and below is a description of each.

The user (illustrated in Figure 11) is perhaps one of the most important entities. The application or service should adapt to the user and not vice versa. Important properties include a user's profile, preferences, mood and current activity. An important thing to note is the distinction between a user's preference (e.g.

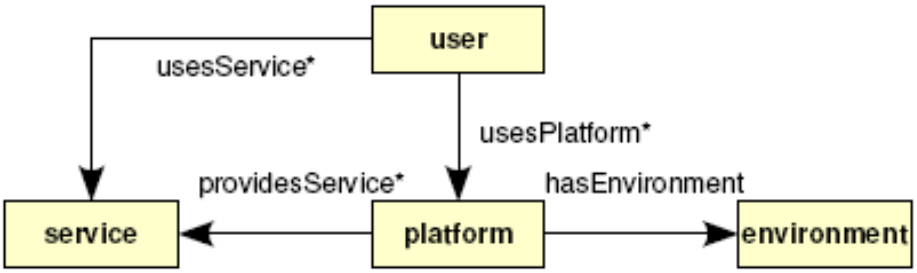

Figure 10: The CoDAMoS overall ontology, taken from [40]

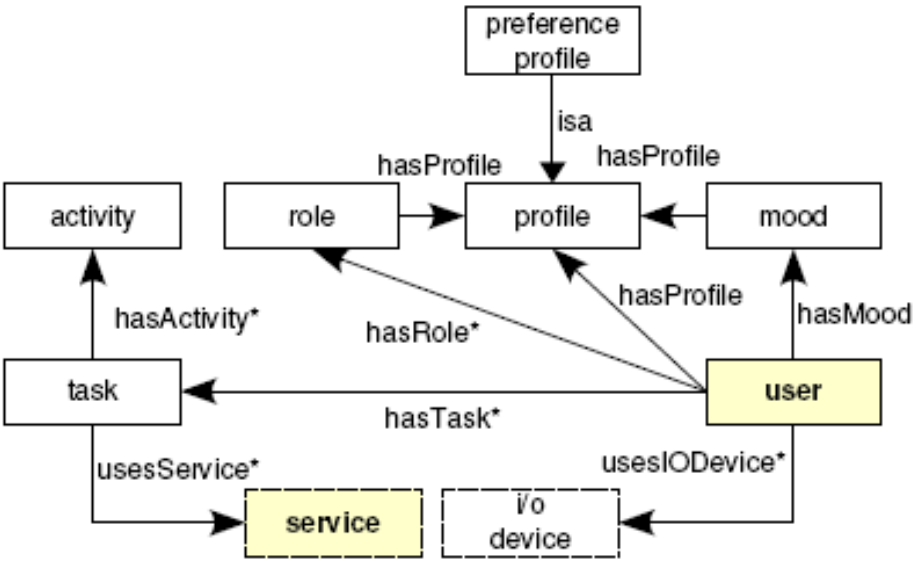

Figure 11: The User ontology, taken from [40]

preference in font type) and profile (containing facts such as gender, name, date of birth). The profile is more or less static while the preference profile is more subject to the current situation.

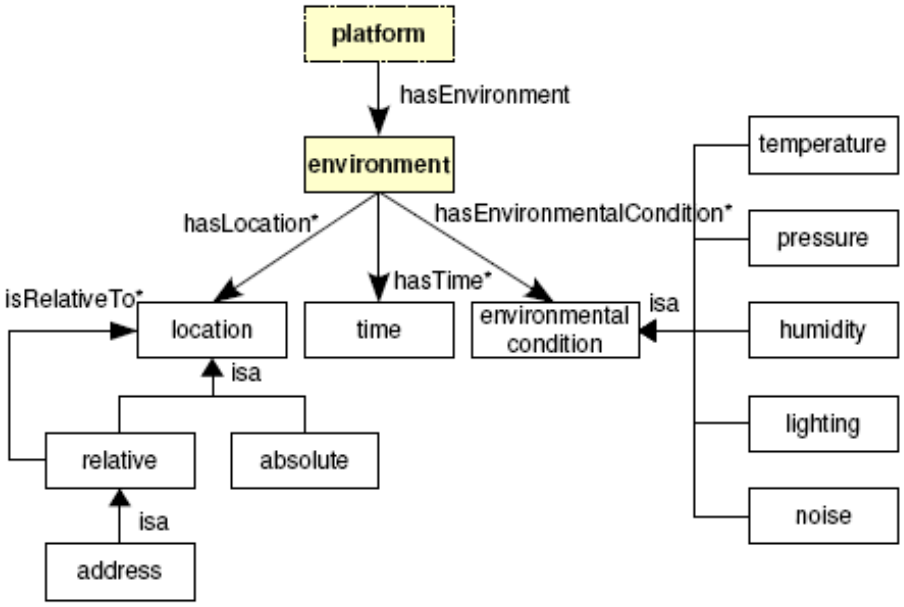

Figure 12: The Environment ontology, taken from [40]

The environment (illustrated in Figure 12) is where the user interacts, consisting of time, location information and environmental (physical) conditions such as temperature and lighting. It is important to note that the environment is always sensed through a platform (device), which is why the user is not directly associated with the environment. Remember that contextual information is both temporal and imperfect. In the CoDAMoS ontology, this is only taken into account when the information is about the environment by delegating the responsibility to the devices that sense this information (denoted as the Low level view, see section 2.9).

The platform (illustrated in Figure 13) is the description of the hardware which specifies the resources of the device and software that is available on the device for the user or other services to interact with. Because certain hardware or software elements in devices can vary or be temporal, only the most relevant entries of the context is specified.

The service (illustrated in Figure 14) provides specific functionality or information to the user. A well-specified semantic and syntactic service interface (or description) sustains easy service discovery and service interaction. Each device is responsible for having a full description of its own services, including how it can be interfaced by other services. The service profile provides a human read-

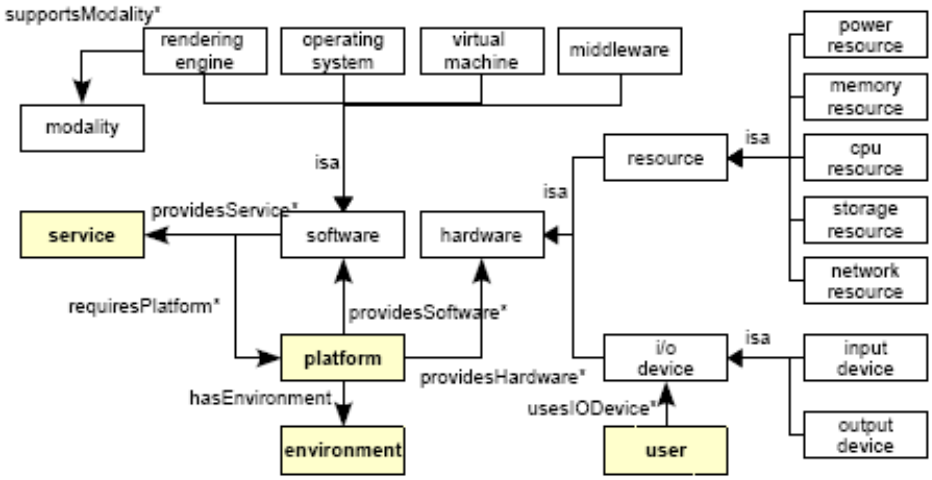

Figure 13: The Platform ontology, taken from [40]

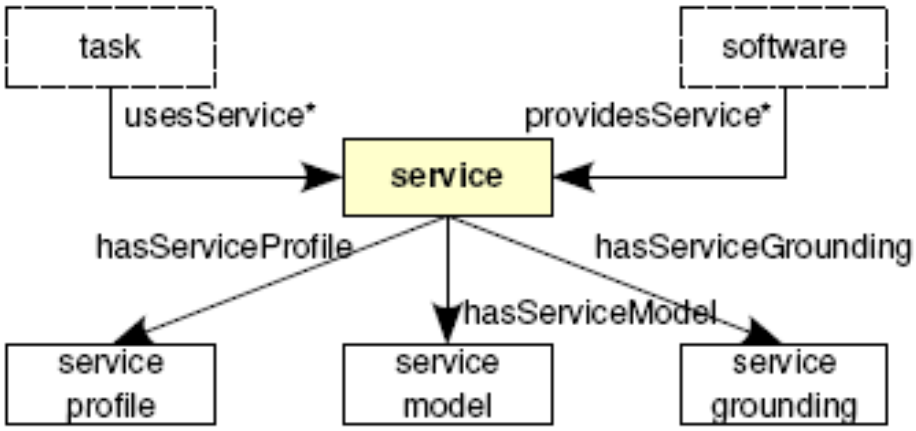

Figure 14: The Service ontology, taken from [40]

able description of what kind of functionality the service offers by specifying the inputs and outputs, the service provider and quality ratings that can be used for service discovery. The service model describes, more detailed, what happens when the service is carried out, including control- and data-flow specifications. The service grounding specifies implementation details such as communication protocols and message formats.

When describing the activity of a user, an application needs to be able to distinguish between different types. Some are planned (which can be sensed from an agenda), others are spontaneous (which can be deduced from location and time). With the User ontology in Figure 11, it is not possible to extract this divide. In addition, a task is associated with a service, which is not always the case (e.g. meetings). This should be handled by the ontology.

4 The context ontology

Context can be modelled using several techniques, of which ontologies were chosen as opposed to the others. As was described in section 2.3 and 2.2, there exists different main types of context that all share some general characteristics. Based on this, some requirements that the context ontology needs to support can be extracted. This also imposes some requirements to the context-managing architecture.

This chapter will provide a discussion of why ontologies were chosen to model context and extract the requirements to the context ontology. Finally, the context ontology will be presented.

4.1 Why represent context through ontologies?

According to [45], a lot of the previous attempts to make context aware application prototypes share that they all lack support for knowledge sharing and reasoning. The source for this is that they are not built on a foundation of common ontologies with explicit semantic representation. There have also been some attempts to define context and its inter-relationships using graphical oriented approaches.

According to [36], there are 3 types of context modeling approaches.

1. The Application-oriented approaches tailor the context model to each specific application, making the context models proprietary. These models lack formality and expressiveness and do not support knowledge sharing across different systems.

2. The Model-oriented approaches use conceptual modeling approaches to represent context, e.g. modelling context using ER diagrams or UML diagrams and managing it with relational databases. These models are better than the application oriented models because they support formality and capture the temporal characteristics of context information. However, they do not address knowledge sharing and context reasoning.

3. The Ontology-oriented approaches use ontologies to represent context, have great formality and facilitates context reasoning.

Remember that context information has a great variety, is inter-related, temporal and imperfect. The modelling approach thus needs to be flexible enough to handle all of this.

As was described in section 3.1, an ontology entails some sort of world view with respect to a given domain by defining a vocabulary that formalizes the domain into semantic terms. The semantic is achieved by describing entities, attributes and the inter-relationships between them. Ontologies are thus well suited to store the knowledge concerning context, because it minimizes the formality gap [38] much more than the other context modelling approaches. And

with formality comes inter-operability, stability (easier to validate and check the consistency resulting in a more stable model) and reusability, which is desirable for any information model.

Ontologies can be specified using a number of different languages (see section 3.3). They all have their strengths and weaknesses related to different criteria (e.g. how much it has been used), but one thing they have in common is the capability to represent information with both soft (few restrictions and properties) and strict (many restrictions and properties) constraints, making it possible to describe (with varying degree) inter-operability and hierarchies between entities. For instance, with OWL the developer is able to create both defined and undefined classes, making it possible to describe information with varying degree of precision. Thus, ontologies are very flexible for creating information models and thus well-suited for representing contextual information.

OWL expands RDF by adding more vocabulary for describing classes and properties among them (see section 3.3). Although WSML and SWSL also handle these things, OWL was chosen since it has been used, and reported to be successful for developing ontologies, far more than WSML and SWSL. In addition, ontology editors and developer guides are not that accessible for WSML and SWSL, making it time-demanding to use them.

4.2 Requirements specification

This section will present the requirements to the context ontology, which are based on the discussion of context in chapter 2.

The context ontology will be the information model of context to be used in the CBCC application and it is vital that it takes the nature of contextual information into account. Remember from 2.3 that contextual information has four general characteristics.

First of all, contextual information is temporal making the contextual information that is used by applications vary a lot. Thus, the model needs to be able to be updated frequently. Further on, contextual information is imperfect in three ways; it may be out of date (as a result of the temporal nature), faulty (errors on transmitted context data) or missing (loss of transmitted information). Consequently, the ontology needs to take all of these issues into account.

In addition, it has many alternative representations. The same contextual information can be represented in many different formats in different applications forcing the model to facilitate alternative representations of contextual information. Finally, contextual information is highly interrelated. A lot of contextual information depends on other contextual information, requiring the ontology to be able to represent this.

In section 2.2, it was pointed out that when classifying contexts into types, some types are more important (in terms of generality and uniqueness) than others. The former was denoted as primary information while the latter was denoted as secondary information. In this master thesis, the primary information will

not be viewed as more important than the secondary information - instead it shall be the basis of the secondary information. The secondary information will be deduced from the primary information. Thus, the model needs to take this type of hierarchy into account.

4.3 The context ontology

This section presents the context ontology that was developed in OWL using Protégé-OWL. Several suggestions were developed, of which two of them stood out to be the best. They both provide a hierarchy of classes and properties between them, describing the interrelations and hierarchy of the context information. Below is a presentation of each, followed by a discussion explaining why the chosen one was preferred. Figure 15 contains (to the left) the legend of the graphical representations. Before continuing, the reader is encouraged to consult Appendix D for more information on OWL and the Protégé-OWL API.

Alternative 1 - Initial ontology-approach The first solution has defined Context and Entity as the topmost classes in the ontology (the top-most class when using Protégé-OWL is Thing, which represents the set containing all individuals). Remember that context is defined as any information that can be used to characterize the situation of an entity. Consequently, Context and Entity are the most important concepts (see Figure 15).

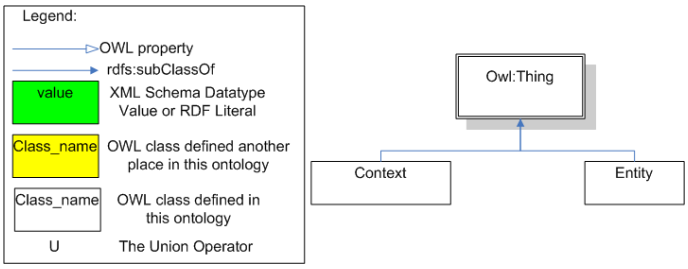

Figure 15: Alternative 1, owl:Thing

Further on, Information is considered to be the most natural sub-class of Context (illustrated in Figure 16). The main context information types are primary (sensed, raw-data) and secondary information (refined, deduced). In addition, one of the requirements to the ontology is to be able to represent quality aspects of the context information. This is achieved by QualityInfo (quality aspect of a certain type of information, e.g. time_freshness), ParametersInfo (parameters that are needed by the QualityInfo, e.g. Freshness) and MetricInfo (the metric of the parameters, e.g. type=integer, unit=seconds and value=500).

The primary information is described in Figure 17. As can be seen, Activity and PhysicalEnvironment are the two main types of primary information. Activity

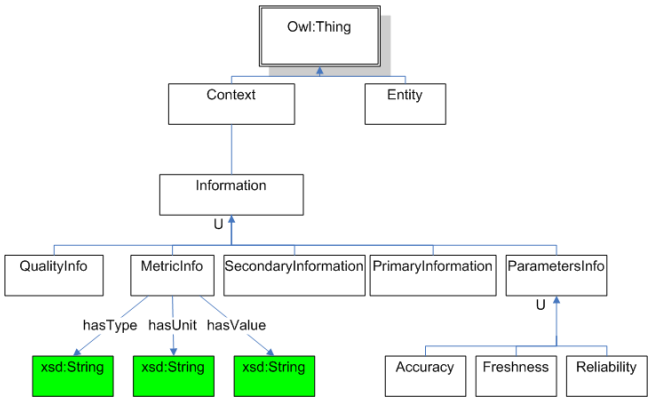

Figure 16: Alternative 1, Context

can be further divided into Status (explicit user-defined state e.g. "I am out for lunch"), ScheduledActivity (typically registered in the calendar on the terminal) and DeducedActivity(activity that is deduced based on some other information). The PhysicalEnvironment contains among others information about the space that the entity is in, including distance, surface area and physical location of the user. There are two type of locations: Relative (described by an Address) and Absolute (described by a Coordinate). One might argue that the Relative location should be placed under SecondaryInformation, since it is often deduced based on the Absolute location. This will be discussed in the evaluation of the demonstrator, see section 5.5.

The secondary information (illustrated in Figure 18) is typically more refined and based on primary information. It includes the presence of the user (describes a deduced version of the location of the user) and the degree of how available the entity is (Availability).

Alternative 2 - Final ontology The previous suggestion has several negative features, that are improved in this ontology. The quality-describing information is no longer a subclass of the ContextInformation, instead it is related to the ContextInformation through an object property (illustrated in Figure 19).

The Entity (see Figure 20) is realised as having three disjunct sub-classes: ComputationalEntity, PlaceEntity and PersonEntity. By making all the subclasses disjunct from each other and stating that the Entity is a union of the three subclasses, means that the Entity represents the set of individuals that includes all the individuals belonging to ComputationalEntity, PlaceEntity and PersonEntity. In addition, each Entity has a unique identity that is represented through an XML Schema String value. The PersonEntity has the subclasses PreferenceProfile, ProfessionalProfile and SocialProfile.

It is now possible to define whether the PrimaryInformation (see Figure 21) is

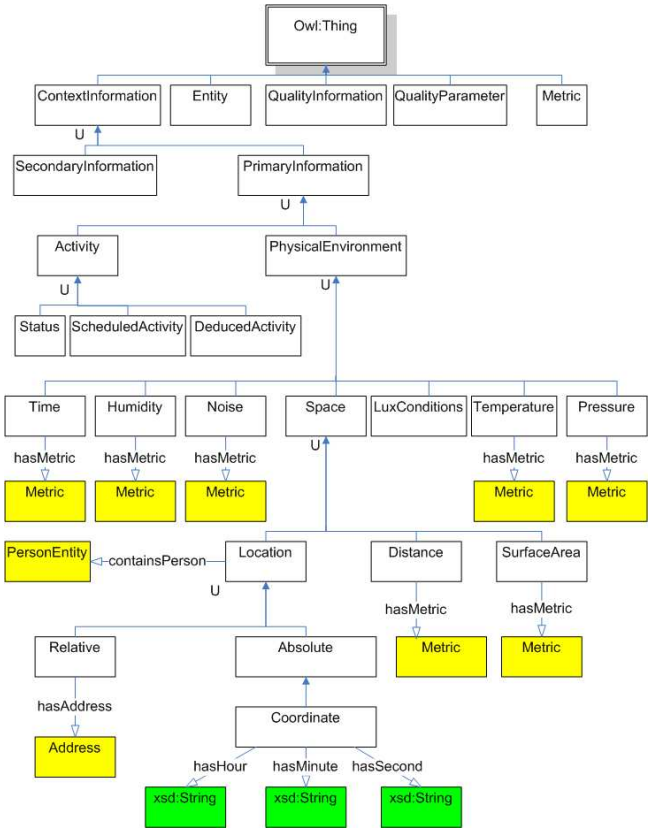

Figure 17: Alternative 1, Primary information

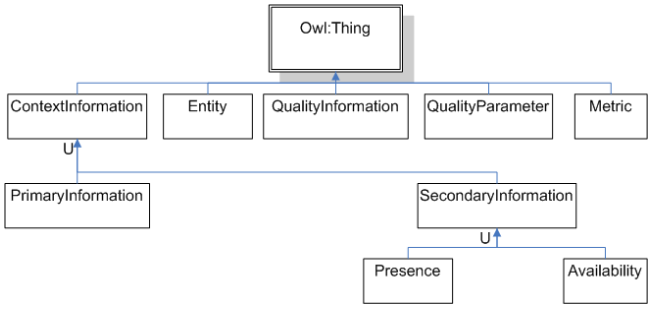

Figure 18: Alternative 1, Secondary information

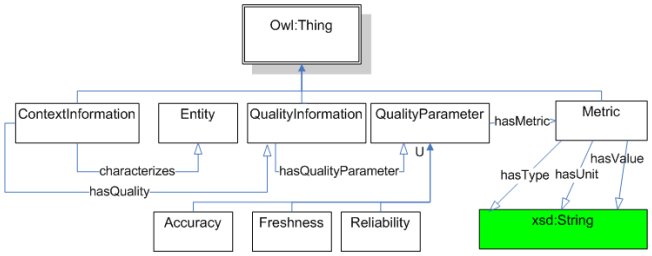

Figure 19: Alternative 2, owl:Thing

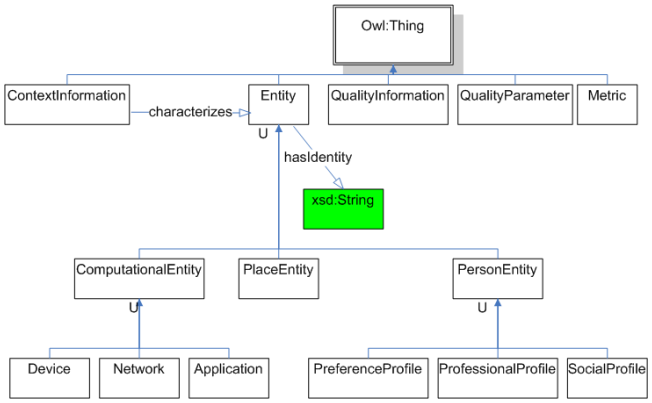

Figure 20: Alternative 2, Entity

explicitly or implicitly acquired and whether it is actively used or stored for later user. The Activity is realised with two subclasses in this scheme; InstantStatus (a renamed version of the Status from Alternative 1) and ScheduledActivity. DeducedActivity is still a part of the ontology, however it is moved to secondary information. This is because it is refined and based on other information, making it more of a secondary information than a primary.

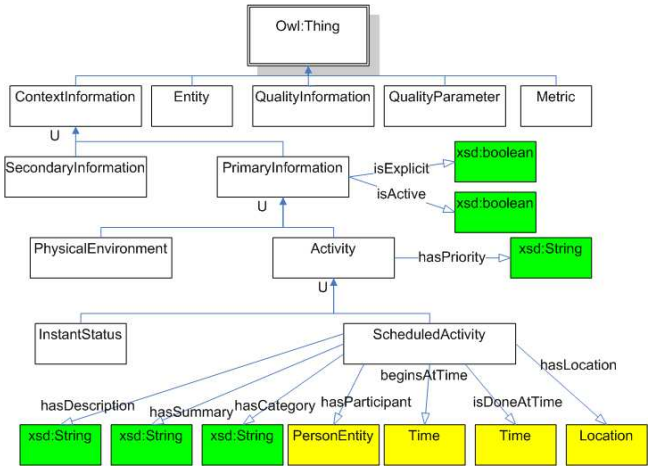

Figure 21: Alternative 2, Primary information - Activity

The Space class is also changed in Alternative 2 (Figure 22), the Destination is moved to secondary information because of the same reason as DeducedActivity was moved. In addition, Relative has an XML schema string value to represent the address instead of having an Address class representing it. This is because there really is not point in having a separate class represent the address of a relative location, since the address class would need to contain a string value of the address. The reason for having classes is to be able to represent different individuals (e.g. homeAddress and houseAddress) that potentially has the same content (homeAddress="Innherredsveien 13" and "houseAddress="Innherredsveien 13"). This is not needed here, if the string values of the addresses are equal, then the addresses must be equal. In addition, to be able to define a Relative location, it can be given an Altitude, Longitude and Latitude. With this information, it is possible to map between Absolute locations and Relative locations. Further on, absolute locations can be expressed in many different formats using different geographical coordinate systems (called a datum). To be able to fully describe an absolute location, a property specifying this is added. In addition, three subclasses of AbsoluteLocation is added that each has a metric describing it.

