NTNU
Innovation and Creativity

A Semantic Web-driven Approach to Self-Configurating Computer Systems

Jahn Arne Johnsen

Master of Science in Communication Technology
Submission date: June 2006
Supervisor: Lill Kristiansen, ITEM
Co-supervisor: Egil Østhus, TANDBERG

Norwegian University of Science and Technology
Department of Telematics

Problem Description

The objective of this master’s thesis is to asses how one can utilise ontologies encoded using the Web Ontology Language (OWL), possibly alongside with other Semantic Web technologies, to do analysis of computing systems with the goal of classifying error scenarios. The thesis should also review how one can further use these technologies to specify which actions the unit should take if any of these pre-specified situations occur (i.e. a variant of "ontology based self-configuring"). The aim is to provide a solid theoretical foundation which later works and realisations can build on. Although the technology could be used by several domains or systems, the H.323 domain will be used as an ongoing example both in scenarios and later modelling and feasibility testing. More specifically, the thesis limits its scope to the domain’s solvable software configuration problems, thereby leaving a feasibility of use study for hardware issues for future work. The H.323 domain is chosen much due to its documentation accessibility as well as an easy access to domain experts at TANDBERG R&D.

Assignment given: 20. January 2006
Supervisor: Lill Kristiansen, ITEM

To live is to battle with trolls
in the vaults of the heart and brain.
To write: that is to sit
in judgment over one's self

- *Ibsen*

Abstract

The explosive growth, in both the size and complexity, of communication and computing systems has made it increasingly difficult to manage and configure these. In today's ever-changing environments, systems frequently need to adapt and reconfigure their components to suit the restrictions imposed on them by the same environment. However, such configuration tasks may be quite time consuming and is often suited only for trained personnel. If the complexity continues to grow, it will at some point grow beyond any human's ability to manage. Consequently, there have recently been great interest in making computer systems more autonomic, thus giving them the ability to configure themselves.

The purpose of this master's thesis was to assess how one could create such self-configuring systems using Semantic Web technologies to classify and recover from pre-defined error scenarios. The Semantic Web is an effort to give meaning to information in a machine-readable way, through the use of a knowledge-representation technique known as ontologies. Ontologies are formal and explicit specifications of shared conceptualisations [103] that can be used to model and represent an entire domain, including its units and relations. This is useful in self-configuring systems as such world models are considered good starting points for reasoning as well as error solutions [3].

The thesis describes how one can utilise these ontologies encoded using the Web Ontology Language (OWL) combined with another Semantic Web technology, the Semantic Web Rule Language (SWRL), in order to classify the pre-defined error scenarios. This classification is suggested realised using three separate steps: first, the given domain is modelled and an OWL-encoded ontology is created, then necessary unit information is inserted into the ontology by the use of ontology instances. Finally, SWRL's horn-style rules are used to determine which error scenario the given situation should be classified as. It is further suggested using an event-based monitoring solution to decide when such a classification is needed. The thesis also shows how one can use the OWL service specification ontology, OWL-S, to specify and describe which actions a failed unit should take in order to recover from an error.

In addition to a thorough theoretical assessment on how Semantic Web technologies could be used in self-configuring systems, an architecture design for the realisation of such a system is proposed. The proposal includes block diagrams, state machines describing functionality, and message sequence charts. All conforms to the UML2.0 standard. The use of such formal modelling languages allows for easy translation into an executable framework to which the more specific functionality may be added (including OWL-S execution etc).

To test the feasibility of the suggested approach, two experiments were designed and conducted. These set out to test whether SWRL rules could classify error scenarios and if it was possible to execute repair procedures encoded in OWL-S. Although some of the sub-tests failed due to the immaturity of the Semantic Web field and its technologies, the approach seemed to be promising. A complete realisation of the architecture will however require that underlying technology issues are resolved.

Preface

This master's thesis was completed as part of the Master of Technology programme at the Norwegian University of Technology and Science (NTNU). It was carried out at the Department of Telematics in cooperation with TANDBERG ASA in the period January 2005 till June 2005.