Finally, the secondary information (see Figure 23) is expanded with three classes. First, Distance is moved from primary information and given a property that can describe which entity the distance it related to. Second, DeducedActivity

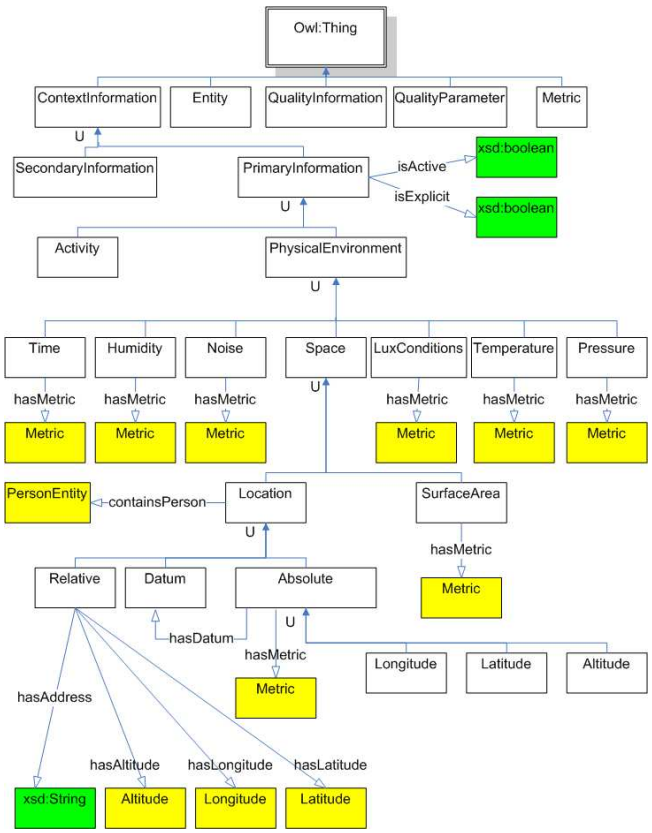

Figure 22: Alternative 2, Primary information - Physical information

is moved from primary information and given properties so that it is able to specify the participants and the description of it. Third, PersonRole is added so that it is possible to explicitly specify what kind of role a person has in a situation. The Presence is related to a relative location in addition to having an Availability that specifies the degree of availability given the entity's presence.

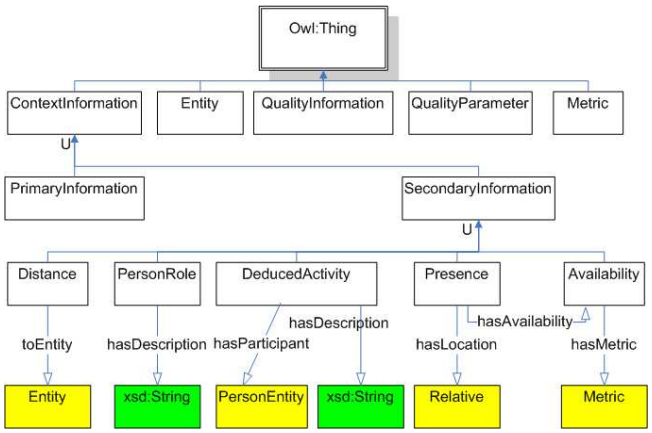

Figure 23: Alternative 2, Secondary information

5 Demonstrator: the Context Based Call Control service

This chapter will present the demonstrator service. First, an overall description of it will be given followed by a requirements analysis. Further on, the architecture of the application will be presented together with the implementation details. Finally, an evaluation of the implementation will be given.

5.1 Requirements specification

This part will first present a problem scenario that highlights the *issues* that the CBCC service will cope with. Secondly, a user scenario is given to illustrate *how* the CBCC service will cope with the issues that are presented in the problem scenario. Finally, based on the problem and user scenario, the explicit functional requirements for the CBCC service will be presented.

5.1.1 Problem scenario

John is a salesman in a telecom company, located in the middle of Trondheim. It is 8:30 a.m. and he is on his way to the office in his car. This morning has so far been really bad; he got up too late, spilled a cup of milk on the floor and fell on the slippery pavement when walking to his car. As if that was not enough, the news on the radio has announced that there will be some traffic this morning.

Suddenly, his cell phone starts ringing. He mutes the radio and picks it up. It turns out that its a customer that he talked to yesterday afternoon. The customer has talked with his superiors and is interested in buying one of the products that he talked with John about yesterday. However, he needs more information about another product before he makes a decision. John tells the customer that he unfortunately is in the car right now but that he will be in the office in 5 minutes and can call the customer back then. The customer is not too positive about this, since he will be in meetings till 2 p.m. John therefore suggests that they call each other then.

After hanging up, his spouse calls him to let him know that she will be late today because of a meeting in the afternoon. They agree on postponing dinner till 7 p.m.

Thirty minutes later, John finally arrives at the office. The traffic was a lot worse than assumed due to a traffic accident. While walking into the office building, John checks his agenda on the cell phone. It turns out that he was supposed to be in a meeting with Jane, one of his co-workers, at 9 a.m. in the meeting room. So instead of going to his office, he runs down the hall to the meeting room.

It turns out that Jane was also a bit late, so they arrive almost at the same time. Ten minutes into the meeting, John's phone starts ringing. He picks it up

to find out that it's the same customer that called him this morning while he was on his way for work. Since it looks like it's OK with Jane that he answers the phone call, he takes the liberty to inform the customer about the product right now. After 5 minutes, he hangs up and gets back to the meeting with Jane. She hasn't been idle, so they're able to finish the meeting a bit earlier than planned.

At the office, his desk phone is ringing. He picks it up to find out that it is the same customer that called him this morning. He informs John that he tried to get hold of him on his cell phone, but that nobody answered so he had to try this phone number instead. John politely apologizes for this inconvenience. The customer then informs John that his company have decided to go for the product they talked about yesterday and would like for John to email him the necessary documents. After hanging up, John fills out the contracts and documents before emailing them to the customer. Suddenly, his desk phone rings again. It is his spouse calling to remind him about buying the present for their friend Paul that has a birthday party this weekend. After hanging up, he wants to make a memo about this on his cell phone. Unfortunately he can't find his cell phone so he starts thinking about where he used it the last time. Strangely, he can't think of having used it more recently than the time he was in the meeting. He walks to the meeting room, to find out that not only has the phone been lying there since the meeting with Jane, but in addition 2 people have tried to reach him while he was in his office. He checks to find out that it was his spouse and the customer from yesterday that called. After making the memo, he calls one of his other customers to schedule a meeting at 1:30 p.m.

At noon, John goes to a restaurant to have lunch with some friends. They have made it into a tradition to have lunch together once a week, and he really enjoys this since they don't get together much more beyond that. On his way there, he calls his secretary to let her know that he is out for lunch.

When they have gotten the main course, his phone starts ringing. He picks it up to find out that it is a customer on the line. He has some questions about the different products that the company offers, and although feeling forced, John decides to answer the customer. Five minutes later he agrees to send the customer an email with some product sheets and hangs up. He gets back to his meal to find out that his friends are almost done with theirs.

When returning to his office, John stops by his secretary to let her know that he's back again. He checks his schedule and starts preparing for the meeting at 1:30 p.m.

In the middle of the meeting, his cell phone starts ringing. He chooses to hang up instead of answering the phone call. When the meeting is done, he retrieves his phone and finds out that it was his spouse that was trying to call him. He calls her back and finds out that her meeting was cancelled so they can have dinner earlier still.

5.1.2 User scenario

John is a salesman in a telecom company, located in the middle of Trondheim. It is 8:30 a.m. and he is on his way to the office in his car. This morning has so far been really bad; he got up too late, spilled a litre of milk on the floor and fell on the slippery pavement when walking to his car. As if that was not enough, the news on the radio has announced that there will be some traffic this morning.

At the same time, his secretary gets a phone call destined for John. It turns out that it is a customer that John talked to yesterday afternoon. John has defined a policy that forwards all phone calls for his cell phone to his secretary if he hasn't arrived work yet and the call origins from a phone number that John has defined as work. The customer has talked with his superiors and is interested in buying one of the products that he talked with John about yesterday. However, he needs more information about another product before he makes a decision. Since John is not in, his secretary forwards the customer to another salesman that is able to give him the information he needs.

A couple of minutes later, John's spouse tries to call him on his cell phone. Since the call origins from his spouse's phone number, which John has defined as family, the call is forwarded to his cell phone. She lets him know that she will be late today because of a meeting in the afternoon. They agree on postponing dinner till 7 p.m.

Thirty minutes later, John finally arrives at the office. The traffic was a lot worse than assumed due to a traffic accident. While walking into the office building, John checks his agenda on the cell phone. It turns out that he was supposed to be in a meeting with Jane, one of his co-workers, at 9 a.m. in the meeting room. So instead of going to his office, he runs down the hall to the meeting room.

It turns out that Jane was also a bit late, so they arrive almost at the same time. They work well together and are able to finish the meeting 10 minutes earlier than planned.

On his way to the office, he passes his secretary's desk. She tells him that a customer called this morning and so she forwarded him to Dave, one of John's colleagues. John then makes a detour to Dave's office to ask him about what the customer wanted. Dave tells him that the customer only wanted to see the product sheet of another product, and so Dave sent him this in an email, including John as a recipient. John thanks Dave for the favor, and heads for his office. At the office, his desk phone is ringing. It is the same customer that tried calling him this morning that is calling. Since John is at the office, the call is forwarded to his desk phone. The customer has decided to go for the product they talked about yesterday and would like for John to email him the necessary documents. After hanging up, John fills out the contracts and documents before he emails them to the customer. Suddenly, his desk phone rings again. This time it is his spouse that is calling to remind him about buying the present for their friend Paul that has a birthday party this weekend. After hanging up, he

wants to make a memo about this on his cell phone. Unfortunately he can't find his cell phone so he starts thinking about where he used it the last time. Strangely, he can't think of having used it more recently than the time he was in the meeting this morning. He walks to the meeting room, to find out that the phone has been lying there all the time. He doesn't bother to check if somebody tried calling him while he was in his office, since he knows that all phone calls will be forwarded to his desk phone in the office. After creating the memo, he calls one of his customers and schedules a meeting at 1:30 p.m.

At noon, John goes to a restaurant to have lunch with some friends. Forgetful as he is, he has scheduled and stored his lunch appointment in his calendar on his cell phone. They have made it into a tradition to have lunch together once a week, and he really enjoys this since they don't get together much more beyond that.

They are left unbothered throughout the whole meal and get to talk about old times and enjoy the food jointly. John has defined a policy that will forward all work-related calls to his secretary, so that he is left unbothered during his lunch time.

At the office, John is informed by his secretary that a customer tried to get hold of him but that she forwarded the call to Dave. John then goes to talk to Dave, and gets to know that it was a new customer that had some questions about some of the products and that Dave made him one of his customers. Feeling bitter that he missed out on a new customer, John heads back to his office. After all, they work on provision. Back in his office, he is in a better mood - after all the customer did call in the lunch break. He checks his schedule again and starts preparing for the meeting at 1:30 p.m.

The meeting goes as planned since John has registered a policy that forwards all calls to his voicemail so that he and the participant are left unbothered. After escorting the customer out, he checks his voicemail to find out that his spouse has left him a message informing him that her meeting was cancelled so they can have dinner earlier still.

5.1.3 Functional requirements for the CBCC service

This section will present the functional requirements of the CBCC service that is based on the User scenario presented in section 5.1.2. Table 2 provides a summary of the functional requirements of the CBCC service.

In addition, the contextual information will be sent from various sources (called sensors) in varying formats. Consequently, the system needs both to be able to represent the information in varying formats and cope with the varying formats. In addition, it needs to be able to receive context information from different types of sensors. This is what requirement 1 describes.

In R3 it is claimed that there are different *levels* of context information. Remember from section 4.3 that there are (among others) sensed context information, deduced context information, primary context information and secondary information. These are all different types of context information, and also go

in different levels in terms of how understandable they are to the user. It is important that the system is able to distinguish between the different levels of the context information in addition to the different types of context, so that it is possible to map the current context to the context values that are used in the policy predicates. This is what requirement 4 describes. For instance, a user would define a policy as "Forward all calls to my cell phone when I am in a meeting" and not "Forward all calls to my voicemail when I am present at longitude X, latitude Y, altitude Z and I have a scheduled activity at the current date and time".

Requirement	**Description**
R1	The system must be able to update different kinds of context information
R1.1	The system must be able to fetch and receive context information
R1.1.1	The system must be able to fetch and receive time
R1.1.2	The system must be able to fetch and receive location information
R1.1.2.1	The system must be able to fetch and receive location information from several types of location sensors
R1.1.3	The system must be able to fetch and receive information from the user-agenda about scheduled activities
R2	The system must be able to distinguish between different kinds of callers
R2.1	The system must be able to distinguish between phone numbers that the user has defined as work- and family-related
R3	The system must be able to distinguish between different levels of context information
R3.1	The system must be able to distinguish between primary (sensed) and secondary (deduced/reasoned) information
R4	The system must be able to reason on the context information to map it to context information that is understandable for users
R4.1	The system must be able to map primary context information to secondary context information
R5	The users of the system must be able to define call control policies that are based on user-understandable context predicates

Table 2: Functional requirements of the CBCC service

5.2 System architecture

This section will present the architecture of the CBCC service. Issues when designing the system architecture was how it could be deployed in ServiceFrame and how the context managing unit could be realised.

5.2.1 System overview

As was described in section 2.4, there are three isolated types of context-aware functions that can be supported by an application: presentation of information and services, automatic execution of a service and tagging of context for later retrieval. As for the CBCC application, it supports automatic execution of a service - namely call control. Routing decisions shall be made based on the user defined policies and the context of the users, where calls origin from different types of telecom networks. To achieve this, the application interacts with NRG to fetch location and perform call control. An overview of the CBCC service is given in Figure 24.

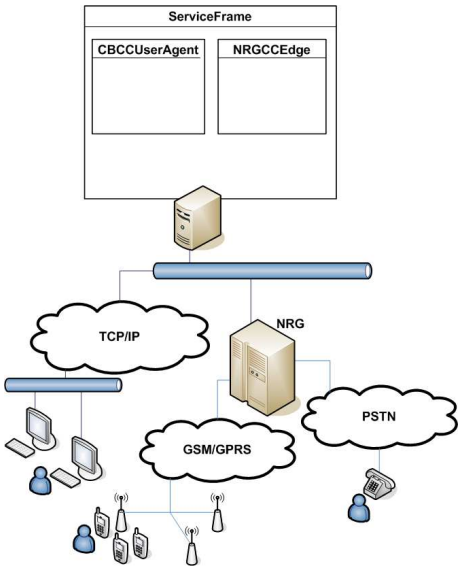

Figure 24: Overview of the CBCC service

5.2.2 The architecture of the CBCC service when deployed in ServiceFrame

This section will present the different alternatives of the architecture of the CBCC service related to deployment in ServiceFrame. There are two main alternatives: a service oriented one and an agent oriented one. The different approaches as to how the context managing unit could be realised will be presented in section 5.2.3.

Alternative 1: Service oriented approach In the first main approach (illustrated in Figure 25), the context engine is realised as a standalone node in

ServiceFrame. Remember from section 2.8 that most of the architectures for managing context (Context Toolkit, SOCAM, CoBrA) are realised as standalone nodes where other services can subscribe to the context information that is collected, interpreted and collocated. In ServiceFrame, it is also possible that the UserAgent itself can receive or fetch context information from the context engine. The services are realised as standalone actors interacting with UserAgents, TerminalAgents and technology-specific edges for communication. Since the CBCC application shall collaborate with the NRGCCEdge that was designed in the project assignment "Status Based Call Control" and therefore only is to make a routing decision, terminal agents are not needed. With this approach, the ContextReasoner would administer OWL models (see section 5.2.3) for all the UserAgents.

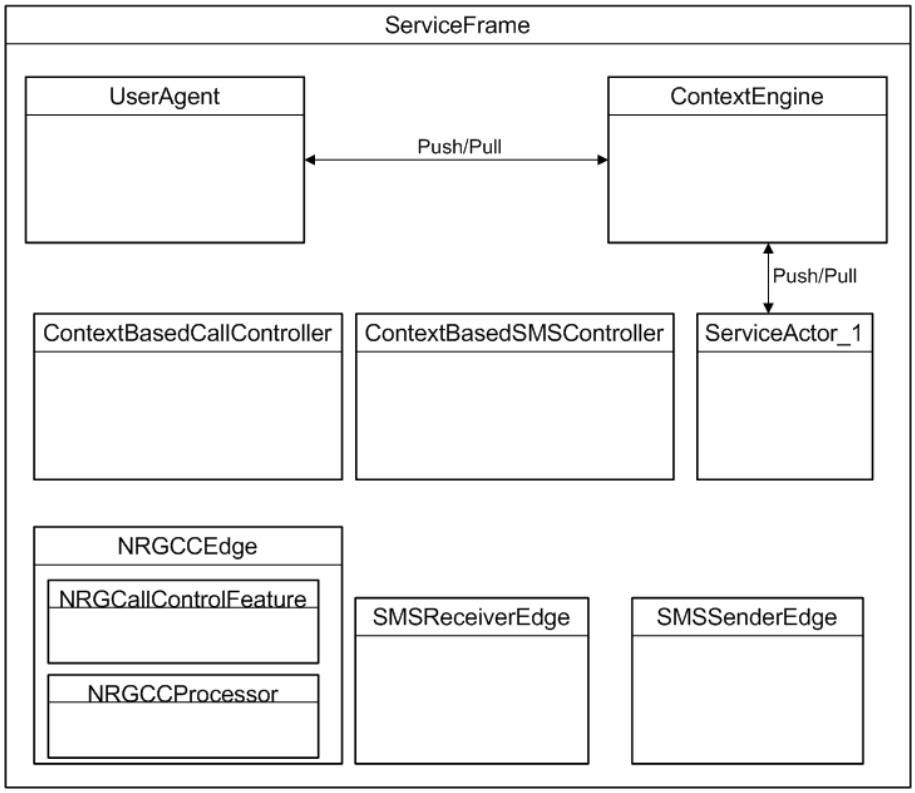

Figure 25: Alternative 1 of the CBCC service when deployed in ServiceFrame

A consequence of this scheme, is that some sort of context manager is needed. There could be a number of services or UserAgents that are subscribing to context information from the context engine. There needs to be some sort of managing agent that is responsible for mapping context information with the correct UserAgent.

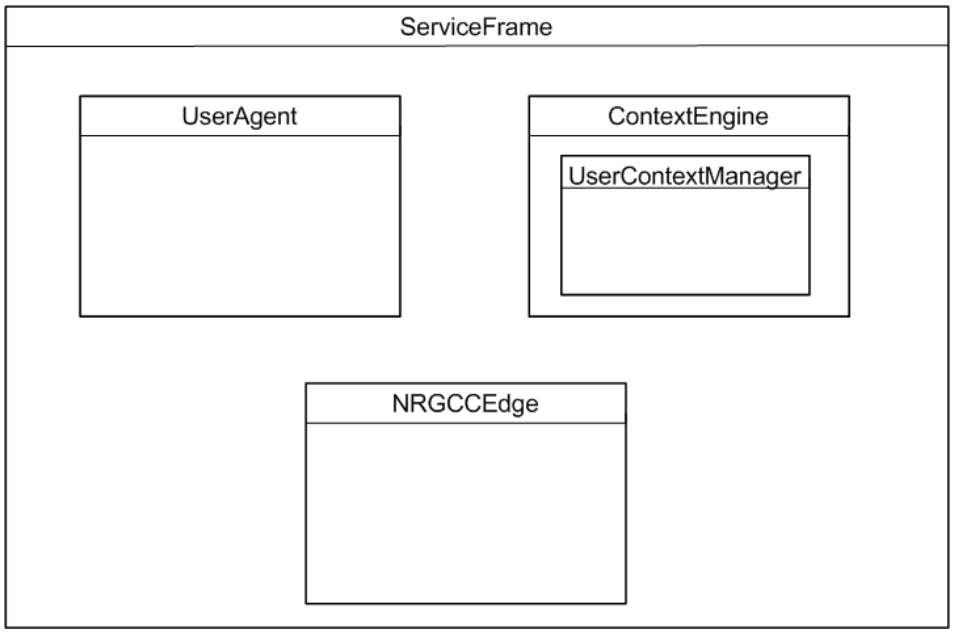

Figure 26: Alternative 1 - including the user-context-managing unit

This can be solved by including the managing functionality as a part of the ContextEngine. With this scheme, it is assumed that the context engine knows the identity of the service- and user agents. This is a good solution, because contextual information in general is tightly coupled with identity. The context engine thus needs to be able to map between contextual information and the identity of the owner entity.

Alternative 2: Agent oriented approach In the second approach (illustrated in Figure 27), the service logic and context engine is placed within the UserAgent, making it a specialised case of a UserAgent (and is therefore renamed to CBCCUserAgent) that has Context Based Call Control capabilities. In ServiceFrame, the UserAgent is considered to represent the user of a service, including the information that the user knows about himself (profile, context, preferences), and is supposed to make service related decisions on behalf of the user. This is in accordance with the agent paradigm of pervasive computing [47]. ServiceFrame (see Appendix D.1) consists of agents that are defined in ActorFrame (see Appendix D.2) as actors, and actors play roles to create a service. The inner parts of CBCCUserAgent are defined in ActorFrame as actors, each playing their distinct part of the CBCC service that is provided by the CBCCUserAgent.

The first approach needed some sort of managing actor mapping sensed context identities with ServiceFrame identities. This requirement is gone with the second

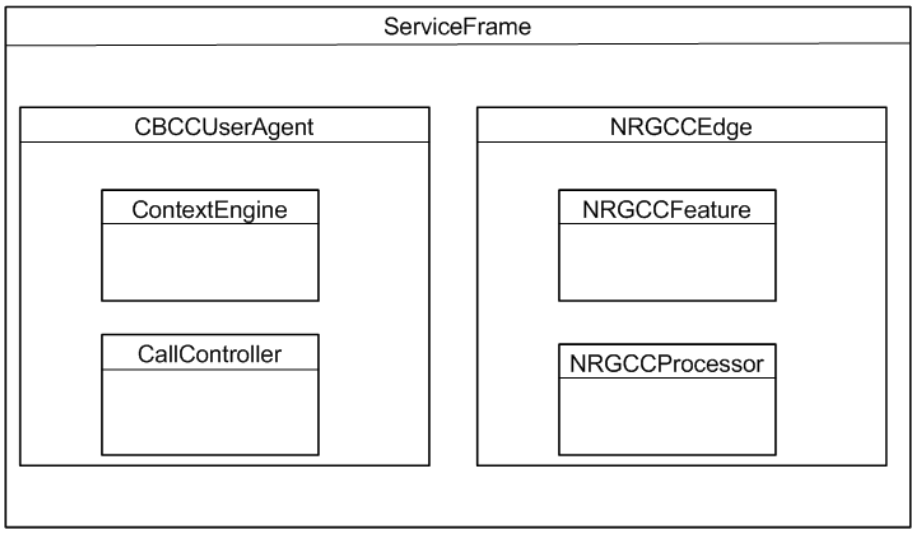

Figure 27: Alternative 2 of the CBCC service when deployed in ServiceFrame

approach. Since each UserAgent contains a context engine, the sensed context identity will be easy to map to the service-related identity, since there is a 1-to-1 mapping - the entities are the same. With this approach, the ContextReasoner would administer only one OWL model - namely the model of the UserAgent.

5.2.3 The architecture of the context managing unit

As was described in section 2.9, there are four views to context reasoning: the low level (sensors making error controls and inferring context), model monitoring (keeping the learned models consistent and valid), context monitoring (detect context changes and react to these) and application view (refinement of raw data and reasoning methods allowing policy forming). The different views divide context reasoning into different parts, where each part is responsible for offering a part of the context reasoning. The two architectures presented below are based on these views. Both of them focus on the context management, and not on how the unit is deployed in ServiceFrame (which is addressed in section 5.2.2).

Alternative 1 - initial approach The first alternative (see Figure 28) has a number of classes that are put in the 4 views, each providing a part of the functionality for the omitting view. For instance, it is assumed that the classes in the model monitoring view shall collaborate to keep the learned models consistent and valid.

The low level view is where the sensors are placed. Each sensor implements the ContextSensor's methods so that they're able to validate, format and forward

the received information. In addition, they all extend from the ContextMonitor making them able to fetch context data in addition to receiving it. Thus, the sensors should be able to both passively receive and actively fetch (thereby monitoring) context data from their environment.

In the model monitoring, we find the OWLModel class, which contains the ontology and all the individuals that are created during runtime. In addition we find the OntologyReader that is able to instantiate the model from the disk and read from it during runtime. The OntologyPopulator is able to write the model to the disk when the application shuts down and modify it during runtime, while the OntologyValidator is supposed to validate the model when changes are done.

The ContextReasoner is a part in all the views, and is the most central class in the ContextEngine. It receives information from the sensor and is responsible for deducing what the secondary information will be based on the primary information it receives. In addition, it manages the OWLModel by keeping a virtual model of it during runtime (instead of writing the information to the disk, which is quite time-consuming). This is achieved by requesting the OntologyReader to fetch the model at startup and requesting the OntologyPopulator to update the model whenever new information is received. Thus, it is both responsible for handling information from multiple sources (low level), keeping the model up to date (model monitoring) and refine the data that is received from the sensors (application view).

Alternative 2 - second and final approach The second alternative of the architecture (illustrated in Figure 29) is a modification of Alternative 1. The most important change is that the interaction between the ContextReasoner and the sensors are made more general. This is why Alternative B is preferred. In Alternative 1, if a sensor were to send context updates to the reasoner, it would invoke specialised methods that suited it specifically (and no other sensor), where the context information would be represented by string values. In the same way, the ContextReasoner had no way of populating the model during runtime. The model needs to be populated with the type of individuals that are needed by the different types of information provided by the sensors. This means that with Alternative 1, the reasoner needed to know what kind of individuals that was needed in the model at the time it is instantiated, meaning that it needs to know what kinds of sensors to support *before* they send any updates.

Whenever a sensor wants to be able to provide context information to the ContextReasoner, it is requested to register. The ContextReasoner is informed of what kind of sensor that is registering with it through the parameter. It is assumed that the sensor knows what kind of information it provides. When a sensor registers with the reasoner, the reasoner is able to populate the model with the individuals that are necessary for the type of information the sensor provides. With this approach, the reasoner doesn't have to know what kind of sensors it needs to support before it receives any information from them. In ad-

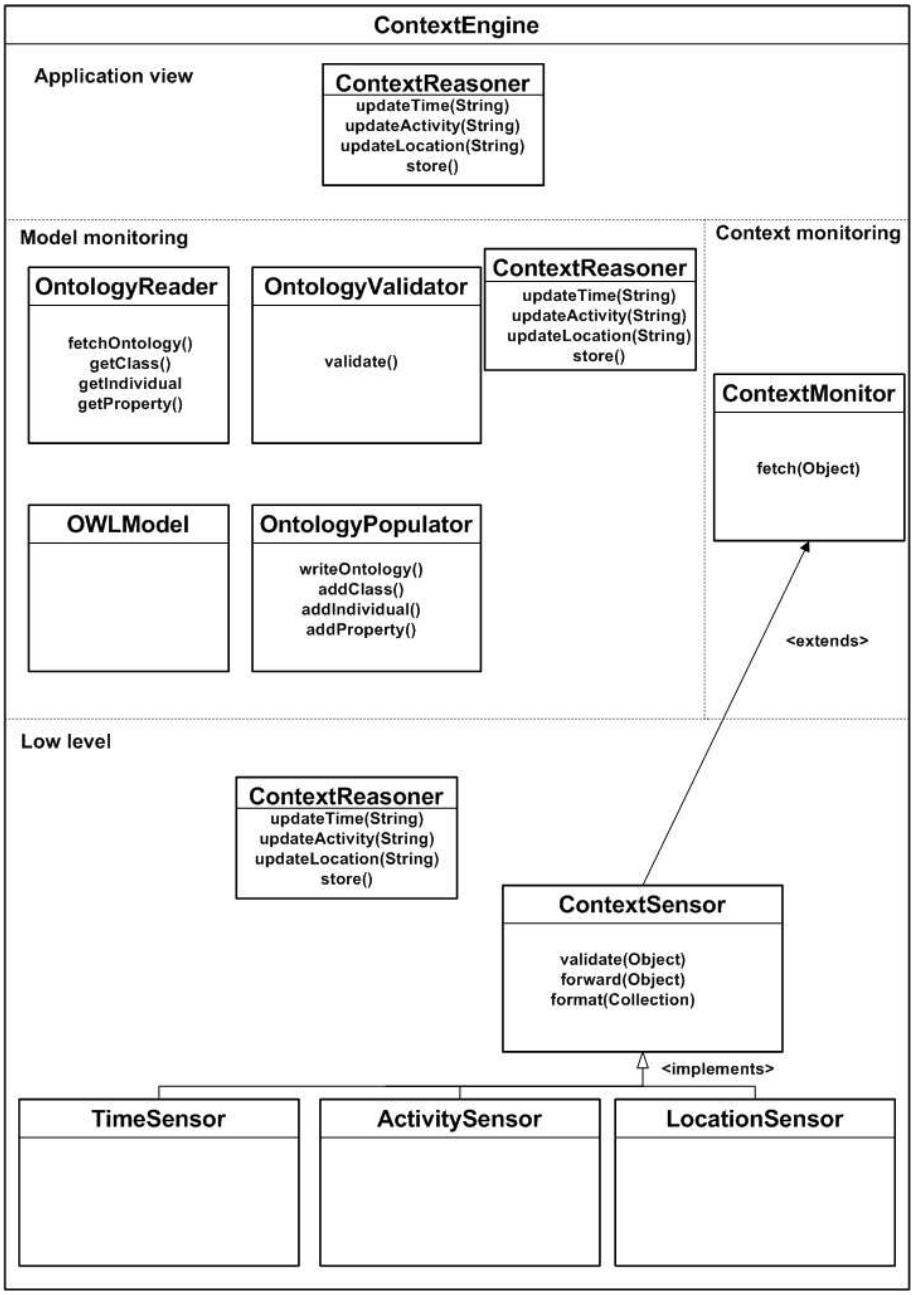

Figure 28: Alternative 1 to the ContextEngine

dition, instead of having sensor-specific methods for context updates, a general method can be invoked to update the context with the reasoner. To be able to do this, some value objects are created in the model view of the ContextEngine that the sensors can format the received context information into and forward to the reasoner. They are simple objects that contain context information of a certain type (e.g. time-information will be carried in a Time value object). A value object is created for all the classes defined in the ontology, so that new type of sensors are able to notify the reasoner about context updates. The ContextReasoner has also a new supportive class that helps it in deducing secondary information from primary information (RuleReasoner).

In addition, the OntologyPopulator and OntologyReader are combined into the OntologyOperator, the OWLModel is substituted with the JenaOWLModel because of technicalities in the Protégé-OWL API (JenaOWLModel is more functional than OWLModel).

Finally, a new view is created in which the ContextReasoner and RuleReasoner is placed: context administration. In Alternative 1, the view in which the ContextReasoner should be placed was not settled. This was because it provides (coordinates) functionality that is provided by three of the views, and so it was difficult to place it in one. However, the views do not compose separate layers of the service, offering different kinds of functionality. They simply provide different views on which tasks that are done in context reasoning - making it possible that a class is responsible for performing several tasks. The reasoner is not really directly responsible for any of the tasks that the 4 views should perform, the tasks that are supported by the reasoner are more of a managing or administrating character. This is why the context administration view was created, offering a junction for context updates, primary/secondary context reasoning and model-managing.

The application view is not really a part of the context engine, it belongs to the omitting or subscribing (depending on the architecture being used) service actors containing service-specific logic.

5.2.4 Final architecture of the CBCC service

Of the two architectures presented in section 5.2.2, alternative 2 is the preferred one. The reason for this is that it is in accordance with the paradigm of actors and agents that is imposed by ActorFrame and ServiceFrame. With this approach, each user agent is responsible for knowing as much as possible about the context of the user that it represents and make decisions related to call control on behalf of this user. All of this functionality is integrated in the user agent, instead of having a standalone node collecting context information about all users and entities.

The CBCCUserAgent (illustrated in Figure 30) will contain functionality for collecting context information and make call control decisions based on the context information and user defined policies (that are registered when the system

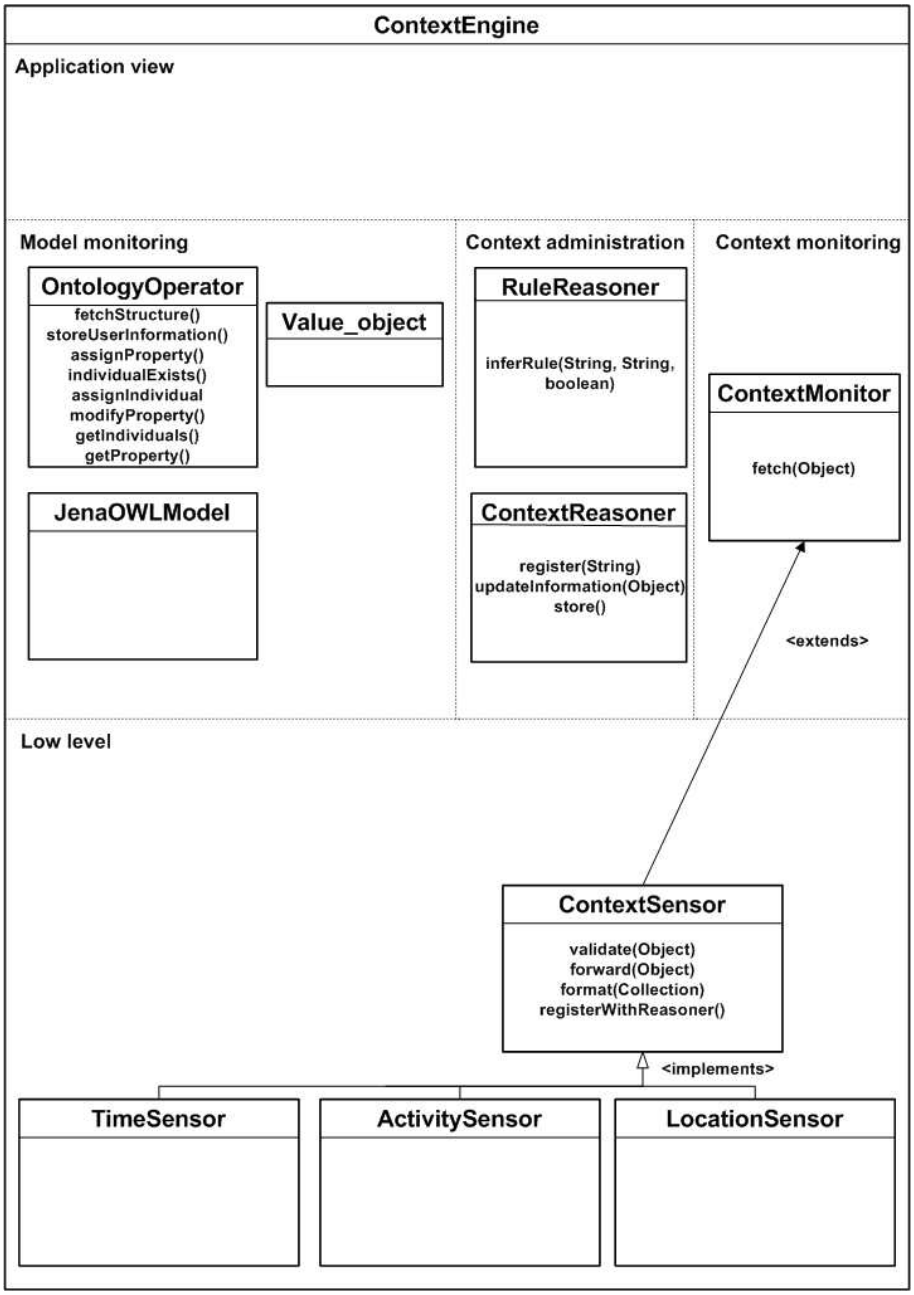

Figure 29: The final architecture of the ContextEngine

starts up). It contains 6 actors providing this functionality, and below is a description of each.

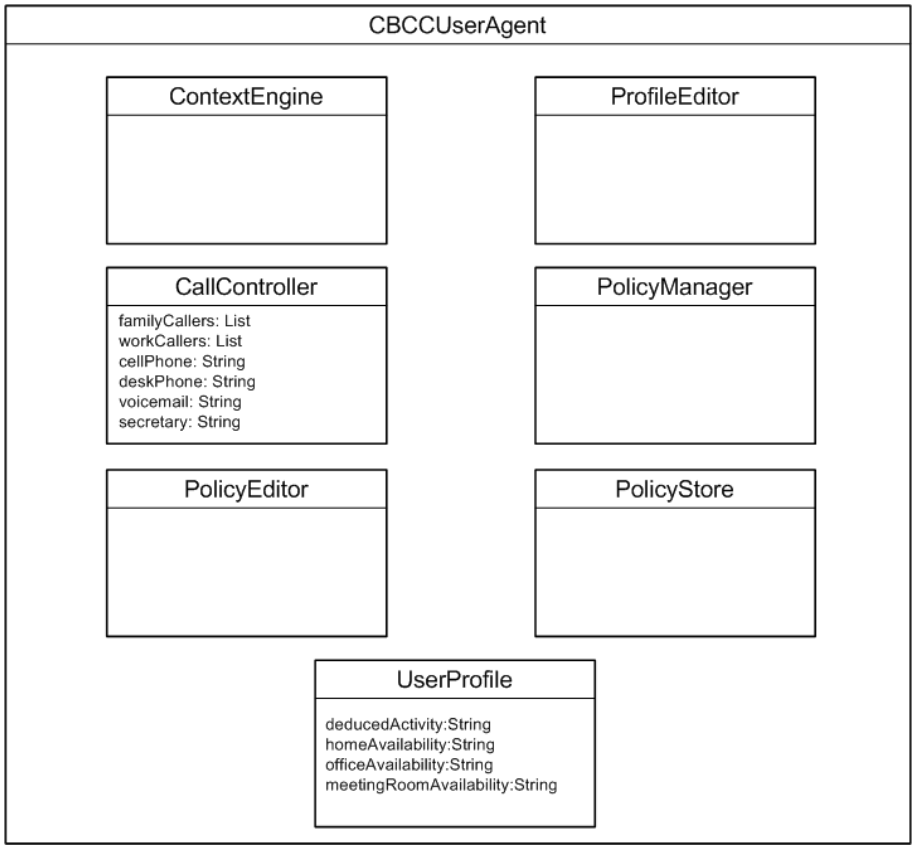

Figure 30: The structure of CBCCUserAgent

The policy management consists of 4 actors: the PolicyManager, the PolicyStore, the PolicyEditor and the ProfileEditor. A detailed description of these actors can be found in [48]. The PolicyManager is responsible for checking if the policies are satisfied for a given role. In addition, it interacts with the ProfileEditor to get notifications whenever the UserProfile changes. The PolicyStore manages storing and retrieval of policies from disk. Whenever the PolicyManager is to check the policies for a given role, it requests the PolicyStore for all the existing policies for that role. The PolicyEditor is a service actor that is used to register new policies with the PolicyStore. The ProfileEditor is responsible for updating the UserProfile (containing the secondary context information) whenever it receives updates from the ContextEngine. Whenever this happens, the PolicyManager is being notified of the change, so that the context information is up to date.

Finally, the CallController is the actor that receives route requests from the NRGCCEdge whenever a call is made to the user. It is then responsible for finding a route, and this is done by first checking what type of caller is calling the user (work or family), and then request the PolicyManager to check if there is a policy given the type of caller and the user's context.

Since the ContextEngine will only be serving 1 UserAgent, it only needs to manage 1 context model during runtime (see section 5.2.3).

5.3 CBCC concepts

The Context Based Call Control service consists of a centralized service agent that makes routing decisions on behalf of the subscribers of the service. The service agent is supposed to make these decisions in accordance with the user defined policies that are made up of several predicates containing secondary context information. For the CBCC application, the context information that is available to the user for defining policies are deduced activity, presence and availability. In addition, the type of caller is also a part of the predicate making it possible to have different policies for different types of callers (e.g. family or work) given the same context information. The policies that are used for the simulator are written in a separate property file which is described in Appendix A.7.

The ContextEngine will gather contextual information about the user, and deduce the secondary contextual information that is to be used when checking policies. Thus, reasoning rules are also defined for the ContextReasoner so that it is able to deduce the secondary information based on the sensed primary information. These rules are written in a separate property file which is described in Appendix A.8. As can be seen, the rules contain information about the primary context types time, relative location and whether or not a scheduled activity is registered (at the current time). For each of the rules, the deduced activity, home availability, office availability and meeting room availability values are specified, given the specific primary information. The relative locations (home, office, meeting room and restaurant) are defined in the property file for the OWL individuals. The time describes whether or not there is registered a scheduled activity at the current time (officeTime or activityTime). Thus, one might claim that the boolean variable describing whether or not a scheduled activity is registered at the current time is superfluous. However, other types of time may be defined that are independent of whether or not a scheduled activity is registered making the boolean variable necessary.

There are 3 levels of context reasoning in the CBCC application, performed by different actors.

1. Ontology reasoning - this reasoning is being done by the ContextReasoner. The ContextReasoner needs to recognize what type of information the sensor is sensing and recognize where to put this in the OWL model. The ontology reasoning is performed whenever a sensor registers with the ContextReasoner (the ContextReasoner needs to create the necessary individuals in the OWL

model) and whenever new information is received from the sensors (which individuals shall be updated in the OWL model). This reasoning also include potential ontology validation (validation of the model if it is changed during runtime). *The ontology reasoning is thus concerned about finding out what the sensed information means to the modelled information.*

2. Primary context reasoning - whenever new primary information is received from the sensors, the ContextReasoner needs to deduce what the secondary contextual information will be. *The primary context reasoning is thus concerned with what the primary information means.*

3. Policy reasoning - the PolicyManager is responsible for deducing which policy is prevailing at all times and the policy contains information that can be used to deduce where the call shall be routed. This kind of reasoning is performed whenever a call notification is received from NRG. *The policy reasoning is thus concerned with deciding what to do based on the context information.*

5.4 Implementation details

This part will present the major MSCs for the CBCC application. The actors and classes are explained above and in section 5.2.3. Since some of the entities in the diagram are actors or agents while others are regular java classes, the messages that start with small letters are method invocations while the messages that start with capital letters are ActorFrame messages (denoted as signals in Ramses). Before continuing, the reader is encouraged to consult Appendix C for more information on NRG and Appendix D for more information on ServiceFrame and ActorFrame.

Initializing the connection to NRG

The entities that collaborate to initiate the connection with NRG are the NRGCallControlFeature and the NRGCCProcessor for the call controlling part of the CBCC service (illustrated in Figure 31), while it is the NRGLocationFeature and the NRGLocationProcessor that collaborates for the location fetching part (illustrated in Figure 32). As it is described in Appendix C, the NRGCallControlFeature requests a reference to the SCS (IpMultiPartyCallControlManager) for call control from the proxy. From then on, service requests can be sent to the SCS. In addition, the call control processor is instantiated with a reference to the SCS for call control so that it is able to request and receive call notifications from the SCS. Further on, NRGCCProcessor is requested to register the call notifications with the SCS. For the CBCC application, it is only interesting to get notifications for 1 subscriber (the user that is represented by the CBCCUserAgent) so this will only be done once.

In the same manner, the NRGLocationFeature requests a reference to the SCS for user location (IpUserLocation). The processor for location fetching (NRGLocationProcessor) is also instantiated with a reference to the SCS for location fetching.

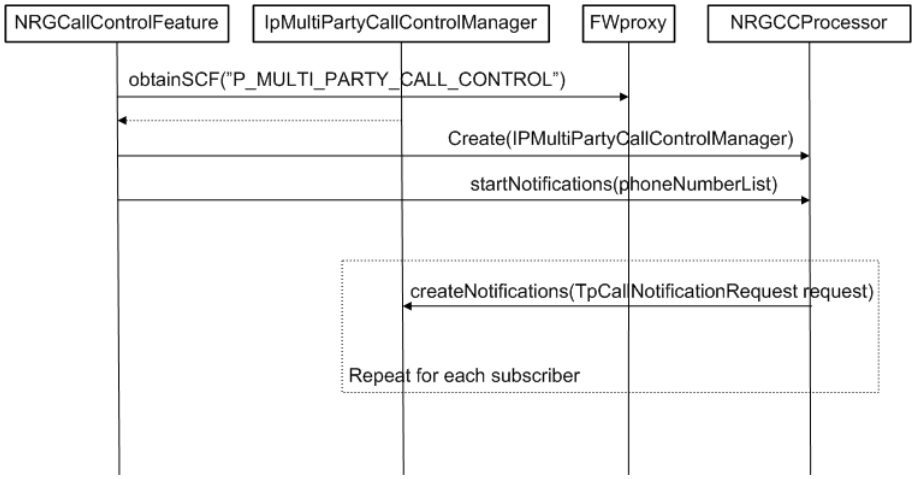

Figure 31: Initializing NRG connection for the call controller

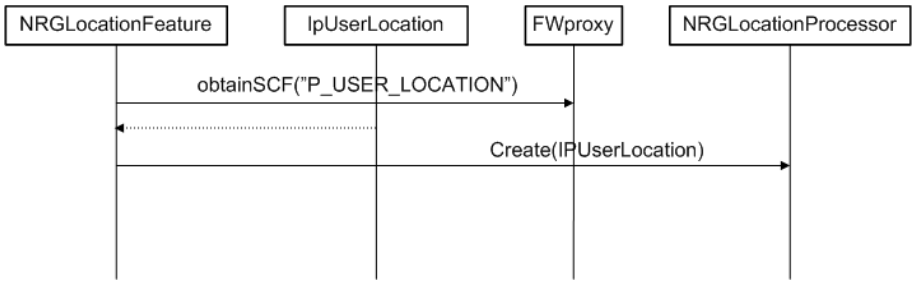

Figure 32: Initializing the NRG connection for the location fetcher

Starting up the UserAgent and registering the policies

The first thing that happens when the CBCCUserAgent is started up is that it requests the CallController to register the policies that are defined for it. As can be seen in Figure 33, the CallController interacts with the PolicyEditor to achieve this. This is done for all the policies that are defined for this CBCCUserAgent.

Secondly, the ContextEngine is requested to start up. The parameters in the signal refers to the interval size of how often the location shall be requested in hours, minutes and seconds respectively. As for the demonstrator, seconds is the only valid unit.