I would like to thank my guidance professor Lill Kristiansen for helpful ideas, thoughts and suggestions throughout this thesis. My supervisor at TANDBERG R&D, Egil Østhus, also deserves many words of acknowledgement for proposing such a difficult and demanding, but in the end, very rewarding research project as this thesis turned out to be. I would further like to express gratitude to the both of them for encouraging words during times where things did not go as originally planned. Thank you. In the words of Woody Allen:

> *"If you're not failing every now and again, it's a sign you're not doing anything very innovative."*

Finally, I would like to thank my brother, Terje Johnsen, as well as fellow students, Trond Øivind Eriksen, Lars Erik Karlsen and Jonny Mauland, for extensive proof reading and last minute comments in the final process of this report.

Trondheim, June 2006

Jahn Arne Johnsen

Abbreviations and Thesaurus

Conceptualisation	Description of the concepts and relations that can exist in a given domain.
Consistency	Used in relations to the ontology concept, *consistency* is whether structural constraints are met and if there is contradictive information.
EXPTIME	Used for decision problems that can be solved by a deterministic sequential machine in an amount of time that is polynomial function of the problem size
Decidable / Undecidable	A decision problem is said to be is *decidable* if one can construct an algorithm which terminates after a finite amount of time. Similarly, the problem is known as *undecidable* if no algorithm can decide it.
Declarative	A computer program is called *declarative* if it describes what is, rather than how to create it. E.g. a collection of rules, axioms etc. which can be used to solve a reasoning problem.
Model	An abstract representation of a domain.
Monotonic	In a *monotonic* program, adding new knowledge will not cause the retraction of previous conclusions.
NEXPTIME	Decision problems whose solution can be found in polynomial time on a non-deterministic machine.
Ontology	A formal and explicit specification of a shared conceptualisation.
OWL	Web Ontology Language
URI	Unified Resource Identification
URL	Unified Resource Locator
RDF	Resource Description Framework
SWRL	Semantic Web Rule Language
Taxonomy	A class hierarchy
XML	Extensible Markup Language

Table of contents

Table of figures

Index of tables

1 Introduction

This chapter will present and elaborate on the motivation for this thesis, as well as present scenarios, a research statement and finally a summary of known related work.

1.1 Motivation

Throughout history, advances in technology have brought on many revolutionary changes. During the last few decades, the world has seen the advent of the personal computer, the breakthrough of the Internet and the introduction of mobile telephony. From the evolution of single machines into today's huge networks of personal and mainframe computers, an unmistakable pattern emerges: computer and information technology have grown at almost exponential rates and have had an incredible progress in almost every aspect. The architectures of computer and telecommunication systems have become increasingly sophisticated, and are now governed by literally millions of code lines which have been put together by thousands of programmers. By interconnecting these systems through networks, we are adding yet another layer of complexity, but vast possibilities are provided to the users.

As the complexity of modern communication systems increases, so does the need for proper configuration and management. In an ever-changing environment, these systems frequently need to adapt by performing actions and (re)configuration of their services and workflow. Such system configuration and management can be a time consuming and error-prone process which is already today is often suited only for skilled IT personnel. In the future, not only will there be a shortage for people with the right knowledge and skills to manage computer and telecommunications systems, but the complexity it self is growing beyond any human's ability to manage it. One can only imagine the impact on users having to go through three thousand manual pages in order to install and use the newly acquired conferencing system. It is therefore becoming increasingly desirable to design computer systems that are able to configure themselves. According to Horn [53]:

> *"It's time to design and build computing systems capable of running themselves, adjusting to varying circumstances, and preparing their resources to handle most efficiently the workloads we put upon them." [53]*

By attempting to accomplish this self-management by a computing model analogous to the human autonomic nervous system, this approach has been named Autonomic Computing [99]. Autonomic Computing helps address the growing complexity by using technology to manage technology [61]. It accomplishes its functions by taking an appropriate action based on one or more situations which they sense in their environment. For an autonomous system to behave appropriately in an uncertain environment, it has been suggested that:

> *"The system must have an internal representation (world model) of what it feels and experiences as it perceives entities, events, and situations in the world. It must have an internal model that captures the richness of what it knows and learns, and a mechanism for computing values and priorities that enables it to decide what it wishes to do."[3]*

This is in accordance with Eracar [31], which states that a key factor for achieving self-awareness, an essential element of any autonomic system, is self-modelling. While there is a large body of work in various knowledge representation and ontology areas, relatively little has been applied to the area of world modelling in autonomous systems. It would thus be desirable to research ways which such formal world models can be used in autonomic systems.