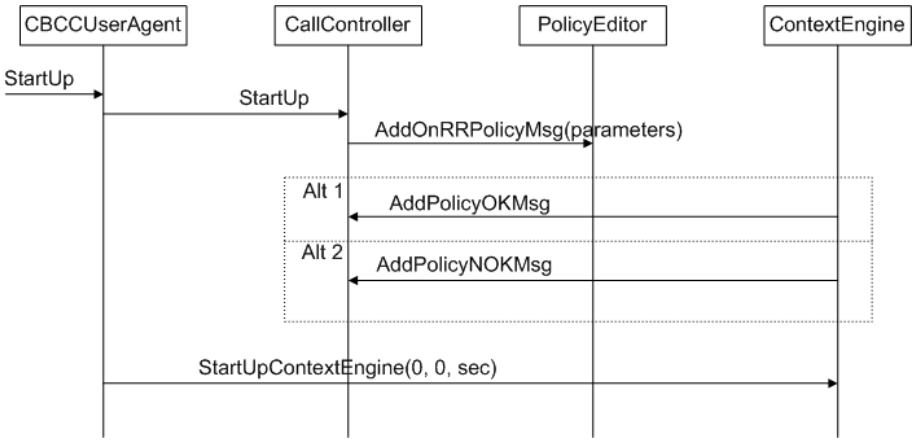

Figure 33: Starting up the UserAgent

Starting up the context engine

When the ContextEngine is requested to start up (see Figure 34), it requests all the sensors to register with the ContextReasoner. All the sensors invoke the ContextReasoner with the same method only with different parameters. The parameter in the method tells the ContextReasoner what kind of sensor that is currently registering with it. This is necessary so that the reasoner can update the OWL model with the proper individuals that is needed when new information from the sensor is received. The ContextReasoner has 1 instance of the OWL model that contains the context ontology (with all the classes and properties and the relationships between these) and the individuals of the OWL classes that is to be updated by the different sensors. The sensors therefore need to know what kind of context information they are providing.

Because the LocationSensor is an actor and that sending a signal to it is performed asynchronously, the ContextEngine is forced to wait for an acknowledge-

ment (LocationSensorRegistered) from the LocationSensor when it has registered with the ContextReasoner.

Finally, the LocationSensor is requested to start fetching the location as often as was stated in the StartUpContextEngine signal and the ActivitySensor is requested to fetch the scheduled activities that is stored for the user.

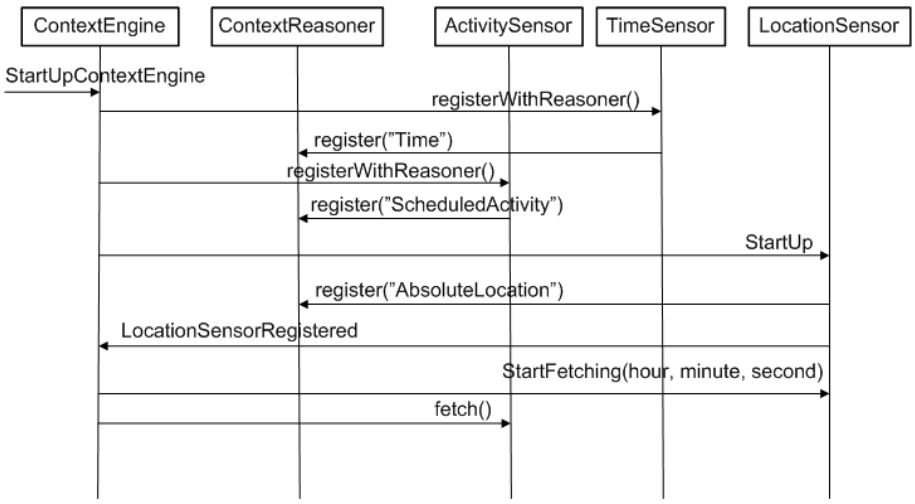

Figure 34: Starting up the context engine

Fetching the scheduled activities

The scheduled activities for the user are fetched from a property file (see Appendix A.9). The only information about each scheduled activity that will be used is the time at which it begins and ends. After the information is fetched, it is formatted (by constructing a ScheduledActivity object of it) and finally it is forwarded to the ContextReasoner (see Figure 35).

The ContextReasoner then updates the OWL model with the newly received activity and then requests the TimeSensor to fetch the time at the time when the activity begins and ends. In the demonstrator, the only times that the time needs to be updated is when a scheduled activity begins and ends. Remember from section 5.3 that the time is either after a scheduled activity has started and before it has ended (the activity is taking place now) or before an activity has started or after it has ended (no activity is taking place now) . Thus, the ContextReasoner requests the TimeSensor to send an update of the time when an activity starts (an activity is taking place right now) and when an activity ends (no activity is taking place right now).

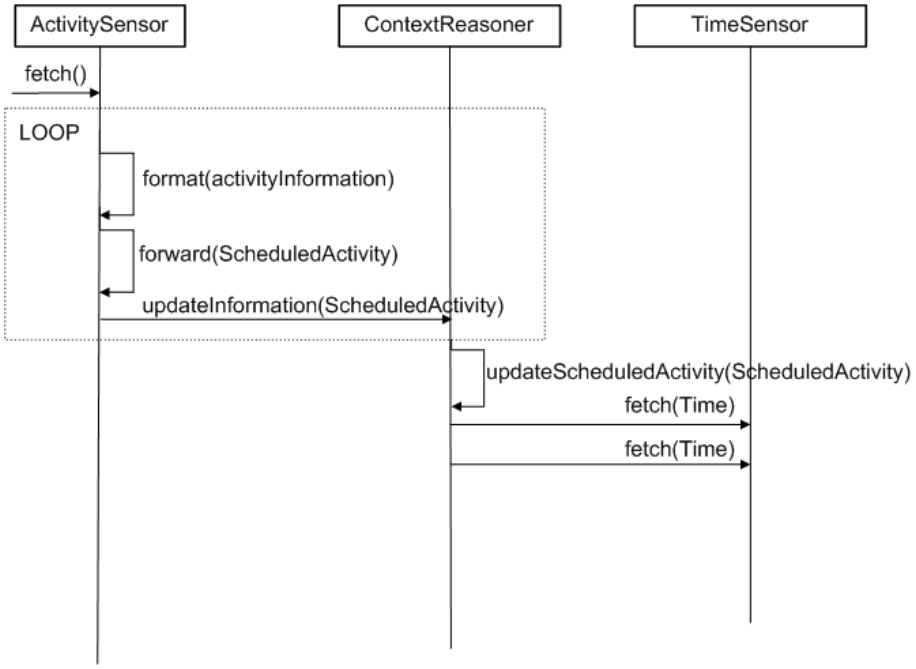

Figure 35: Fetching the scheduled activities

Updating the time

When the TimeSensor is requested to fetch the time (see Figure 36), it is also given a Time value object containing the actual date and time at which the ContextReasoner needs to get an update. It therefore creates a reminder that will invoke receiveReminder() when the date and time is reached. When receiveReminder() is invoked, the TimeSensor fetches the current date and time, formats it (creates a Time value object containing the current date and time) and forwards the information to the ContextReasoner. Since the information that is received at the ContextReasoner is of the type Time, updateTime() is invoked by the ContextReasoner.

When the ContextReasoner updates the time it performs four steps. First, it updates the OWL model with the current date and time. Second, it retrieves the current (or most recent) location and all the activities of the user that is stored in the OWL model. Third, it deduces the secondary context information (deduced activity, presence and availability) based on the current primary information. This is done by requesting the RuleReasoner to infer the secondary context (see Figure 37). This reasoning is done by having a file containing the secondary information reasoning rules (see Appendix A.8) and checking whether or not there is a rule for the current primary context. If there is, the rule will be followed. If there isn't, all of the secondary information will be set to 'nil' which

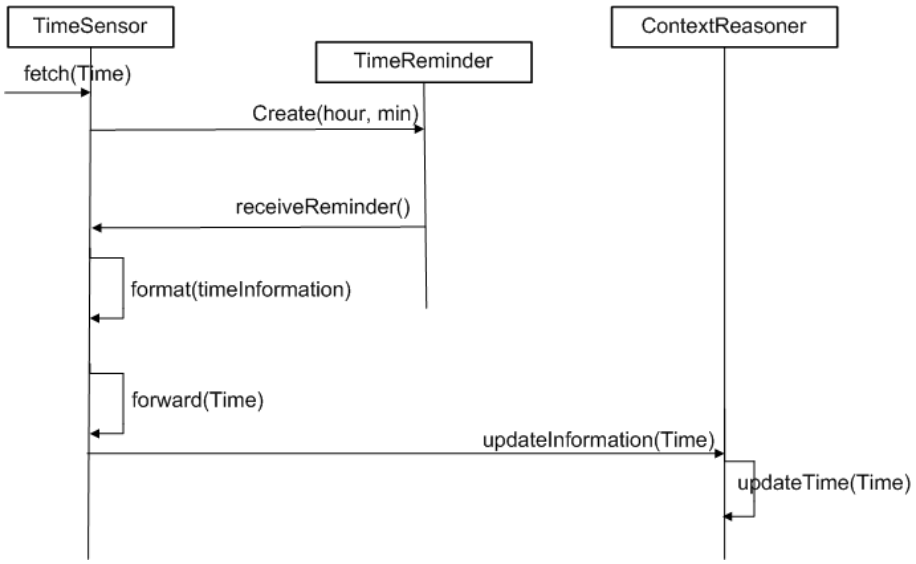

Figure 36: Fetching the time

means that they're undefined. Finally, the secondary information is sent to the omitting ContextEngine.

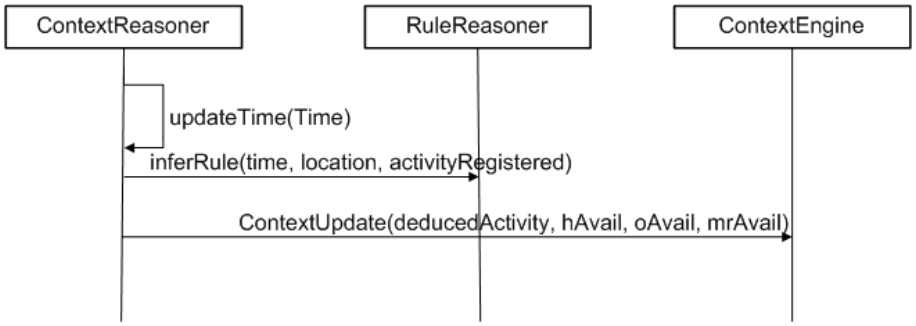

Figure 37: Updating the time

Updating the location

When the LocationFetcher is started up, it first request the NRGLocationFeature to fetch the location (see Figure 38). Second, it starts up a timer that is responsible for letting the LocationSensor know when the time for the next location fetching is (in the specified amount of time that is stated in the StartUp signal) to take place. After the location is received from NRG it is forwarded all the way back to the LocationSensor. However, this is not the only signal

that may be received by the LocationSensor first. Depending on how often the location shall be fetched, a TimeIsUp signal may be received before the location fetch response, making the LocationSensor fetch the location again (see Figure 39).

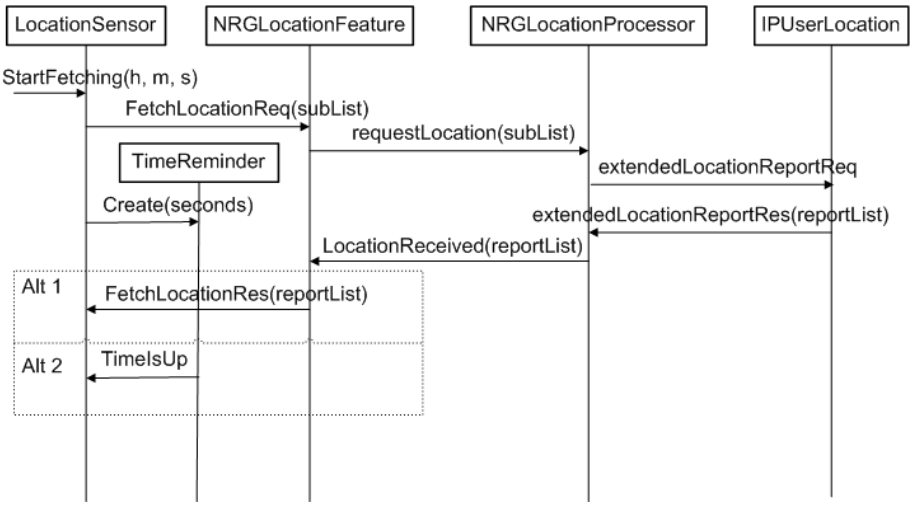

Figure 38: Fetching the location

When the location fetching response is received, the information is formatted into a Location value object and forwarded to the ContextReasoner (see Figure 39). Since the newly received context information is of the type Location, the reasoner will invoke updateLocation.

When the ContextEngine invokes updateLocation, it performs 4 things. The first thing it does is to check whether or not the newly sensed location information is different from the modelled one. If it is, it updates the location information in the OWL model. The three next steps are the same as the ones performed when it updates the time. Updating the location is shown in Figure 40.

Updating the context

When the ContextEngine receives a context update, it requests the ProfileEditor to update the user profile of this CBCCUserAgent (see Figure 41). The ProfileEditor updates the profile by writing it to the disk and notifying the PolicyManager about the change.

Handling calls

When a call to the cell phone of the user is made, a notification of this is sent to the NRGCCProcessor (see Figure 42). When it receives the notification, it

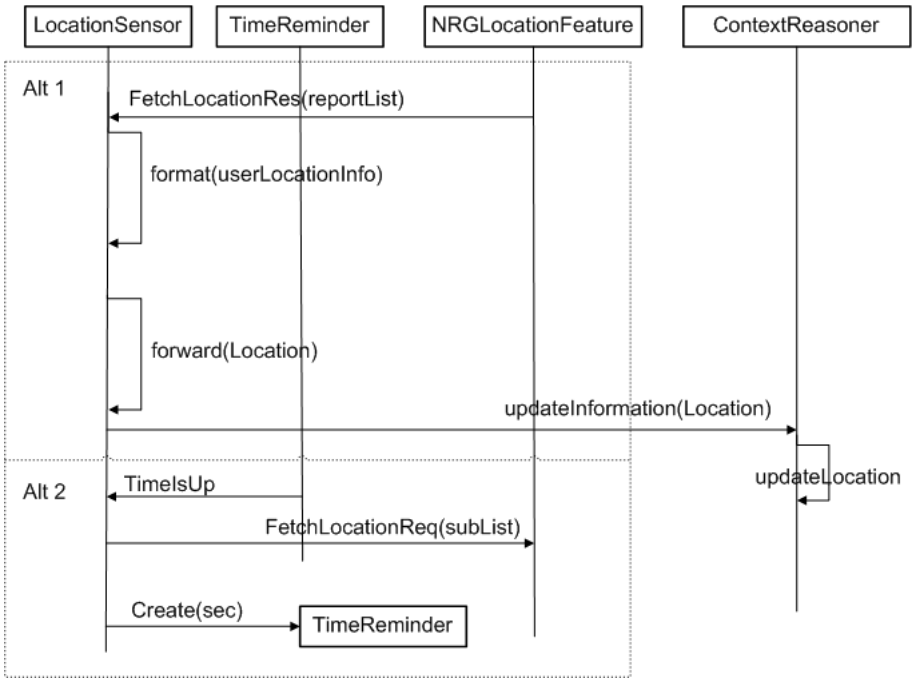

Figure 39: Waiting for location updates

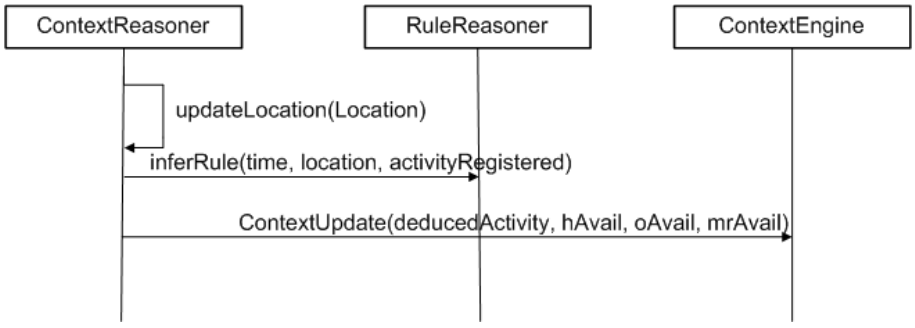

Figure 40: Updating the location

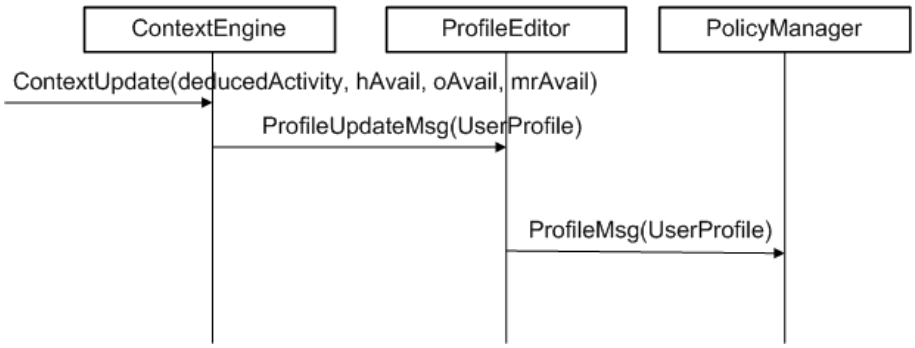

Figure 41: Updating the context

also gets a reference to the call in progress, the call-legs associated with the call and additional information about the notification. The additional information is used to retrieve the address of the destination (callee) and the source(caller). The call processor then spawns a new thread that requests the NRGCallControlFeature to handle the call.

When the NRGCallControlFeature is requested to handle a call, it temporarily stores the information about the call (the caller, the call and the call leg), requests the CallController to find a route and awaits the CallController to return a route for the call. Based on the route returned from CallController, the NRGCallControlFeature requests the NRGCCProcessor to route the call and the call legs. Finally, the NRGCCProcessor is requested to deassign the call and the information that was stored about the call (the caller, the call and the call leg) is removed. When the call is deassigned, it is not released, but simply forwarded into the network by letting the core network handle the call.

When the CallController is requested to calculate the route of a call, it requests the PolicyManager to check whether or not there exist any policies, given the type of caller (work- or family-related). The PolicyManager then requests the PolicyStore to return all the policies for the requested role (cell phone). After receiving all the policies, the PolicyManager checks the condition of each policy to see if it matches the current secondary context information in the UserProfile and the current type of caller. For a detailed presentation of how the policy checking works, please see [48].

When the policy check is performed, the PolicyManager returns a PolicyCheckOKMsg to the CallController with directions on which role that is to be played (where to route the call). This is illustrated in Figure 43. Since the role to be played is a string representing the destination name, the CallController needs to map this to the destination phone number (see Appendix A.10).

If an error occurred while checking the policies, the PolicyManager will return a PolicyCheckNOKMsg. Upon receiving this, the CallController will forward the call to the original destination (the cell phone of the user).

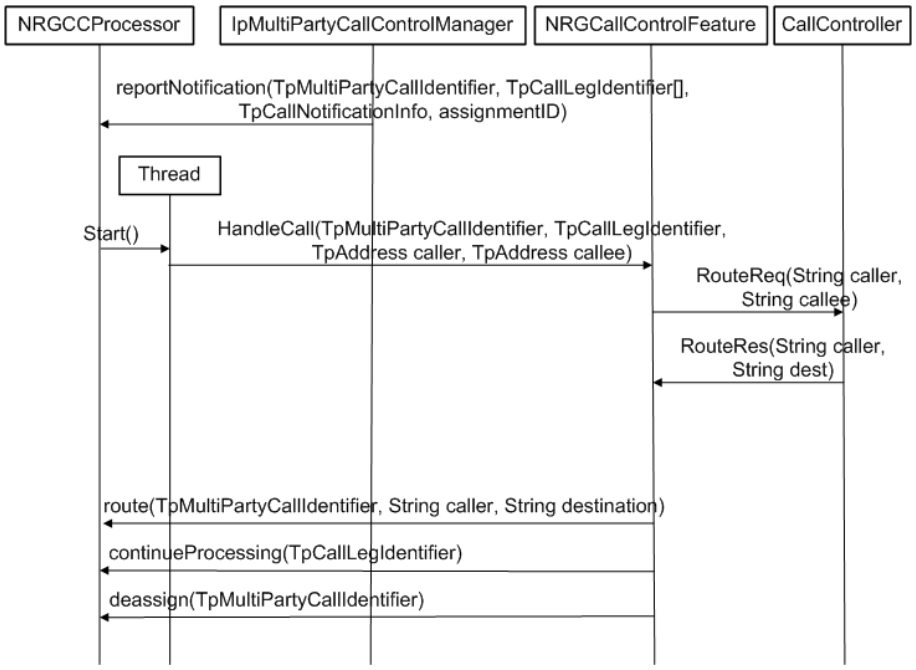

Figure 42: Receiving call notification from the NRG

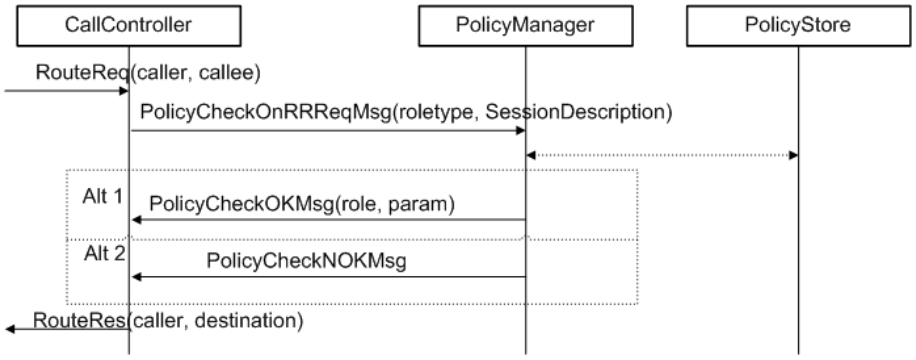

Figure 43: Handling route requests

5.5 Evaluation

This section will present an evaluation of the implementation of the demonstrator service. The evaluation will be based on how it is realised (design), limiting aspects of it and challenges and problems that were encountered when implementing it.

5.5.1 The design

This section evaluates the implementation of the CBCC service with respect to how it is designed. Key issues are how it handles exceptional scenarios, how well it scales and how the three levels of context reasoning are solved.

Exceptional scenarios First of all, every time any information is requested from the OWL model, the ContextReasoner needs to interact with the OntologyOperator. Sometimes, the reasoner might request information from the model that hasn't been set yet. For instance, if the reasoner gets an update about the time, the ContextReasoner needs to retrieve the location from the model to be able to deduce what the secondary information will be. If the location has not been set yet, the OntologyModel needs to be able to cope with this so that it doesn't return a reference to the null value. This is achieved by setting the information that is requested from the model (the location in the example above) equal to "undefined".

When the ContextReasoner gets a time update, there are four main exceptional scenarios that might happen. First, when the current absolute location is not recognized as any of the defined relative locations, the ContextReasoner still needs to define the location for primary reasoning purposes. Therefore, it sets the location equal to "nil", meaning that the observed location is valid but undefined. Second, if the location has not been set in the OWL model yet (equal to "undefined"), the reasoner sets it to be equal to "nil". Since the location in the primary reasoning rules might be defined as nil (meaning that the current absolute location doesn't map to any relative locations), this case will be interpreted as the first exception. This is not optimal, and should be handled in a different manner. Instead of setting it to be equal to "nil", it should be set to something else (e.g. "not set in model") so that this case would not be mixed up with the first one. The first exceptional case is valid and will happen every time the user is located at an undefined location, while the second case is not valid for reasoning purposes. The third exceptional case is when there are no activities registered in the OWL model. This should in theory not happen, since the time fetching is only based on the scheduled activities' beginning and end time. However, as is discussed below in the Limitations section, time could also be based on some user defined policy. Therefore, whenever there are no scheduled activities registered and the ContextReasoner receives a time update, the time should be set to something else than "nil" for primary reasoning (e.g. lunch). The fourth exceptional scenario that might happen is when the sensed

time is different from any of the activities' beginning and end time. The ContextReasoner then sets the time equal to "nil", so that it is able to deduce what the secondary information shall be. However, as was the case above, the time might depend on something else than only the scheduled activity, and should therefore be set to something else than "nil".

When the ContextReasoner gets a location update, there are four exceptional scenarios, of which three of them are equal to the ones that might happen when the time is updated. These are the second, third and fourth exceptional case above. The first exceptional case is when the time has not been set in the OWL model yet. The ContextReasoner then sets the time to be equal to "nil", meaning that the time is currently undefined.

Whenever the ContextReasoner is to deduce what the secondary information will be, it requests the RuleReasoner to infer this based on the current time and location and the scheduled activities that the user has registered. The RuleReasoner then checks whether or not the observed time and location and the registered activities match with any of the secondary rules. If they do, the rule is followed and so the secondary information is set to what the rule defines it to be. If not, all of the secondary information is set equal to "nil", so that it is still possible to perform policy reasoning. After all, there could be a policy with a condition where none of the secondary information is defined (equal to "nil").

When the CallController receives a route request and the callee of the call is not equal to the cell phone number of the user, a route response is simply sent back to the NRGCallControlFeature with the same caller and callee as parameters. In this way, the call is not processed by the policy managing part nor prevented from being routed to the intended destination. This is not likely to happen, since this would mean that it is NRG that made an error. However, it is important to cope with the possibility of the error happening. If the policy checking produces an error, the CallController receives a PolicyCheckNOKMsg. When this happens, the call is routed to the cell phone of the user so that an error made by the service doesn't prevent the user from communicating.

Scalability Scalability is by [13] defined to be "How well a solution to some problem will work when the size of the problem increases". One might say that the scalability of a system is the degree of how well it performs when the task it handles grows in size. For the CBCC application, the task is to perform context based call control; to make a routing decision based on the current context of the user and the user-defined call control policies. So how well does the application handle lots of route requests at the same time?

The NRGCCEdge can handle multiple call notifications at the same time, since whenever a call notification is received by the NRGCCProcessor, it spawns a new thread to make the routing decision in. In that way, it is still able to receive new call notifications at the same time as the first one is being processed by the system. Further on, NRGCallControlFeature is implemented using one state

only (see appendix A.6). This is done so that it is able to receive more requests from the NRGCCProcessor to handle a call at the same time as the first call is being processed by the system. In addition, all the information about the calls that are being processed is being stored so that after a routing decision is made by the CallController, the NRGCallControlFeature is still able to route and process the call. It is vital that the NRGCCEdge is scalable, since it is thought to serve not only one UserAgent but all the UserAgents that need call control.

When the CallController receives a route request, it requests the PolicyManager to decide where the call is to be routed. This decision is based on two things: the current context of and the policies defined by the user. The CallController is implemented to only handle 1 call at a time, meaning that if it receives a new route request while another is being processed the new one will be neglected. However, the routing decision scales well. When the PolicyManager requests all the policies for the given role, the PolicyStore never performs any time consuming tasks (like reading from the disk) since the policies are kept virtually in objects. The list of policies is simply sent back to the PolicyManager. The PolicyManager then goes through all the policies to see if any of them match with the current context of the user. This may be a time consuming task for a large amount (meaning millions) of policies. However, the number of policies registered by a user would most probably not be high at all (compared to the scale that is used in this discussion), due to the resulting complexity of the service. Thus, the route decisioning part of the CBCC application scales well.

In addition to making routing decisions, the system shall also gather context information from multiple sources. The most important class of the ContextEngine is definitively the ContextReasoner. It is responsible for interacting with the ContextEngine (that forwards the context updates to the UserAgent) and all of the sensors in addition to collocating the information received from the sensors. New sensors register with the ContextReasoner at startup telling the reasoner what kind of information they deliver. This is made general so that a large amount of sensors can register with the reasoner. In addition, whenever a sensor wants to send new information to the reasoner, it invokes the general update information method. The method takes the an Object as input so that all the sensors can use the same method. This is possible because the information they provide (e.g. Location) are subtypes of Object. In this way, the ContextReasoner is designed to handle a lot of different context information from different types of sensors.

Context reasoning As is described in section 5.3, there are three levels of reasoning in the CBCC service: ontology reasoning, primary context reasoning and policy reasoning.

When a sensor registers with the ContextReasoner, the ContextReasoner will populate the OWL model with the individuals that are needed for the type of information that the sensor delivers. For instance when the TimeSensor registers, the ContextReasoner populate its OWL model with a Time (the OWL

class Time that is defined for the context ontology) individual. The name of this individual is contained in a separate configuration file (called individualConfiguration.properties and can be found on the electronic attachment). Whenever the reasoner receives a time update, it needs to retrieve the name of the Time individual to be able to update this Time individual in the OWL model. This goes for all the other individuals that are needed too.

This fashion is good because if, for some reason, the name of the individuals were to be changed, the change only needs to be done in the configuration file. However, if new individuals were to be added, they would first need to be added in the configuration file and secondly some code that adds them to the OWL model would have to be written in the ContextReasoner's register code. For instance, if a new relative location (e.g. the gym) were to be added, the name of the Relative location would first have to added to the configuration file (in addition to the Latitude-, Longitude-, Altitude- and their respective Metric individuals) and then some code that added this information to the OWL model would need to be written. Instead of this hard-coded schema, the configuration file could contain information about how many Relative locations there were, and have some general code that added all of them in a loop. In this way, the only thing that needed to be added were the names of the new Relative individuals.

Adding new individuals will also have to be done if the context reasoning of the application were expanded to involve new types of context information (e.g. pressure). This means that new sensors register with the ContextReasoner, and so in addition to adding new individual names to the configuration file - code for this type of information would need to be added in the reasoner's register- and update information code. In addition, the file containing the rules for primary context reasoning would have to be modified to take the new kind of context information into account. Finally, the policy reasoning would also need to be changed so that the user is able to define policies that take the new type of context information into the condition field. This is achieved by adding the new type of context information to the policy vocabulary (in addition to writing code that both adds and evaluate this information in the policies). Please consult [48] for a presentation of how the policy vocabulary is defined.

The three levels of reasoning are very dependant on each other, but there is no functionality for coping with this. If one of them is expanded, there is no functionality that expand the other two. Instead, the other two need to be expanded manually.

5.5.2 Limitations

This section evaluates the implementation of the CBCC service with respect to the limitations of it. In general, most of the implemented sensors have limited functionality. However, their sole purpose is to illustrate how real sensors would behave. The core of the demonstrator is the ContextReasoner and how it manages and refine context information. In addition, due to the limited functionality

of the sensors - quality aspects of the received information is never taken into account. Finally, the sensors and reasoner are assumed to be deployed on the same node. To be able to deploy them on different nodes, they should be implemented on ActorFrame (using the ActorRouter for routing purposes) or using RMI.

The activity sensor The implementation of the activity sensor is only reading the scheduled activities from a file. A full implementation of an activity sensor should provide mechanisms to synchronize calendar information from (mobile) terminals, such as cell phones, PDAs, laptops and desktops. In addition, the activity sensor lacks validation functionality of the received data.

In ServiceFrame, a calendar synchronizer is already developed that synchronizes the information contained in the calendar of mobile terminals with the UserAgent using Sync4j. The vCalendar format is used in the synchronization messages. This module would fit into the ContextEngine as an activity sensor. The only thing that needs to be modified, is that instead of updating the information directly in the UserAgent, the information should be reformatted into a ScheduledActivity object (which has all the fields that is needed to be in accordance with vCalendar).

In addition, NRG contains a PIM Calendar module that can retrieve calendar information from users. However, it does not provide any synchronization capabilities itself - this is assumed to be handled by a synchronization service.

The time sensor The time is totally dependant on the scheduled activities. If the user were to define a policy that forwards all work related calls to the voicemail when he/she is at home at lunch time, there are no functionality for this.

The reason for this is that the time will only be updated when an activity begins or ends, and is thus discrete. There should be some functionality for updating the date and time not only when a scheduled activity begins or ends, but also based on a user defined policy. This can be solved in two ways. In the implementation of the CBCC service, it is assumed that the interaction between the UserAgent and the ContextEngine is one-directional; from the ContextEngine and to the UserAgent. This is obviously not good enough, and so the first solution is to allow the UserAgent to request the ContextEngine to update the date and time in accordance with the user defined policy. The second solution to this is to implement a different type of time sensor that constantly updates the time, so that the CBCCUserAgent constantly knows what the time is (making the time continuous as opposed to discrete).

In addition, time is not a part of the policy reasoning. As described in section 2.6, time is important in context-computing applications because time is used by people when describing their situation. In addition, time is used when people set their preferences (and thus service-policies), and so the time should therefore not only be a part of the information-model but also the policy-reasoning. To

make this work, time should be added as a term in the predicate vocabulary of the context reasoning. In that way, the UserAgent would also know what the time is and not only the ContextEngine.

The location sensor The implementation of the location sensor only fetches the GSM location of the users, which is based on which Base Station that the user is connected to. Needless to say, the granularity of the location results is not very high (200 m - 10 km). However, newer GSM equipment can use more advanced location-measuring algorithms, making the accuracy less than 50 m, see [51].

In the scenario, GSM location sensing would not be enough to be able to represent all the locations. For outdoor locations, GPS should be used as the location method because of its accuracy (10 - 20 m), see [50]. For indoor locations there are several possibilities: infrared (20 cm - 2 m), radio frequency (with a range up to 20 m) or ultrasonic technologies. The most common radio frequency technologies are Radio Frequency ID, Wireless LAN and Bluetooth.

In addition, the location sensor could have some validating functionality that validates the received location with respect to some criteria, making the sensor more intelligent. The criteria could for instance be whether or not the sensed absolute location can be mapped to a known relative location. This would require the sensor to know the name and coordinates of the relative locations - making the sensor capable of performing reasoning. This kind of sensor would have to register with the ContextReasoner declaring that the kind of information it provides is relative location (which is supported by the demonstrator). This is the reason for declaring the Relative location as a subclass of Location instead of declaring it to be a subclass of SecondaryInformation.

The User Profile In the CBCC application, the context information is kept in the user profile. This is not optimal, since the user profile normally contains information of a more permanent character (e.g. contact information, date of birth etc.) and not context information, which varies a lot. Instead of putting the context information in the user profile, each UserAgent should have a context profile containing the context information about the user.

5.5.3 Challenges and problems

As is described in section 5.2.3, the information that is updated with the reasoner by the sensors need to be in suitable formats and the solution that was chosen was to have value objects containing the context information. In this way, it is possible to generalize the interaction between the sensors and the reasoner. One value class is needed per ontology class. Protégé-OWL actually contains a functionality for generating Java classes based on the ontology that is developed, and it was initially thought that these classes could be used as value classes. However, they all extend DefaultRDFIndividual, which is a class

that implements RDFIndividual (see Appendix D.4 for an overview of the most important interfaces in the model package of the Protégé-OWL API). This will not work, because all of these classes cannot be instantiated without being having a model (the JenaOWLModel, see section 5.2.3) to be referenced to, which is not possible for the demonstrator - the sensors are assumed to have no information about the model (that contains all the context information of a user). The only option left was therefore to modify the classes that were generated by Protégé into not implementing or extending any classes from the Protégé-OWL API. In this way, the only hierarchy (in terms of sub- and super-classes) is the one imposed by the hierarchy in the ontology (e.g. Time is a subclass of ContextInformation).

When the ContextEngine requests its sensors to register with the reasoner (see section 5.4) it needs to make sure that the reasoner has populated its model with all the necessary individuals before it can request the sensors to start fetching information. This was necessary so that when new information is received from any of the sensors and the reasoner tries to update this information in the model, the individuals need to be contained in the model. If they are not, the reasoner assumes that the kind of sensor that provides this kind of information hasn't registered yet and so this kind of information should not be possible to add. This could have been solved in another fashion (e.g. if the individual hasn't been added yet, then simply add it before updating the information received). To be able to be sure that the individuals are populated before any sensors can start fetching information, the ContextEngine needs to wait on an explicit acknowledgement from the LocationSensor. This was not implemented at first, but is necessary because the ActivitySensor and the TimeSensor is implemented in a synchronous fashion while the LocationSensor is implemented in an asynchronous fashion (because the LocationSensor is an actor, see Appendix D.2).

When simulating the service, it is really important to pay attention with the time when making phone calls and changing the location of the subscriber. If the location is not changed at the right time, the policies can not be validated when simulating the demonstrator since the primary and secondary information will not be what they are defined to be in the scenarios. This doesn't mean that there are errors in the demonstrator, it simply means that to be able to validate the context reasoning and policy reasoning it is important (and challenging) to change the location and make the phonecalls in accordance with how they are defined in the simulation scenarios.

The location fetching part of the system uses the GSM location request functionality in NRG. As can be read in [49], there are two ways to implement location fetching: the application requests the location when it is needed (polling) or the application can subscribe to periodic location reports (pushing) from NRG. The latter was first chosen. It turned out that the location report subscription doesn't work with this version of the NRG. Consequently, the former was chosen instead. Because of the nature of the CBCC service, a timer needed to be implemented telling the LocationSensor when (how often) to request location reports from the NRG (as is illustrated in section 5.4). Nevertheless, the code

for requesting location report subscriptions from NRG is kept in the source code, because the location report subscription functionality of NRG may be added in future implementations. If desirable, the reader is encouraged to take a look at the code for this in the attached electronic attachment.

6 Simulation of the demonstrator

The demonstrator service will be validated by setting up some simulation scenarios that are in accordance with the user scenario described in section 5.1.2. Table 3 describes these simulation scenarios, and the screenshots of the results from the simulation in the NRG simulator are pictured below.

Before simulating, the policy- (see appendix A.7) and primary (see appendix A.8) reasoning rules were defined in addition to setting the agenda (see Appendix A.9) of the user. The phonenumbers that were used for the user's cellphone, voicemail, secretary, desk phone and the work- and family-related phonenumbers were also specified (see Appendix A.11). In the maps shown in the screenshots from the simulation the upper-left corner is assumed to be the coordinates for an unspecified location, the upper-right corner is assumed to be the coordinates for the meeting room, the lower left corner is assumed to be the coordinates for the office and the lower-right corner is assumed to be the coordinates for the restaurant at which the user is having lunch.

Nmbr	**Caller**	**Location**	**Scheduled activity**	**Expected result**
1a	Family	Undefined	No	Cellphone
1b	Work	Undefined	No	Secretary
2a	Family	Meeting room	Yes	Voicemail
2b	Work	Meeting room	Yes	Voicemail
3a	Family	Office	No	Desk phone
3b	Work	Office	No	Desk phone
4a	Family	Restaurant	Yes	Cellphone
4b	Work	Restaurant	Yes	Secretary

Table 3: Simulation scenarios

Figure 44: Scenario 1a - Family-related call is made while the subscriber is on his way for work. The call is directed to his cellphone.

Figure 45: Scenario 1b - Work-related call is made while the subscriber is on his way for work. The call is directed to his secretary.simulated in the NRG Simulator

Figure 46: Scenario 2a - Family-related call is made while the subscriber is in the meeting room having a planned meeting with Jane. The call is directed to his voicemail.

Figure 47: Scenario 2b - Work-related call is made while the subscriber is in the meeting room having a planned meeting with Jane. The call is directed to his voicemail.

Figure 48: Scenario 3a - Family-related call is made while the subscriber is in the office and no meeting is scheduled at the current time. The call is directed to his desk phone.

Figure 49: Scenario 3b - Work-related call is made while the subscriber is in the office and no meeting is scheduled at the current time. The call is directed to his desk phone.

Figure 50: Scenario 4a - Family-related call is made while the subscriber is having lunch with his friend at the restaurant. In addition, the lunch was scheduled and stored in the subscriber's agenda. The call is directed to his cellphone.

Figure 51: Scenario 4b - Work-related call is made while the subscriber is having lunch with his friend at the restaurant. In addition, the lunch was scheduled and stored in the subscriber's agenda. The call is directed to his secretary.

7 Discussion

This chapter will present a discussion about the work that has been done in this master's thesis. Central to this is how the presence information and the user preferences and reasoning services have been realised. In addition, a short discussion of how the ontology can be modified during runtime will be given.

7.1 Presence and Availability

The CBCC application never presents the context information (see section 2.4) to the users, it performs an automatic service execution. As was discussed in section 2.5, the communication taking place is not motivated by any of the parties' knowledge of the other party's context. Both 3GPP and IETF have defined their own models of presence information, as was described in section 2.6. It is defined differently in the CBCC application, see section 4.3. The Presence class is composed of having a relative location and an availability (instead of a status variable). A presence-individual for each relative location will be populated in the model, meaning that the relative location for each presence-individual never changes. It is the availability of each presence-individual that changes during the runtime. The other approach is to, at all times, have 1 presence individual (that can have different values for location and availability) describe the current presence of the user. The presence information in the CBCC application is thus of a more static character. The ability and willingness of the user to be reached for communication is expressed using **both** the static presence and the dynamic availability of the user, making it possible for the application (when receiving route requests) to go through all the presence-individuals defined for the user and check where he/she is most available. Based on this evaluation, a route is calculated expressing the presence and availability of the user to the requesting party. In the conceptual models provided by IETF and 3GPP, it is more important to have the presence information (location and status) be more human readable since this information is to be presented to a human. The presence information in the CBCC application is more complex (made up of several presence-individuals, that each has a location and an availability) and not designed for being presented to users. Instead, it is designed to be used in automatic service execution. However, a similar representation that is used by 3GPP and IETF is possible to express (if needed) using the existing presence information format of the CBCC application.

7.2 User Preferences and reasoning services

One of the most challenging issues with pervasive and context-aware computing, is to make the system act in accordance with the preferences of the user. To be able to do this, it needs to both be able to describe the situation of the user as accurate as possible in addition to having policies that are in accordance with the users' preferences given the current context.

In the demonstrator, the context information that is used to model the situation of the user is time, location and scheduled activities. These are important attributes to the context of the user, however other attributes could be used in addition - making the model more complex. The issue with specifying the context of a user is that the amount of attributes that are needed to do this varies a lot. More attributes could always be used to make the model more accurate. However, the complexity-level of the context model should not be too high. The complexity level of the model (and thus the behaviour of the system) doesn't correlate with the usability of the system.

Not all context information is updated automatically - some information can be updated by the users. The modelled context will most probably differ with the user-perceived context a lot. In addition, the system is also doomed to make some wrong decisions now and again. According to [4], it is the end-users that are best suited for specializing context-aware applications to their own needs. Users are not able to set their preferences at runtime in the demonstrator, but should be able to do so. This can be achieved through machine learning techniques (statistical models, Bayesian networks, reinforcement learning), requiring the system to be able to store context information, decisions and feedback in addition to be able to reason on this information.

When both the user and system is able to specify the context and the resulting behaviour, one might ask who is in control? Is it the system that is controlling the user or is it the user that is controlling the system? A prerequisite for any technical service or artifact in general, is that the user is able to understand it. Users need to be able to make up a mental model (not necessarily a complex one) of how the system works to be able to use it. If the system surprises the user with a decision - he/she will not be satisfied since it gets difficult to understand its behaviour. The behaviour of context-aware applications may become unstable because of the frequent context-changes, and this may encourage the user to change the policies of the system - which may lead to even more frequent changes. The result is a service that will not be used. Thus the system need to be able to cope with this, making its behaviour stable enough. This can be achieved by introducing user-modeling, making it possible to differ between an experienced user and a novice and thus react to user-changes differently based on the type of user that is performing the change.

According to [43], rule-based synthesizers require explicit definition of rules by humans. These rules are not flexible enough to adapt to changing user-preferences, because the possibility of ending up with rules that are contradictory to each other becomes large with time. The reasoning used in the CBCC application (service-policies and primary-context reasoning) are written in first order logic. Instead, the demonstrator could make use of temporal logic, fuzzy logic and learning mechanisms.

The fetching of information from the different sensors can be automated more by introducing predictability models of when the specific type of context will change. In the demonstrator, the location is fetched in pre-defined intervals independent of how likely it is that the location has changed since the last time.

This could have been solved in a different fashion, for instance the location could have been fetched more often when there are no scheduled activities registered and the time is between 08:00 and 16:00.

Finally, one of the issues when designing the demonstrator service was whether or not the sensors should be able to perform context reasoning (ontology reasoning or primary context reasoning). This alternative was rejected, because it was assumed that the sensors have no knowledge of the context ontology and should therefore not be able to perform reasoning based on the context information and the ontology. This assumption is too strict. Context-aware applications should be able to interact with not only simple sensors, but intelligent ones too. In addition, the different views of context reasoning are merely views - they are not separate layers. Consequently, the sensors may also be able to perform context reasoning, which means that the context ontology should be able to be shared among different entities. Although context reasoning may be performed by different entities in different places of the service domain, the reasoning can always be divided into ontology reasoning (or context model reasoning), primary context reasoning (or lower level reasoning) and policy reasoning. The different levels may, however, be distributed over the service domain.

7.3 Ontology modification

There is no such thing as a perfect ontology - there are always room for improvements. In addition to changing the user-preferences and policies of the system, it could be desirable to make it possible to change the context ontology during runtime. Although this is not provided by the demonstrator, a discussion of how this could be achieved will be given.

To modify an ontology, there are 5 steps that need to be performed (see Figure 52). First, the structure of the ontology needs to be fetched (contained in the JenaOWLModel, see section 5.2.3). After the modifications are performed, the structure needs to be validated by a reasoner, and depending on the outcome the structure is either stored or the inconsistencies of the structure needs to be resolved.

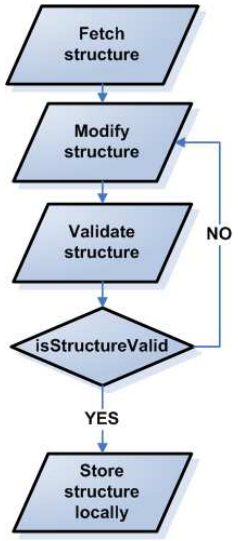

Figure 52: Modifying an ontology

How to resolve the inconsistencies is a vital and tricky question. To be able to do this, the outcome of the reasoner first needs to be interpreted semantically. Secondly, a set of policies that can map the outcome from the reasoner onto some (resolving) actions need to be specified. Finally, the actions need to be performed followed by re-validation of the structure.

Since the ontology only is kept centrally and not shared with other services, no more steps are necessary in the demonstrator. However, if it was shared with other service domains, the ontology needs to be synchronized with these. This introduces synchronization issues, e.g. a prerequisite for updating the structure is that the structure is up to date.

8 Conclusion

This chapter will conclude this master's thesis by providing a short presentation of the achievements related to the objectives and scope. Finally, some issues that could be investigated in future research will be presented.

8.1 Achievements

This master's thesis has investigated principles for context-aware call control. A demonstrator service for context-aware call control was designed, implemented and tested using the NRG Simulator. The types of context information that were to be used in the demonstrator were explicitly specified to be the location, agenda, availability and presence of a user. However, these are not the only context information that could be useful. The call control could be expanded to include new types of context information. To be able to do this, a definition and clear model of context was needed. Because of these issues, 3 sub-goals were defined to scope the tasks of this master's thesis. Below is a description of each.

reflecting the location, the agenda, availability, presence and preferences of the user.

First, context needed to be defined as a term and characterized. Before this could be done, a thorough discussion of what context information is was needed. Context has been defined in many ways, of which many come from HCI-related research. Many previous attempts of defining context merely enumerated examples, which makes the definition too static, non-reusable and strict. Others were too general, by not restricting the definition of context enough. The definition need to restrict the term according to some criteria. The definition that was adopted in this master's thesis was:

> "*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the communication between two users, including the users themselves.*"

Second, an information model of context was needed. This can be (and have been) solved in many ways, both in terms of content and technique. The technique that was adopted in this master's thesis for modeling context was ontology-based. Ontologies were found be well-suited for modeling context because they are good at representing and modeling inter-related and hierarchical information in a formal fashion. The ontology-based context model was implemented using OWL, where the content was based on the definition and characteristics of context. The demonstrator service implements functionality for context management that takes the model into account. It is responsible for monitoring the modelled and sensed information, perform context reasoning and forward the refined context information to the application.