1.2 Research Statement

The objective of this master's thesis is to asses how one can utilise ontologies encoded using the Web Ontology Language (OWL), possibly alongside with other Semantic Web technologies, to do analysis of computing systems with the goal of classifying error scenarios. The thesis should also review how one can further use these technologies to specify which actions the unit should take if any of these pre-specified situations occur (i.e. a variant of "ontology based self-configuring"). The aim is to provide a solid theoretical foundation which later works and realisations can build on.

Although the technology could be used by several domains or systems, the H.323 domain will be used as an ongoing example both in scenarios and later modelling and feasibility testing. More specifically, the thesis limits its scope to the domain's solvable software configuration problems, thereby leaving a feasibility of use study for hardware issues for future work. The H.323 domain is chosen much due to its documentation accessibility as well as an easy access to domain experts at TANDBERG R&D.

1.3 Scenarios

In [82], Muller states that:

> *"(...) we think of a scenario as a story that one person (or group of people) can tell to another, and that describes human work, human collaboration or human activities (with or without computers)". [82]*

This is similar to Regnell et al. which state in [91] that the purpose of scenarios is to give a partial description of system usage as seen by its users [91]. Thus, the scenarios presented in this section will be used as a mean to justify and explain the technology for the reader, as well as to obtain goals and a purpose for a desired system. The two different scenarios are in essence based on one single story presented with two separate variations. Scenario 1 present a plausible current situation of a given user, while the scenario 2 shows a different outcome of the situation based on the desired technology which will be researched throughout this thesis.

1.3.1 Scenario 1 – Today

Alice, a project manager for a large telecommunication firm, is about to attend a meeting using her brand new video conferencing system. Sitting down at her desk in her home office, she starts dialling the number which she received by email from the secretary at her client's firm. After the number has been entered into her video conferencing unit, nothing happens. The unit merely states "Call setup failed, please try again". Alice tries again, but with the same result. Since this is Alice's first attempt to use the unit on her own, she decides to try to get in touch with the IT personnel in the Help department at her firm. After a while, she finally gets hold of Bob, who is head of the company's video conferencing project. As Bob takes her through the configuration steps, he discovers that her unit is not registered properly at any video gatekeepers. He looks up the needed gatekeeper id and tells Alice to type it in using the configuration manager. After some time, Alice manages to register with the gatekeeper and is ready to attend the video conference with her client. By now, Alice is quite frustrated with the equipment, half an hour late, and feels a bit amateurish as she calls the client.

1.3.2 Scenario 1 – Near Future

Alice, a project manager for a large telecommunication firm, is about to attend a meeting using her brand new video conferencing system. Sitting down at her desk in her home office, she starts dialling the number which she received by email from the secretary at her client's firm. Once the number has been entered into her video conferencing unit, nothing happens, but the unit states "Call setup failed, please wait while the unit tries to resolve the problem". After a few seconds, the screen shifts to state "Problem resolved, redialling". Although Alice did not quite understand what just happened, she appreciated the fact that the unit seemed to fix itself and goes on with her scheduled meeting.

1.4 Related Work

Although there are, to the author's knowledge, no published research on a Semantic Web driven approach to self-configuring system, some related previous work can be found.

The problem of configuring complex computer systems has existed for years and a large number of management standards such as the Simple Network Management Protocol (SNMP), the Desktop Management Interface (DMI) and the Telecommunications Management Network (TMN) have been proposed [76]. Common for these are the definition of protocols for exchanging information between the managing and the managed computing devices as well as definition of basic manipulation functionality such as information access, addition, editing and deletion. Unfortunately, the various initiatives all define their own resource models and protocols without any coordination between them, resulting in severe interoperability issues, as well as incompatibility between concepts and terminologies [108].

In order to overcome these issues, the Web Based Enterprise Management (WBEM) initiative, developed by the Distributed Management Task Force (DMTF), was introduced in 1996. The WBEM is a standard independent of the underlying platforms and resources. It defines management architecture, management protocol, management schema, and object manager. It uses a schema called the Common Information Model (CIM) to represent its entities and managing information. The CIM schema provides definitions for servers, desktops, peripherals etc. Its object oriented approach also makes it easier to track relationships between managed objects [108]. However, even with this new approach, interoperability between different management schemes is still cumbersome and dependent on separate translation approaches. In addition, there are situations where policies are not transferable between the different management domains [111].