Third, in context-aware applications there are different levels of reasoning taking place. When implementing the demonstrator, the reasoning were divided into 4 different views; the low-level view, the model monitoring, the context monitoring and the application view. A fifth view was added that performed context reasoning and coordinated the tasks in the four other views. This resulted in 3 different levels of reasoning; Ontology reasoning (mapping sensed and modelled information), Primary context reasoning (refining and collocating the received context data) and Policy reasoning (deduce what shall be done, given the context information). Although these levels can be distributed over the service domain, they are believed to apply for context reasoning in any context-aware service domain.

8.2 Future work

One of the most challenging tasks of any service, is to behave in accordance with the current user preferences. Context-aware applications try to optimize their decisions with respect to the users' preferences. This is achieved by learning and describing as much as possible about the context of the users and having user-defined policies representing their preferences. In this master's thesis, the policies are defined as predicates using first order logic. However, not many users would like to specify their preferences using predicates. They would like to be able to specify their preferences using their spoken and/or written language.

The above-mentioned issue could be handled by having a separate system that interprets the user preferences in natural language and maps these down to machine-understandable policies, see Figure 53. It is not possible to add policies during runtime in the demonstrator, thus a protocol for this would have to be developed. The protocol simply needs to make sure that the format of the interpreted policies are in accordance with the format of the policies specified in the policy-configuration file (see Appendix A.7). The rest (context management and reasoning) is supported by the demonstrator service.

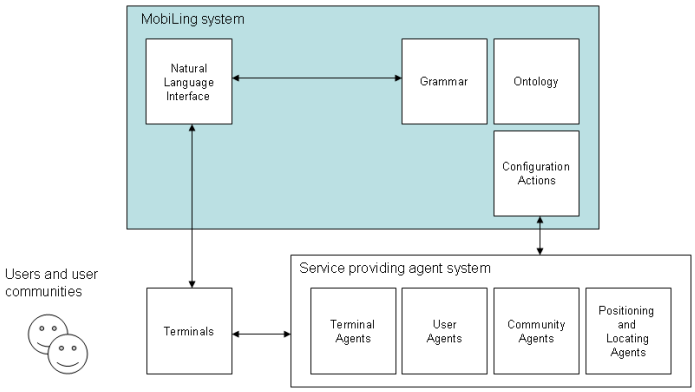

Figure 53: User preferences configuration, obtained from R. Bræk

Further on, when users are able to express their preferences, the system would have to be able to validate these before they are put into effect. This is because the probability of a user specifying preferences that are in conflict is quite high. The validation functionality should be handled by the application offering the service and not the natural language interpreting system, since this would force the natural language-interpreting system to have a list of all the policies for all the services that it manages. With the former approach, the same information is not kept two different places in the same system domain, which is preferable.

References

[1] A. K. Dey and G. D. Abowd: Towards a Better Understanding of Context and Context-Awareness (2000)
(Available at: http://www.csse.monash.edu.au/courseware/cse5610/Students-only/Readings/dey-abowd-99.pdf)

[2] S. Akselsen, W. Finnset, J. Grav, B. Kassah, F. Kileng: MOBIKON - Mobile Tjenester og kontekst. Telenor FoU Scientific Document (2002)
(Available at: http://www.telenor.com/rd/pub/not02/N_17_2002.pdf)

[3] J. Floch, S. Hallsteinsen, A. Lie and H. I. Myrhaug: A Reference Model for Context-Aware Mobile Services. SINTEF Telecom and Informatics (2001)
(Available at: http://folk.uio.no/nik/2001/06-floch.pdf)

[4] A. K. Dey: Understanding and Using Context (2001)
(Available at: http://citeseer.ist.psu.edu/cache/papers/cs/17489/http:zSzzSzwww.cc.gatech.eduzSzfcezSzctkzSzpubszSzPeTe5-1.pdf/dey01understanding.pdf)

[5] G. Chen and D. Kotz: A Survey of Context-Aware Mobile Computing Research (2000)
(Available at: http://citeseer.ist.psu.edu/cache/papers/cs/18650/ftp:zSzzSzftp.cs.dartmouth.eduzSzTRzSzTR2000-381.pdf/chen00survey.pdf)

[6] A. Schmidt, M. Beigl, and H. W. Gellersen: There is more to Context than Location (1998)
(Available at: http://www.comp.lancs.ac.uk/~albrecht/pubs/pdf/schmidt_cug_elsevier_12-1999-context-is-more-than-location.pdf)

[7] B. Kokinov: A Dynamic Approach to Context Modeling (1995)
(Available at: http://citeseer.ist.psu.edu/kokinov95dynamic.html)

[8] B. Schilit, N. Adams, R. Want: Context-Aware Computing Applications (1994)
(Available at: http://citeseer.ist.psu.edu/cache/papers/cs/16662/ftp:zSzzSzftp.cse.ucsc.eduzSzpubzSzwmc-94zSzschilit.pdf/schilit94contextaware.pdf)

[9] J. Grudin: Desituating Action: Digital Representation of Context (2001)
(Available at: http://www.leaonline.com/doi/abs/10.1207/S15327051HCI16234_10)

[10] D. Salber, A. K. Dey, G. D. Abowd: The Context Toolkit: Aiding the Development of Context-Enabled Applications. Proceedings of CHI'99, Pittsburgh, ACM Press (1999)
(Available at: http://citeseer.ist.psu.edu/

cache/papers/cs/17489/http:zSzzSzwww.cc.gatech.
eduzSzfcezSzcontexttoolkitzSzpubszSzchi99.pdf/salber99context.
pdf)

[11] A. K. Dey, G. D. Abowd, D. Salber: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications (2001)
(Available at: http://citeseer.csail.mit.edu/
cache/papers/cs/32861/http:zSzzSzwww.cc.gatech.
eduzSzfcezSzctkzSzpubszSzHCIJ16.pdf/dey01conceptual.pdf)

[12] A. K. Dey, G. D. Abowd: The Context Toolkit: Aiding the Development of Context-Aware Applications (1999)
(Available at: http://citeseer.ist.psu.edu/cache/papers/cs/13819/
http:zSzzSzwww.cs.washington.eduzSzsewpczSzpaperszSzdey.pdf/
dey99context.pdf)

[13] The Free On-line Dictionary of Computing http://foldoc.org/

[14] P. Debaty, D. Caswell: Uniform Web Presence Architecture for People, Places, and Things. Internet and Mobile Systems Laboratory HP Laboratories Palo Alto HPL-2000-67. (June, 200)
(Available at: http://www.hpl.hp.com/techreports/2000/
HPL-2000-67.pdf)

[15] P.J. Brown: The Stick-e Document: a Framework for Creating Context-Aware Applications (1996)
(Available at: http://cajun.cs.nott.ac.uk/compsci/epo/papers/
volume8/issue2/2point1.pdf)

[16] D. Franklin, J. Flaschbart: All Gadget and No Representation Makes Jack a Dull Environment (1998)
(Available at: http://www.infolab.northwestern.edu/infolab/
downloads/papers/paper10072.pdf)

[17] A. K. Dey, G. D. Abowd, A. Wood: CyberDesk: A Framework for Providing Self-Integrating Context-Aware Services (1998)
(Available at: http://citeseer.ist.psu.edu/cache/papers/cs/17489/
http:zSzzSzwww.cc.gatech.eduzSzfcezSzctkzSzpubszSzKBS11-1.pdf/
dey98cyberdesk.pdf)

[18] R. Hull, P. Neaves, J. Bedford-Roberts: Towards Situated Computing (1997)
(Available at: http://www1.cs.columbia.edu/graphics/courses/
mobwear/resources/hull-iswc97.pdf)

[19] J. Pascoe: Adding Generic Contextual Capabilities to Wearable Computers (1998)
(Available at: http://www1.cs.columbia.edu/graphics/courses/
mobwear/resources/pascoe-iswc98.pdf)

[20] T. Rodden, K. Cheverst, K. Davies, A. Dix: Exploiting Context in HCI Design for Mobile Systems (1998)
(Available at: http://citeseer.ist.psu.edu/cache/papers/cs/2480/ftp:zSzzSzftp.comp.lancs.ac.ukzSzpubzSzmpgzSzMPG-98-23.pdf/rodden98exploiting.pdf)

[21] A. Ward, A. Jones, A. Hopper: A New Location Technique for The Active Office (1997)
(Available at: http://citeseer.ist.psu.edu/cache/papers/cs/26981/http:zSzzSzwww-2.cs.cmu.eduzSzafszSzcs.cmu.eduzSzuserzSzsatyazSzWebzSzMCSALINKzSzPAPERSzSzward97.pdf/ward97new.pdf)

[22] Merriam-Webster's Collegiate Dictionary http://www.m-w.com/

[23] N. Ryan, J. Pascoe, D. Morse: Enhanced Reality Fieldwork: the Context-Aware Archaeological Assistant (1997)
(Available at: http://www.cs.kent.ac.uk/projects/mobicomp/Fieldwork/Papers/CAA97/ERFldwk.html)

[24] A.K. Dey: Providing Architectural Support for Building Context-Aware Applications. Ph.D thesis 2000.
(Available at: www.cc.gatech.edu/fce/ctk/pubs/dey-thesis.pdf)

[25] K. Henrichsen, J. Indulska, A. Rakotonirainy: Modeling context information in Pervasive Computing Systems. In Proceedings Pervasive 02 - Zurich August 2002 Springer Verlag, LNCS.
(Available at: http://diuf.unifr.ch/softeng/seminars/SE2003/resources/Pervasive2002.pdf)

[26] P. Nurmi, P. Floréen: Reasoning in Context-Aware Systems (Nov. 2004)
(Available at: http://www.cs.helsinki.fi/u/ptnurmi/papers/positionpaper.pdf)

[27] Ard-Jan Moerdijk and Lucas Klostermann: Opening the networks with OSA/Parlay: Standards and aspects behind the APIS Network IEEE, Volume: 17, Issue:3, May-June 2003

[28] Ericsson Network Resource Gateway Programmers Guide by Ericsson AB, 2003, 2005

[29] J. Scourias: Overview of the Global System for Mobile Communications, University of Waterloo, May 19 1995

[30] The documentation of Ericsson Network Resource Gateway Simulator by Ericsson

[31] The NRG SDK, downloaded from Ericsson's Mobility World http://www.ericsson.com/mobilityworld/sub/open/technologies/parlay/docs/parlay_cd

[32] G. Melbye: ActorFrame Developer's guide, NorARC, ARTS version P1

[33] A. Herstad, G. Melby: ServiceFrame version 1, ARTS

[34] UML 2.0 Superstructure Specification published by the OMG group

[35] M. Uschold & M. Gruninger: Ontologies: Principles, Methods and Applications, Knowledge Engineering Review, Volume 11 Number 2, June 1996

[36] T. Gu, X. H. Wang, H. K. Pung, D. Q. Zhang: An Ontology-based Context Model in Intelligent Environments (2004)
(Available at: http://www.comp.nus.edu.sg/~gutao/gutao_NUS/CNDS2004_gutao.PDF)

[37] T. Gu, H. K. Pung, D. Q. Zhang: A service-oriented middleware for building context-aware services (2004).
(Available at: http://www.comp.nus.edu.sg/~gutao/gutao_NUS/SOCAM_gutao.pdf)

[38] T. Strang, C. Linnhoff-Popien, K. Frank: CoOL: A Context Ontology Language to enable Contextual Interoperability (2003).
(Availale at: http://springerlink.metapress.com/media/3djnnnyuqkqn5bm4ek0j/contributions/d/j/n/h/djnhu2gvpv7cqltv.pdf)

[39] T. Strang, C. Linnhoff-Popien, K. Frank: Applications of a Context Ontology Language (2003).
(Available at: http://citeseer.ist.psu.edu/cache/papers/cs/30448/http:zSzzSzwww.kn.op.dlr.dezSz~strangzSzpaperzSzsoftcom2003zSzSoftCom2003CameraReadyVersion.pdf/strang03applications.pdf)

[40] D. Preuveneers, J. Van den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers and K. De Bosschere: Towards an extensible context ontology for Ambient Intelligence (2004).
(Available at: http://research.edm.luc.ac.be/tclerckx/eusai2004.pdf)

[41] D. Preuveneers and Y. Berbers: Semantic and syntactic modeling of component-based services for context-aware pervasive systems using OWL-s (2005).
(Available at: http://www.cs.kuleuven.ac.be/~davy/publications/mcmp05.pdf)

[42] H. Chen and T. Finin: An Ontology for Context Aware Pervasive Computing Environments (2003)
(Available at: http://www.cs.vu.nl/~heiner/IJCAI-03/Papers/Chen.pdf)

[43] A. Ranganathan, R. H. Campbell: A Middleware for Context-Aware Agents in Ubiquitous Computing Environments (2003).
(Available at: http://choices.cs.uiuc.edu/~ranganat/Pubs/MiddlewareForContext-FinalVersion.pdf)

[44] A. Ranganathan, R. E. McGrath, R. H. Campbell, M. D. Mickunas: Ontologies in a Pervasive Computing Environment (2003).
(Available at: http://www.cs.vu.nl/~heiner/IJCAI-03/Papers/Ranganathan.pdf)

[45] H. Chen, F. Perich, T. Finin, A. Josh: SOUPA: Standard Ontology for Ubiquitous and Pervasive Application (2004).
(Available at: http://ebiquity.umbc.edu/_file_directory_/papers/105.pdf)

[46] The CoBrA website http://cobra.umbc.edu/about.html

[47] M. J. Woolridge and N. R. Jennings: Intelligent agents: Theory and practice. Knowledge Engineering Review, 10(2): 115-152, June 1995.
(Available at: http://www.csc.liv.ac.uk/~mjw/pubs/ker95.pdf)

[48] H. N. Castejón Martinez: Policies in ServiceFrame
(Available in the electronic attachment)

[49] Ericsson H-OSA Interface Specification Mobility Management, User Location 2003-10-06 revision A. (Available from the NRG SDK)

[50] Assistant Secretary of Defence for Command, Control, Communications, and Intelligence: Global Positioning System, Standard Positioning Service, Performance Standard. October, 2001.
(Available at:http://www.navcen.uscg.gov/gps/geninfo/2001SPSPerformanceStandardFINAL.pdf)

[51] Levijoki, S.: Privacy vs Location Awareness, Tik-110.501 Seminar on Network Security.
(Available at: http://www.tml.tkk.fi/Opinnot/Tik-110.501/2000/papers/levijoki.pdf)

[52] OWL Web Ontology Language Reference, W3C Recommendation. 10 February, 2004
(Available at: http://www.w3.org/TR/owl-ref/)

[53] M. Horridge, H. Knublauch, A. Rector, R. Stevens, C. Wroe: A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools Edition 1.0, University of Manchester & Stanford University. August 27, 2004.
(Available at: http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf)

[54] H. Knublauch: The Protégé-OWL API - Programmer's Guide. June 02, 2005.
(Available at: http://protege.stanford.edu/plugins/owl/api/guide.html)

[55] Extensible Markup Language (XML) 1.1, W3C Recommendation. 04 February, 2004 (edited in place 15 April 2004).
(Available at: http://www.w3.org/TR/2004/REC-xml11-20040204/)

[56] XML Schema Part 0: Primer Second Edition, W3C Recommendation. 28 October, 2004.
(Available at:http://www.w3.org/TR/xmlschema-0/)

[57] RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation. 10 February 2004.
(Available at: http://www.w3.org/TR/rdf-schema/)

[58] RDF Primer, W3C Recommendation. 10 February 2004.
(Available at: http://www.w3.org/TR/rdf-primer/)

[59] D. Roman, U. Keller, H.Lausen, J. de Buijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler and D. Fensel: Web Service Modeling Ontology. Applied Ontology 1 (2005) 77-106, IOS Press.

[60] Semantic Web Services Framework (SWSF) Overview, W3C Member Submission. 9 September 2005.
(Available at: http://www.w3.org/Submission/SWSF/)

[61] RFC 3863 - Presence Information Data Format (PIDF)

[62] 3GPP TS 22.141: 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Presence Service; Stage 1 (Release 7). 2005-12.

[63] US Marine Corps: Marine Artillery Survey Operations. 13. February 2004 (Available at: http://www.tpub.com/content/USMC/mcwp3167/css/mcwp3167_45.htm)

[64] Akogrimo D4.2.2 Final Integrated Service Design and Implementation Report Version 1.0. 31.10.05.

[65] Notes on the History of Ontology http://www.formalontology.it/history.htm

[66] T. Berners-Lee: Semantic Web Road Map. September 1998.
(Available at: http://www.w3.org/DesignIssues/Semantic.html)

[67] T. Berners-Lee: Semantic Web on XML. XML 2000 Washington DC.
(Available at: http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide1-0.html)

[68] W3C: Semantic Web.
(Available at: http://www.w3.org/2001/sw/)

[69] I. Horrocks, B. Parsia, P. Patel-Schneider, J. Hendler: Semantic Web Architecture: Stack or Two Towers? 2005.
(Available at: http://www.cs.man.ac.uk/~horrocks/Publications/download/2005/HPPH05.pdf)

List of Figures

List of Tables

List of Abbreviations

CBCC - Context Based Call Control
NRG - Network Resource Gateway
API - Application Programming Interface
MPCC - Multi-Party Call Control
OSA - Open Service Access
H-OSA - High level OSA
SDK - Software Development Kit
3GPP - 3rd Generation Partnership Project
ETSI - European Telecommunication Standardization Institute
ITU-T - International Telecommunication Union Telecom Standardization Sector
CORBA - Common Object Request Broker Architecture
GSM - Global System for Mobile communications
XML - eXtensible Markup Language
XSD - XML Schema
OWL - Web Ontology language
RDF - Resource Description Framework
RDFS - RDF Schema
WSMF - Web Service Modeling Framework
WSMO - Web Service Modeling Ontology
WSML - Web Service Modeling Language
SWSF - Semantic Web Services Framework
SWSO - Semantic Web Services Ontology
SWSL - Semantic Web Services Language

List of Definitions

Caller - the user or person that is trying to make a phonecall

Callee - the user (person) that is being called, that the phonecall is for

Call object - A relation between a number of parties. It relates to the entire call view from the application

Call leg object - Represents a logical association between a call and an address. The relationship includes at least the signalling relation with the party. The relation with the address is only made when the leg is routed. Before that the leg object is IDLE and not yet associated with the address.

Address - Logical representation of a party in the call

E.164 phone number - E.164 numbers are globally unique, language independent identifiers for resources on Public Telecommunication Networks that can support many different services and protocols. E.164 numbers are used to identify e.g. ordinary phones, fax machines, pagers and data modems.

Policy - a definite course or method of action selected from among alternatives and in light of given conditions to guide and determine present and future decisions

Context - Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the communication between two users, including the users themselves.

Appendix A - Design and Implementation details

This appendix will provide the reader with some of the details of the implementation of the demonstrator service. First, the state machines of the actors are presented (as screenshots from the Ramses modelling tool). Further on, the configuration files for the policies, the primary reasoning rules, the user's scheduled activities and the phone numbers used in the simulation are presented. Finally, the code for calculating the route of a call after the policy check is performed successfully.

A.1: State diagram for actor CallController

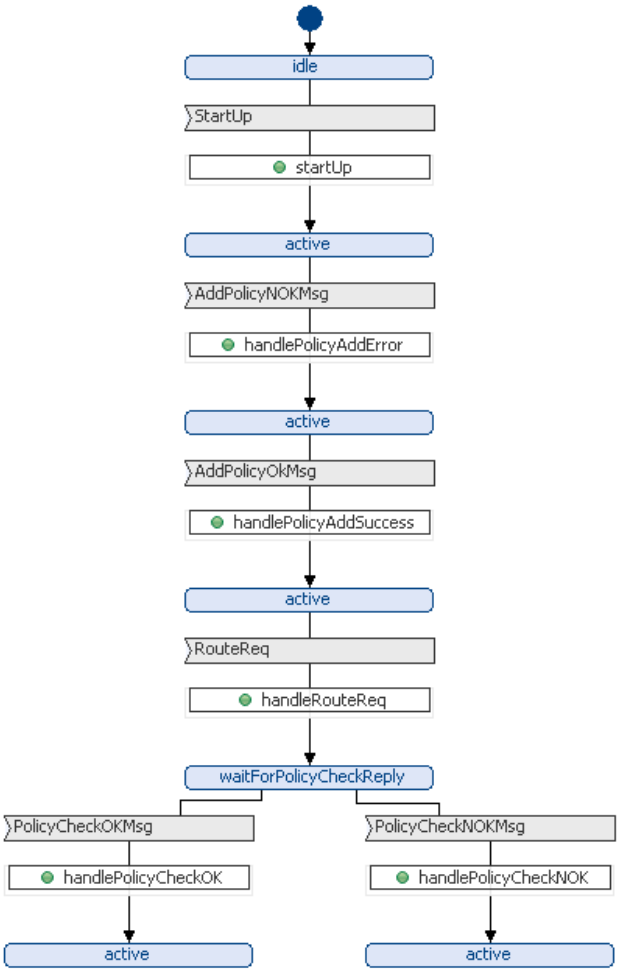

Figure 54: State diagram for actor CallController

A.2: State diagram for actor CBCCUserAgent

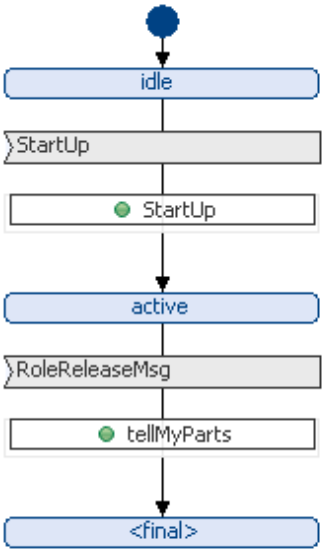

Figure 55: State diagram for actor CBCCUserAgent

A.3: State diagram for actor ContextEngine

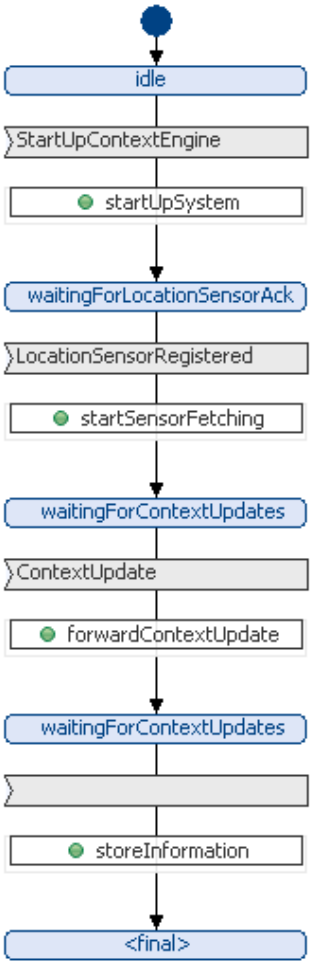

Figure 56: State diagram for actor ContextEngine

A.4: State diagram for actor LocationSensor

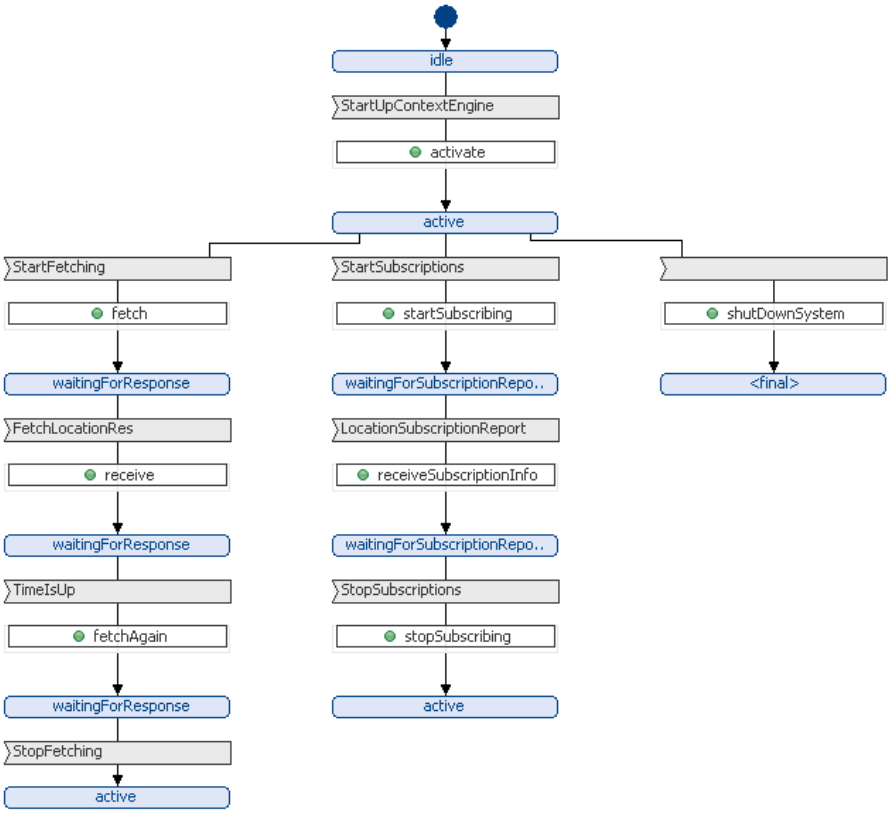

Figure 57: State diagram for actor LocationSensor

A.5: State diagram for actor NRGLocationFeature

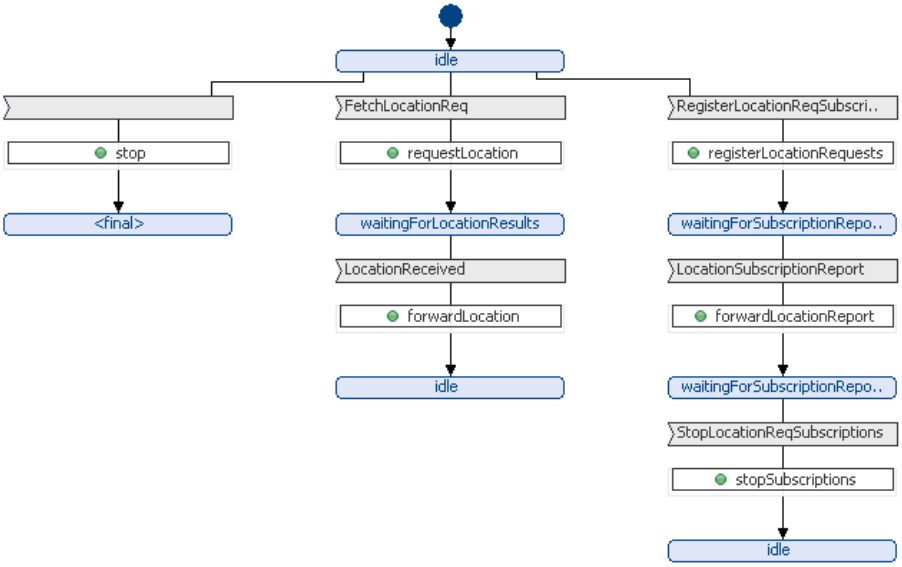

Figure 58: State diagram for actor NRGLocationFeature

A.6: State diagram for actor NRGCallControlFeature

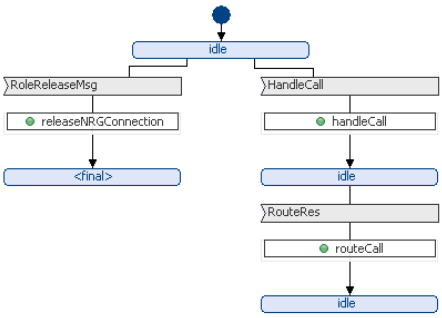

Figure 59: State diagram for actor NRGCallControlFeature

A.7: The policy definitions (policies.properties)

This file contains the policies that will be used in the simulation of the CBCC service.

numberOfPolicies=8

!! Important to follow this scheme when modifying or adding policies

!! The availabilities need to be on this format: prefix_{low, medium, high}, where

!! the prefix names where the availability exists(o=office, mr=meeting room, h=home)

1subjectRole=cell

1deducedActivity=nil

1homeAvailability=h_low

1officeAvailability=o_low

1meetingRoomAvailability=m_low

1callType=family

1alternativeRole=cell

2subjectRole=cell

2deducedActivity=nil

2homeAvailability=h_low

2officeAvailability=o_low

2meetingRoomAvailability=m_low

2callType=work

2alternativeRole=secretary

3subjectRole=cell

3deducedActivity=meeting

3homeAvailability=h_low

3officeAvailability=o_low

3meetingRoomAvailability=m_medium

3callType=family

3alternativeRole=voicemail

4subjectRole=cell

4deducedActivity=meeting

4homeAvailability=h_low

4officeAvailability=o_low

4meetingRoomAvailability=m_medium

4callType=work

4alternativeRole=voicemail

5subjectRole=cell
5deducedActivity=working
5homeAvailability=h_low
5officeAvailability=o_high
5meetingRoomAvailability=m_low
5callType=family
5alternativeRole=desk
6subjectRole=cell
6deducedActivity=working
6homeAvailability=h_low
6officeAvailability=o_high
6meetingRoomAvailability=m_low
6callType=work
6alternativeRole=desk
7subjectRole=cell
7deducedActivity=lunch
7homeAvailability=h_low
7officeAvailability=o_low
7meetingRoomAvailability=m_low
7callType=family
7alternativeRole=cell
8subjectRole=cell
8deducedActivity=lunch
8homeAvailability=h_low
8officeAvailability=o_low
8meetingRoomAvailability=m_low
8callType=work
8alternativeRole=secretary

A.8: The primary reasoning rules (secondaryInformationRules.properties)

!! Contains the rules for mapping between primary information and secondary information

numberOfRules = 4

1Time=officeTime

1Location=nil

1ScheduledActivity=false

1DeducedActivity=nil

1homeAvailability=h_low

1officeAvailability=o_low

1meetingRoomAvailability=m_low

2Time=activityTime

2Location=meetingRoom

2ScheduledActivity=true

2DeducedActivity=meeting

2homeAvailability=h_low

2officeAvailability=o_low

2meetingRoomAvailability=m_medium

3Time=officeTime

3Location=office

3ScheduledActivity=false

3DeducedActivity=working

3homeAvailability=h_low

3officeAvailability=o_high

3meetingRoomAvailability=m_low

4Time=activityTime

4Location=restaurant

4ScheduledActivity=true

4DeducedActivity=lunch

4homeAvailability=h_low

4officeAvailability=o_low

4meetingRoomAvailability=m_low

A.9: The scheduled activities for the user (userScheduledActivites.properties)

numberOfMeetings=3

!! NBNB!!!!

!! IMPORTANT TO FOLLOW THIS SCHEMA WHEN USING TIME AND DATE: XX.XX:XX XX:XX

!! where the format is day.month.year and the time is written with 24 hours and not AM/PM

!! This first one is only to set the beginTime of the application and so the begin/IsDone times

!! should be equal so that the officetime is the one that begins - in accordance with the User scenario

1Summary=null

1Description=null

1BeginsAtTimeMetricValue=22.05.06 15:51

1IsDoneAtTimeMetricValue=22.05.06 15:51

1Location=null

1Participant=null

1Category=null

1Priority=null

2Summary=Meeting with Jane

2Description=Meeting with Jane to discuss something. Remember to bring computer

2BeginsAtTimeMetricValue=22.05.06 15:52

2IsDoneAtTimeMetricValue=22.05.06 15:53

2Location=Office

2Participant=Jane

2Category=Business

2Priority=Medium

3Summary=Lunch with friends

3Description=Lunch with friends. Remember to bring photos

3BeginsAtTimeMetricValue=22.05.06 15:55

3IsDoneAtTimeMetricValue=22.05.06 15:56

3Location=Restaurant

3Participant=null

3Category=Private

3Priority=Low

A.10: Calculating the route after the policy check was successful

```
public static void handlePolicyCheckOK(PolicyCheckOKMsg signal,
CallControllerSM asm) {
String roleToPlay = signal.roleType;
String destination = null;
String caller = asm.currentCaller;
if (roleToPlay.equals("cell")) {
destination = asm.cellPhone;
} else if (roleToPlay.equals("secretary")) {
destination = asm.secretary;
} else if (roleToPlay.equals("voicemail")) {
destination = asm.voicemail;
} else if (roleToPlay.equals("desk")) {
destination = asm.deskPhone;
} else {
// an error occurred because the role( or destination) was not recognized
//need to forward the call to the original destination
destination = asm.currentCallee;
}
RouteRes res = new RouteRes(caller, destination);
asm.sendMessage(res, asm.myCallControlFeature);
}
```

A.11: The configuration of the phonenumbers used in the simulation (defaultCBCCProperties.properties)

terminalAgentPhonenumber=111

deskPhone = 112

userVoicemail=113

secretary=200

familyCaller=300

workCaller=400

Appendix B - Developer's guide

This appendix provides information on how to simulate the demonstrator service. First, the structure of the electronic attachment will be presented followed by an installation guide of the required tools and libraries. Finally, some notes on simulating the demonstrator service is given.

B.1 - Structure of the electronic attachment

This section presents the structure and the content of the attachment. Only the most important files (in terms of relevancy to the simulation and modification of the system) will be presented. The files that are not mentioned should not be modified.

1. **ontology**/

   - finalContextOntology.owl - contains the context ontology developed in this master's thesis, written in OWL
   - finalContextOntology.pprj - contains the context ontology Protégé project to be used by Protégé to view the context ontology

2. **demonstrator**/

CBCC/ - contains the Ramses model files of the CBCC module of the system. The CBCC module includes the CBCCUserAgent except the ContextEngine and the policy managing actors.

   - bin/ - contains the compiled source code of the CBCC module
   - lib/ - contains the necessary jar files for the CBCC module
   - src/ - contains the source code of the CBCC module
   - config.ini - contains the NRG configuration settings for the CBCC module
   - model.uml - contains the Ramses modelling information of the CBCC module
   - defaultCBCCProperties.properties - contains the configuration of the different phonenumbers used during simulation in the NRG Simulator
   - policies.properties - contains the policies

ContextEngine/ - contains the Ramses model files of the ContextEngine. The ContextEngine includes the sensors, the ContextReasoner and the different ontology-related functionality.

- bin/ - contains the compiled source code of the ContextEngine
- lib/ - contains the necessary jar files for the ContextEngine
- src/ - contains the source code of the ContextEngine
- config.ini - contains the NRG configuration settings for the ContextEngine
- model.uml - contains the Ramses modelling information of the ContextEngine
- defaultContextEngineProperties.properties - contains configuration settings for the location sensor
- finalContextOntology.owl - the context ontology written in OWL
- individualConfiguration.properties - contains configuration settings for the OWL individuals
- SecondaryInformationRules.properties - contains the primary reasoning rules
- userConfiguration.properties - contains the configuration settings for the user agent that is the owner of the context model
- userScheduledActivities.properties - contains information about the scheduled activities of the user

exeCBCCcode/ - contains the simulation code for the demonstrator that is generated by Ramses. It contains all the functionality of the demonstrator service.

- bin/ - contains the compiled source code of the demonstrator
- lib/ - contains the necessary jar files for the demonstrator
- src/ - contains the source code of the demonstrator
- config.ini - contains the NRG configuration settings for the demonstrator
- defaultContextEngineProperties.properties - contains configuration settings for the location sensor
- finalContextOntology.owl - the context ontology written in OWL
- individualConfiguration.properties - contains configuration settings for the OWL individuals

- SecondaryInformationRules.properties - contains the primary reasoning rules
- userConfiguration.properties - contains the configuration settings for the user agent that is the owner of the context model
- userScheduledActivities.properties - contains information about the scheduled activities of the user
- defaultCBCCProperties.properties - contains the configuration of the different phonenumbers used during simulation in the NRG Simulator
- policies.properties - contains the policies

3. documentation/

policiesInServiceFrame.doc - Equal to reference [48]: "Policies in ServiceFrame" by Humberto Nicolás Castejón Martínez. This is reference.

B.2 - Installation

To be able to simulate or modify the demonstrator, the following sections provides information about the things that need to be installed.

Install Java 1.4.2

Install Java 1.4.2 (or a later version), available from `http://java.sun.com/j2se/downloads.html`. It is recommended to install the full SDK, as opposed to just the JRE (Java Runtime Environment).

Set JAVA_HOME to to point to the directory containing your Java installation (e.g. "c:\jdk1.4.2"). Note that the directory specification cannot contain any spaces, so if you have installed Java in "Program Files" under Windows, for example, you will need to specify this using the DOS format "PROGRA~1".

Verify your installation by bringing up a shell window and typing "java -version". The response should indicate that 1.4.2 (or later) is installed. For example:

C:\Documents and Settings\Student>java -version

java version "1.5.0_04"

Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_04-b05)

Java HotSpot(TM) Client VM (build 1.5.0_04-b05, mixed mode, sharing)

Install Ericsson's NRG SDK

To be able to utilize the services provided by NRG, you need to download and install the NRG SDK (for this master's thesis, version 4.0 was used). This can be downloaded from Ericsson's Mobility World `http://www.ericsson.com/`

`mobilityworld/sub/open/technologies/parlay/docs/parlay_cd`. Just follow the download- and installation instructions. The SDK also contains the NRG Simulator [30] that simulates the underlying network resources, and thus makes it possible to simulate the demonstrator service.

Protégé and the Protégé-OWL API

To develop the context ontology using OWL, Protégé[7] 3.1 with the OWL plug-in was used as the ontology-editor. It contains all the necessary tools through the plug-in facility, in addition to including the Protégé-OWL API [54]. Protégé and the Protégé-OWL API doesn't need to be installed to be able to run or modify the demonstrator service, since all the necessary resources (contained in jar-files) are included in the demonstrator's lib folder. However, to get a more human-readable presentation of the context-ontology that was developed, Protégé-OWL should be used.

B.3 - Simulation

To simulate the demonstrator service, (after installing Java and the Ericsson's SDK) the scheduled activities of the user need to be modified to the time-range of the simulation scenarios. This is achieved by modifying the activity property file (see Appendix A.9). In addition, when configuring the NRG simulator with telephones and their phonenumbers, the reader should consult the file containing this configuration (see Appendix A.11). It is also vital that the configuration of the different relative location-coordinates are followed. This file is described in Appendix B.1. The reader is on a general basis encouraged to consult this section when simulating the demonstrator service.

[7] Available at: `http://protege.stanford.edu`

Appendix C - Relevant Technologies

This appendix will present the relevant technologies that the demonstrator service interacts with. The demonstrator actually only uses the resource provided by Ericsson's Network Resource Gateway (NRG). However, since NRG is Ericsson's implementation of OSA/Parlay, a brief introduction to OSA/Parlay will also be given.

Appendix C.1 - OSA/Parlay

Below is a brief presentation of OSA/Parlay. It is mostly based on the contents of [27, 28].

C.1.1 - Background

Traditionally, telecommunication networks, applications and services were a part of the network operator's domain. Services and applications were developed using Intelligent Network (IN) technology. This was well suited for simple mass-market and carrier class applications, but with the introduction of mobility and IP in the networks, easy creation and rapid deployment of innovative applications became a challenge beyond the capability of IN.

In 1998, the Parlay group was formed as a response to this trend. It was initiated by a community of operators, IT vendors, network equipment providers and application developers. They wanted to develop APIs that combined the best of the two worlds (telecom and IP), because telecommunication applications and services were traditionally developed using specialized telecom technology. The overall goal was to develop 1 API for 1 developer community. Today, the APIs are standardized by the Joint Working Group, a collaboration between the Parlay Group, 3GPP, ETSI and ITU-T, and the Parlay group has grown twelve-fold.

C.1.2 - Logical architecture

The OSA/Parlay architecture consists of three layers; the Connectivity Layer, the Control Layer and the Service Layer. The OSA/Parlay APIs are located at the service layer, or Service Enabling Layer [27], and they interact with the control layer and the connectivity layer, thereby hiding the network-specific complexities.

To put this in context, let us have a look at GSM. [29] defines the Base Station Subsystem (BTS and BSC) as the Connectivity Layer. This is where the radio link is controlled and coverage for the terminals are provided. The Network Subsystem (MSC, HLR, VLR) is defined as the Control layer, and this is where the signaling takes place. It handles switching of calls and mobility issues, thereby controlling the Connectivity Layer.

The point with the OSA/Parlay Architecture is that as an application developer, you do not need to concern yourself with detailed network complexities. The APIs are located at the Service Enabling Layer, simply offering some of the functionality of the underlying network elements as service features or service capabilities. Below, a brief introduction to the logical architecture (illustrated in Figure 60) of OSA/Parlay is given and it is mostly based on [27]. It consists of Applications, Application Servers (AS), Service Capability Servers (SCS), the OSA/Parlay Framework and core network elements.

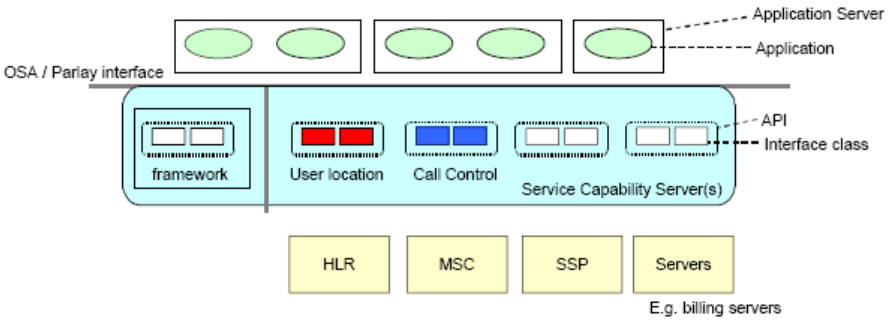

Figure 60: The OSA/Parlay Logical Architecture, taken from [27]

Applications are defined as the client-side implementation of the APIs and they use the capabilities offered through the OSA/Parlay APIs. They are deployed on Application Servers. The Application Server could either be provided by a telecom network operator or a third party company. The SCSs represents the server-side implementations of the APIs and they provide the capabilities that the Applications use. Further on, they communicate with network specific components, e.g. the HLR in GSM, and serves thus as a gateway to the core network. They are logical entities, and could be distributed in the network or implemented directly on a network element, e.g. the MSC. The SCSs are typically provided by network operators. Communication between the Application and the SCSs is done through some sort of middleware infrastructure, e.g. CORBA. The OSA/Parlay Framework provides controlled access to the APIs on the SCSs. Applications that need access to service capabilities in the SCS therefore first need to authenticate themselves and request access for the requested capability from the OSA/Parlay Framework.

Appendix C.2 - Network Resource Gateway

The Ericsson's Network Resource Gateway (NRG) is a framework that provides a set of APIs. It can be seen as Ericsson's implementation of an OSA/Parlay

Gateway, see section C.1.2. Below, a presentation of the NRG is given. Throughout this presentation, the term application and service will be used in the following manner; an application runs on top of the NRG and uses the services within the NRG. Most of the information is taken from [28].

C.2.1 - Overview

The NRG is located at the Service Layer in the OSA/Parlay Architecture as an application gateway. It follows the goal of OSA/Parlay; to simplify the telecom network complexities by hiding the low layers and providing a standardized interface for the application developer. In section C.1.1 it was stated that the OSA/Parlay APIs connect applications with the Service Layer. Figure 61 shows how NRG and the NRG service relate to OSA/Parlay.

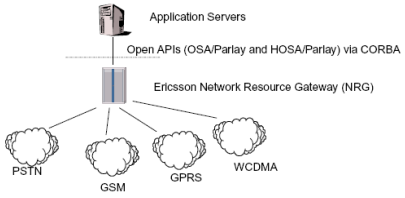

Figure 61: The NRG big picture, taken from [28]

The following things are possible with the OSA/Parlay defined services in the NRG:

- Obtain access to a service by requesting it from the Framework service
- Create or route phone calls using the Multi Party Call Control (MPCC)
- Play announcements and collect digits during a phone call by using the User Interaction (UI) service
- Request the location of mobile phones by using the User Location (UL) service

In addition to the standard OSA/Parlay APIs, the NRG has been enhanced by a superset of these, the High level Open Service Access (H-OSA) APIs. These provide extended services and functionality and each interface specification will extend from the OSA/Parlay specification. The following are possible with the H-OSA defined services in NRG:

- Receive SMS and MMS messages and send SMS, MMS, WAP-push and E-mail messages by using the Messaging service

- Read E-mail by using the Message Retrieval service
- Request the status of mobile phones by using the User Status service
- Handle subscriber contacts by using the Personal Information Management (PIM) Contact service
- Handle subscriber calendars by using the PIM Calendar service

C.2.2 - The NRG Software Development Kit

The above all most important part of the NRG is the SCSs (Service Capability Servers). Note that the NRG is often seen as both the SCSs and the whole NRG product. In a sense, this is correct since the most fundamental part of the NRG is the SCSs. However, the NRG product contains not only the SCSs but also the NRG Software Development Kit (SDK).

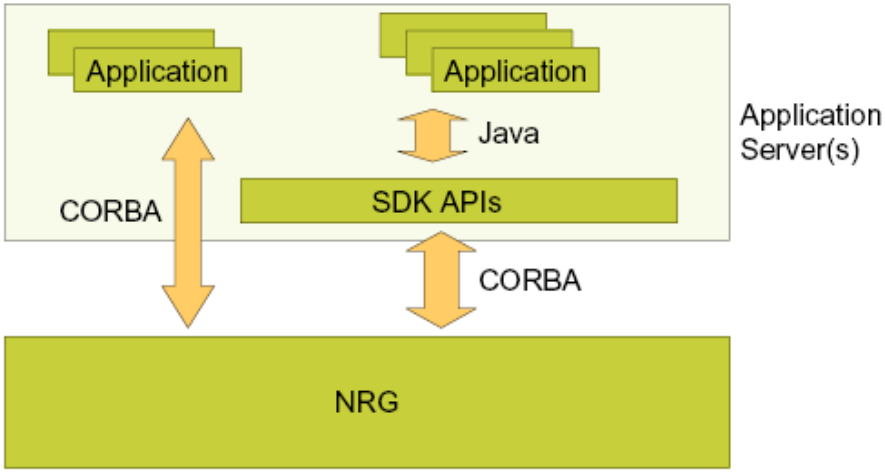

Figure 62: Overview of the NRG SDK, taken from [28]

The SDK offers software libraries to Java application developers so that no detailed CORBA knowledge is needed to use the NRG services. This is done by using one of the features of these software libraries; its CORBA abstraction. Method calls are translated from Java to CORBA and then back again when received from NRG. It also offers functionality to support network redundant NRG systems, that is when two NRG nodes operate in parallel and can take over each other's work. Besides standard content like documentation, the SDK contains an automated test tool, making it easy for the developer to test an application against a simulated NRG with automatically generated

traffic. In addition, it contains the NRG Simulator. The Simulator simulates the NRG node and the underlying network (with resources). It has a graphical user interface where peripherals, e.g. phones, are simulated facilitating testing of services in a user-friendly and dynamic environment. From an application's point of view, it is transparent whether it is communicating with an actual NRG node or the NRG Simulator. For a complete reference to the NRG SDK and its Simulator, please refer to [31, 30].

C.2.3 - The Framework service

NRG contains a Framework and one or more services (SCSs), see Figure 63. Services can be seen as the functional entity providing a standard interface toward an application. So by this definition, the Framework can be seen as a NRG service. The big difference between the Framework service and the other services is that the Framework is mandatory. It acts in many ways as the OSA/Parlay framework does (see section C.1.2) namely as a gatekeeper to the NRG services. When an application wants to use a NRG service, the application first has to request it from the Framework.

C.2.4 - The NRG enabled Application Life Cycle

The application life cycle looks as follows:

1. Obtain access to the Framework
2. Obtain access to needed services
3. Handle multiple transactions (e.g. phone calls, messages) by using the services
4. Release the used services
5. End the access to the Framework

When an application requests access to services, it can get multiple instances of the same service type. By specifying service properties, the right instance of a service type can be selected.

The application can be located in the operator domain as well as somewhere on the Internet. If it is not located in the operator domain, an agreement with the operator is needed where authentication and authorization data is defined. The agreement is called a Service Level Agreement (SLA). It describes what services the application is able to use through functional-based properties (e.g. types of allowed triggers, what API methods are allowed) and performance-based properties (e.g. maximum number of calls per second, maximum number of messages).

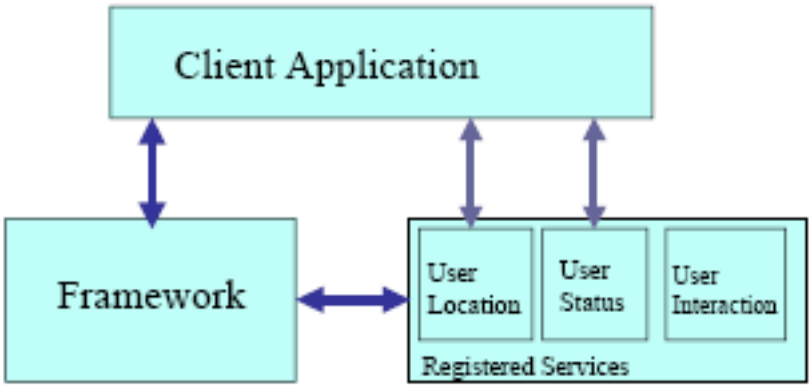

Figure 63: NRG architecture overview, taken from [28]

The first that happens when an application wants to use some of the services in NRG, is mutual authentication between the Framework and the application. When this is done, the application can request the Framework for access to services by signing the SLA. Further on, when the SLA has been established, the Framework requests the service to create a service manager and the reference to this service manager is returned to the application. The service manager is the proxy of the service that the application interacts with. The application is now ready to access the service and start using its resources through the service manager.

C.2.5 - Getting access to a service

When an application needs access to a service, it needs to follow a certain sequence of actions. This sequence is almost the same for each service the application needs access to, so the SDK Software Libraries contain a component that simplifies the needed interaction with the Framework to get access to the service. This component is called the framework Proxy. Instead of having to invoke 13 method calls, the application developer only needs to invoke one. Figure 64 illustrates this.

When requesting a service from the framework proxy, all the application needs to specify is which service it needs. This is done by specifying the name of the service. An example of the name of a service is "SP_HOSA_USER_STATUS" for the User Status service. The name of a H-OSA service can be found in the Ericsson specification of the given service, which can be found in the documentation of the NRG SDK, see [31]. The rest of the information that the framework proxy needs, e.g. authentication information, is located in a configuration file.

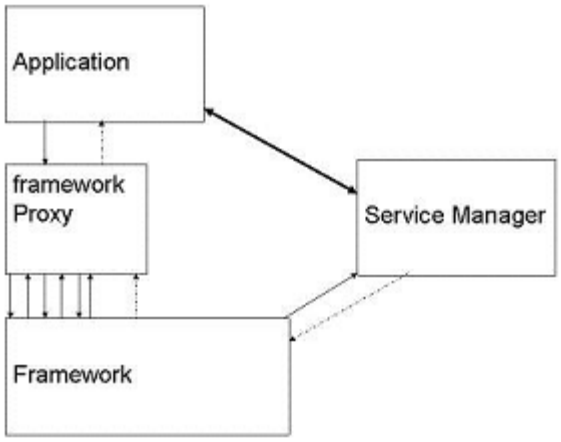

Figure 64: The framework Proxy

C.2.6 - Callback objects

In section C.2.4, it was explained that applications need some special interfaces called service managers to use a service. In addition, the service manager needs callback objects to communicate with the application, since almost all OSA/Parlay and H-OSA methods are asynchronous.

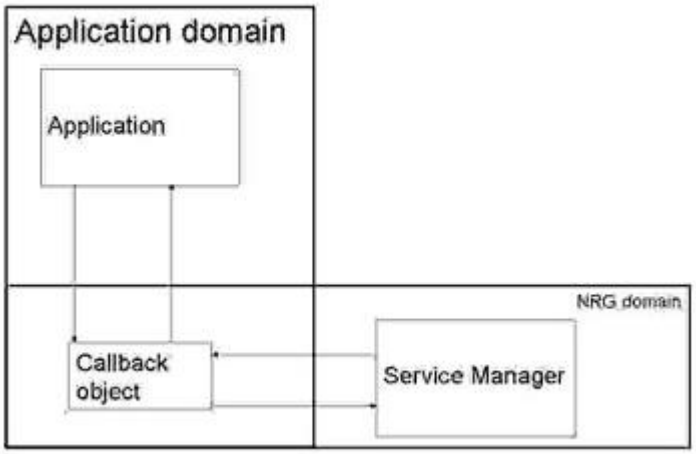

Figure 65: Communication between NRG and an application

When an application requests something from the service manager, the response is returned asynchronously. Each service manager has a special kind of callback object with predefined methods that can be invoked by the service manager when it is ready to return a response. In this way, the application will not have to await the response of the Service Manager. The callback object is implemented as a part of the application, see Figure 65.

Appendix D - Application platforms

This section will present the different platforms that were used to develop the demonstrator service. The demonstrator uses ServiceFrame as service platform running on top of ActorFrame. In addition, OWL and the Protégé-OWL API were used to, respectively, specify and populate the context ontology.

Appendix D.1 - ServiceFrame

This section will provide a presentation of ServiceFrame. An overall description will be given followed by a presentation of its architectural support and the framework model. The content of this presentation is taken from [32, 33].

D.1.1 - Overall description

ServiceFrame is a service creation and execution environment and can be seen as an application server in the service network (see Figure 66). It provides users on different terminals to communicate with each other and provides access to network resources through the OSA/Parlay API. Although ServiceFrame can be used as an application server, it does not offer all functionality that a commercial application server, e.g. Bea Weblogic, such as management and database storage. However, it can communicate with different resources in the network, e.g. Parlay's SCSs, clients and regular servers. It can be used to deploy and run services in the service network.

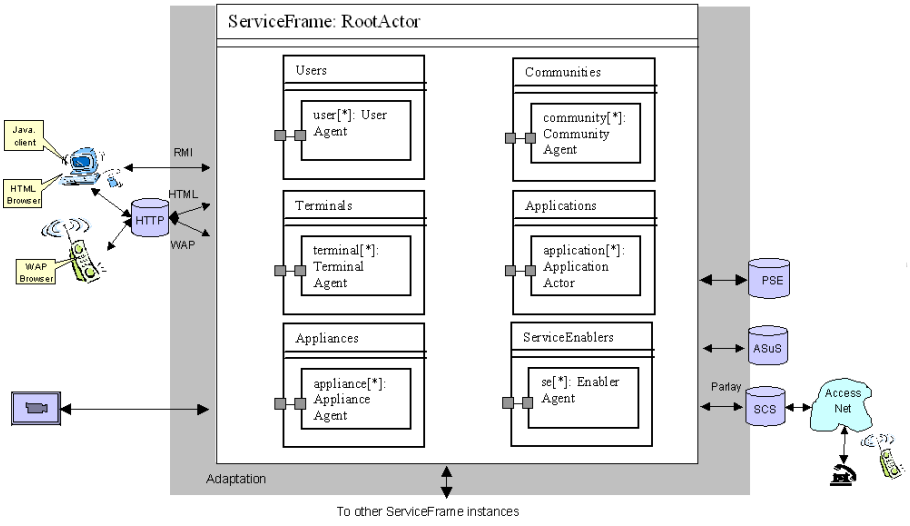

Figure 66: ServiceFrame, taken from [32]

D.1.2 - The architectural support

The most important characteristic of ServiceFrame is its architectural support for service creation, service deployment and service execution from an application's point of view. It is the applications, when deployed on ServiceFrame, that constitute the end-user services. The applications can make use of predefined ServiceFrame classes, extend their behaviour as well as define their own classes that interact with the framework classes. In this context, ServiceFrame can be seen as a framework providing class libraries, architectural design-guidelines and functionality for easy deployment of services. As Figure 67 shows, the architectural support of ServiceFrame is provided as three layers. In this view, ServiceFrame is itself an application of ActorFrame.

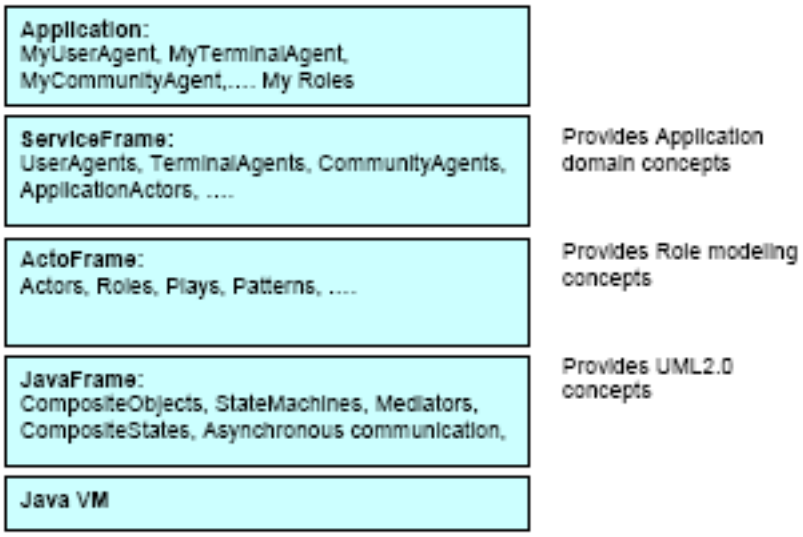

Figure 67: Ericsson's service creation architecture, taken from [32]

ServiceFrame consists of agents, that are defined in ActorFrame as actors, and actors play roles to create a service. It is layered on top of ActorFrame and JavaFrame. For further information about ActorFrame, see Appendix D.2. JavaFrame is both an environment for execution and a class-library used to implement state machines and asynchronous communication between state machines. The idea of ServiceFrame is that service developers shall be relieved from having to deal with technicalities that are not service specific. ServiceFrame has therefore defined a lot of general service-specific functionality. Classes and their respective behaviour are modelled using UML 2.0. For a presentation of UML 2.0, please refer to [34].

D.1.3 - The framework model

As already described, ServiceFrame provides architectural support through class libraries. The main actor is ServiceFrame and contains inner actors in a predefined structure. By extending ServiceFrame, an actor can extend the basic behaviour of ServiceFrame (e.g. add new parts that interact with the predefined inner parts of ServiceFrame). Figure 68 shows the structure of ServiceFrame with its inner actors. For a detailed description of these, please refer to [33].

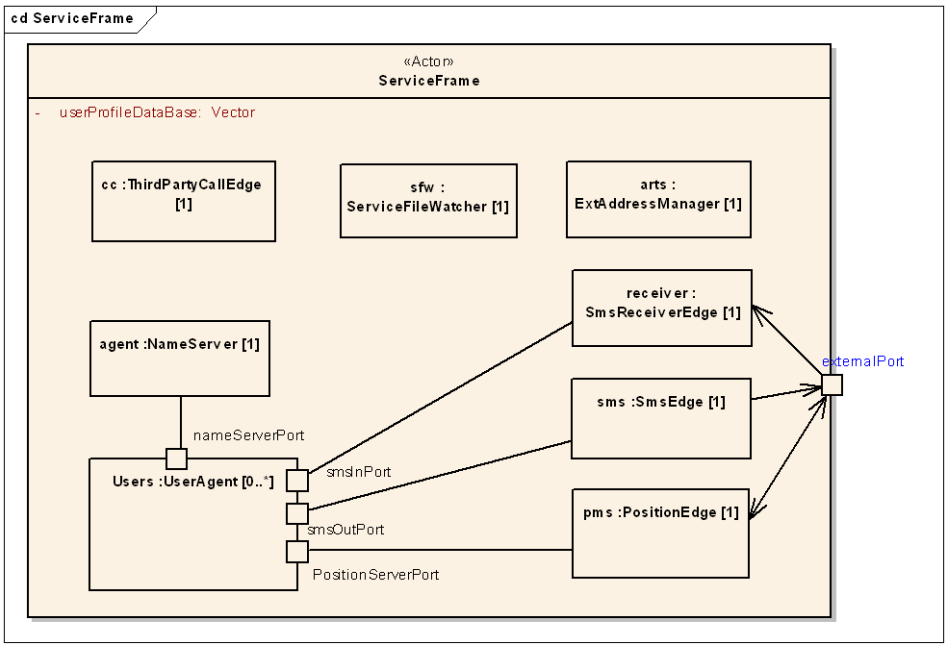

Figure 68: ServiceFrame UML model, taken from [33]

Appendix D.2 - ActorFrame

D.2.1 - The service view

ActorFrame makes use of the "Actors play service roles" concept, which means that a service is made up of collaborating actors each playing a role and offering specific functionality that form a part of the service. The role that an actor plays can be seen as a service role and the collaboration and interaction of these specify the service functionality. A service may be seen as a play made up of actors playing service-roles. Below, a presentation of the Actor concept and the ActorFrame protocol will be given. It is based on the information found in [32].

D.2.2 - The Actor

The Actor is, above all, the core concept of ActorFrame. It has a predefined behaviour (state machine) and structure, and can contain a number of inner parts (actors). These can be static and follow the life cycle of the enclosing Actor or be dynamic and follow their own life cycle in an ad-hoc manner. The inner parts are roles that the Actor may play. The predefined behaviour is realised through a generic state machine, where the Actor can send and receive predefined messages, e.g. request inner parts to play a certain role. The communication between the Actor, its environment and its inner parts takes place through ports (in Figure 69 called in and out) or actor addresses. The actor address is made up of the actor name and the actor type. The actor name has to be unique within the name scope (among actor instances) of a requested actor.

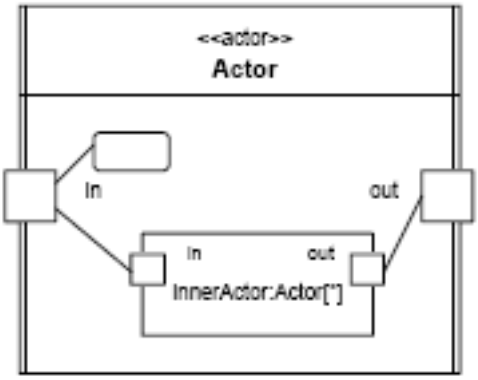

Figure 69: The Actor class, taken from [32]

Other actors can inherit and extend the base class Actor. They will then inherit the generic behaviour and structure of the Actor class, making it possible to define inner parts playing certain roles and extend the state machine and functionality of the Actor class. The Actor class provides management functionality that makes it possible to control the life cycle of actors extending it. It knows the available roles its inner parts may play and the rules for role invocation, role adding and role removal, see section D.2.3.

D.2.3 - The ActorFrame protocol

The ActorFrame protocol specifies how to request actors to play roles, interact to perform a service or a play and release them from playing these roles. The idea is that an actor can request other actors to initiate new roles to do a service, see Figure 70. In section D.2.1, it was stated that services are realised as a number of actors playing roles, each providing a part of the service functionality. The

ActorFrame protocol makes it possible to request a service actor that contains inner parts (actors). The service actor requests its inner parts to play their roles without the application's knowledge of this. This makes it possible to dynamically request roles when needed from the service actor's point of view. All an application needs to know is how to request the service actor.

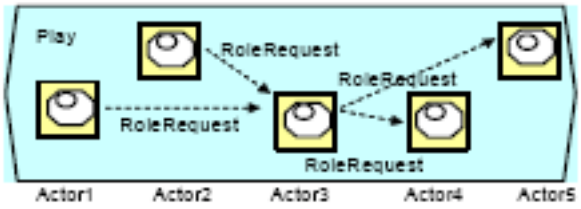

Figure 70: Multiple role requests, taken from [32]

Appendix D.3 - The Web Ontology Language

The Web Ontology Language (OWL) makes it possible to describe and define concepts with a rich set of operators. In addition, the logical model allows a reasoner to be used to check the consistency of the statements and definitions in the ontology and recognize which concepts fit under which definitions.

OWL comes in 3 sub-languages: OWL Lite, OWL-DL (Description logics) and OWL-Full. OWL-Lite is the least expressive sub-language while OWL-Full is the most expressive. OWL-DL falls in between OWL-Lite and OWL-Full. Automated reasoning is offered by OWL-DL, making it possible to compute the classification hierarchy and consistency checking. This is not possible with OWL-Light (too simple) nor OWL-Full.

Below is a presentation of the most important concepts of OWL and it is based on the contents of [52, 53].

D.3.1 - OWL individuals

Individuals represent entities in the domain of interest and can be seen as being instances of classes. OWL does not use the unique name assumption, which means that two different names can refer to the same individual. For example, "now_time", "oslo_time" and "trondheim-time" might all refer to the same individual. In OWL, it must be stated explicitly that two individuals are different or else they might be the same as each other or they might be different to each other.

D.3.2 - OWL Properties

Properties are binary (between two things) relations on individuals, e.g. two individuals are linked together with a property. There are different types of properties, and the most important ones are Object properties, Datatype properties and Functional properties. Object properties link an individual to another individual, while Datatype properties link an individual to an XML Schema Datatype value[8] or an RDF literal[9]. If an individual has a Functional property, then there can be at most one individual that is related to the individual through the property. This implies that, if the individual "A" is linked with the individual "B" via the functional property "p" and "A" is linked via "p" to "C", then the individuals "B" and "C" are the same individual.

D.3.3 - OWL Classes

OWL classes are interpreted as sets that contain individuals and can be viewed as a concrete representation of a concept. They are described using formal descriptions that state precisely the requirements for an individual to be a member. The classes may be organised into hierarchies of super- and sub-classes, where the sub-classes specialise their superclasses. So when a class A has a superclass B, all individuals of A are also individuals of B. Alternatively, the fact that an individual is a member of the class A implies that it is also an individual of the class B. These superclass-subclass (called a taxonomy) relationships can be automatically computed by a reasoner when OWL-DL is used.

An OWL class can be defined to be disjoint from other OWL classes, making it impossible for individuals of the first class to be individuals of the disjoint classes. Again, in OWL, classes are assumed to overlap each other meaning that one cannot assume that an individual is not a member of a particular class simply because it has not been asserted to be a member of that class. This is why it is necessary to make classes disjoint from each other, which ensures that an individual of a class (A) that has been asserted to be disjoint from another class (B) cannot be a member of the other class (B). In addition, classes can be asserted to be intersections, unions and complements of each other to further increase the accuracy of the descriptions.

D.3.4 - Describing and defining OWL classes

One of the most powerful features of OWL is the property restriction, which can be divided into value constraints and cardinality constraints. In OWL, properties are used to compose restrictions (that restrict individuals belonging to a class). Properties may have a range and a domain, where individuals from the domain are linked *to* individuals from the range.

[8] See `http://www.w3.org/TR/xmlschema-2/` for information on this

[9] See `http://www.w3.org/TR/rdf-concepts/#section-Literals`

The types of value constraints are owl:someValueFrom, owl:allValuesFrom and owl:hasValue. A value constraint is used to restrict an individual via a property to some other individual. For instance the class "Child" could be linked via the property "hasParent" to the class "Physician" through the value constraint owl:someValuesFrom. This restriction describes the set, or the class, of individuals that have *at least one* parent that is an individual from the class Physician. If the value constraint was owl:allValuesFrom and the range was the class "Human", then the restriction would describe the set, or the class, of individuals that have parents that *only* are individuals from the class Human. If the value constraint owl:hasValue was to be used, then the range of the property would have to be an individual. For instance, if the expression was the same except that the value constraint was owl:hasValue and the range was the individual "George Bush", the restriction would describe the set of , or the class, of individuals that *have* the individual referred to as George Bush as their parent.

The cardinality constraints are owl:maxCardinality, owl:minCardinality and owl:cardinality, and can be used to describe the class of individuals that (respectively) have *at least*, *at most* and *exactly* a specified number of relationships (properties) with other individuals or datatype values.

Classes may be described using necessary and/or sufficient conditions (restrictions). Necessary conditions can be read as "if something is a member of this class then it is *necessary* to fulfil these conditions". With necessary conditions alone, it is not possible to say "if something fulfils these conditions, then it *must* be a member of this class". With sufficient conditions, this is possible. Thus, if a class is described using at least set of necessary and sufficient conditions, it is considered to be defined. If it only is described using necessary restrictions, it is not considered to be defined (denoted as primitive in Protégé-OWL).

Appendix D.4 - The Protégé-OWL API

To develop the OWL ontology, Protégé version 3.1[10] was used as an editor. It contains the Protégé-OWL plug-in that facilitates the creation of OWL ontologies using their implementation of OWL. In addition, the RACER[11] reasoner was used to check the consistency and classify the taxonomy of the ontology.

Figure 71 shows a UML model representing the most important interfaces that are used in the implementation of the CBCC application. They are all contained in the model package of the Protégé-OWL API, and can be obtained from the JenaOWLModel (see section 5.2.3) that is used in the demonstrator.

[10] Available at `http://protege.stanford.edu`
[11] Available at `http://www.sts.tu-harburg.de/~r.f.moeller/racer`

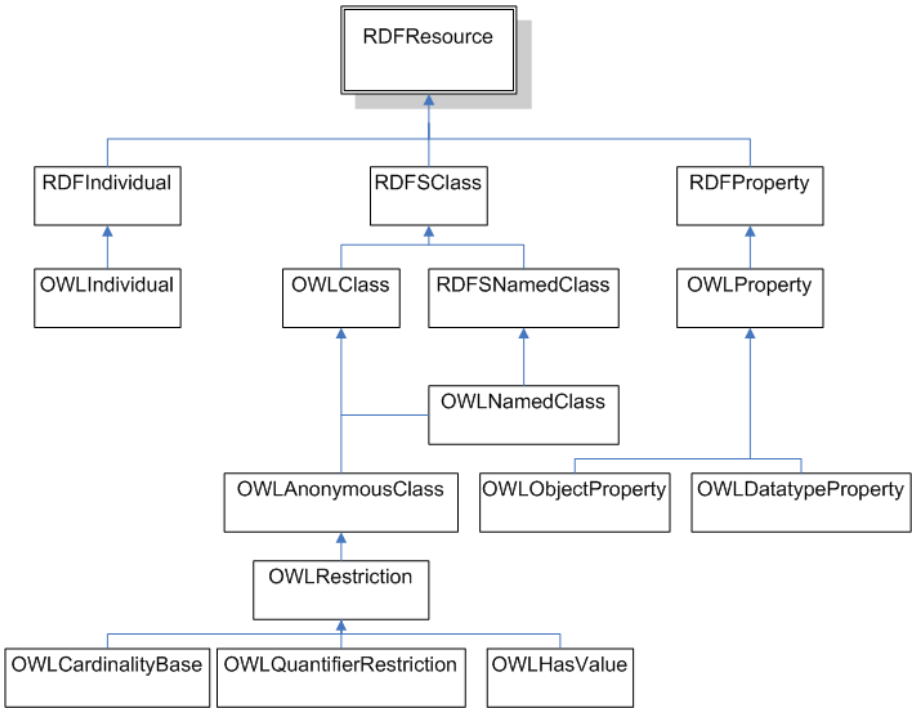

Figure 71: UML model of the most central interfaces used