In [111], Vergara et al. address this issue and state that management policies should be generic and independent of the models used to define the resources. In order to accomplish this, they suggested using a common ontology to define the domains, and on this ontology specify policies that should be enforced on the system. The policies are suggested to be represented through the use of a predicate logic language such as KIF or OCL [111]. These policies would then be applied to the different sub-management systems through specialised gateways. Its architecture is shown in Figure 1-1.

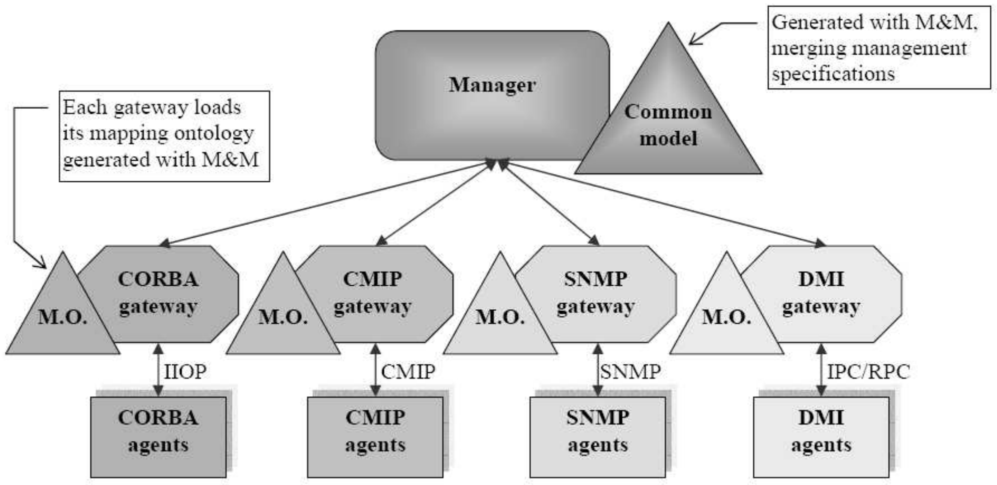


Figure 1-1 – Architecture of an ontology-based management system [111]

The ontology and policy technology components of this initial architecture have later been suggested replaced by Guerrero et al. in [49]. Here they propose the use of the Web Ontology Language (OWL) to represent the ontology and to further use the Semantic Web Rule Language (SWRL) to express the different management policies, making it a Semantic Web driven approach [49]. However, any complete architecture is suggested, and neither are behavioural models or any general logic for the system. Also, there are no empirical attempts to show the feasibility or soundness of the proposal.

1.5 Thesis Outline

This thesis has been divided into 11 chapters and one appendix. In the following the different chapters will be briefly described.

The H.323 standard provides a solid foundation for multimedia conferencing over packet switched networks. Since the H.323 domain is used as an on-going example in this thesis, the second chapter provides a brief introduction to the technology. This includes its architecture as well as main components.

The third chapter presents the concept of autonomic computing. Autonomic computing has been inspired by the human autonomic nervous system and is aimed at designing computer systems that are self-managing [99]. The chapter further explain and elaborate on the important characteristics that an autonomic system need to possess as well as present the control architecture needed for such systems.

Chapter four is used to discuss one of the central concepts of this thesis; the ontology. An ontology is a "formal and explicit specification of a shared conceptualisation" [103]. It begins with discussing reasons for use, before the general structure and properties are presented.

In the fifth chapter, the Semantic Web initiative and its technologies are presented in detail. The chapter starts out by justifying the need for a new Semantic Web, before one of the enabling technologies for its knowledge representation formats, *description logics*, is presented. In the remainder of the chapter, the Web Ontology Language (OWL), the Semantic Web Rule Language (SWRL) and the OWL service description ontology (OWL-S) are described.

These first five chapters are meant to provide the reader with an introduction to the necessary background knowledge so that the reasoning and thoughts in the remainder of the thesis are easily understood.

The sixth chapter present several methodologies which have been used to drive the research through the different phases of the thesis. Specifically, methods for literature reviews, system engineering and ontology engineering are described.

In chapter seven, the proposed architecture for a Semantic Web driven self-configuring system is presented alongside a thorough discussion of each of the necessary parts for realising such a system. This includes choosing how units should perform self-modelling as well as how they can classify error scenarios and recover from these. The proposed architecture is at the end of the chapter presented through the use of block diagrams, state machines explaining its behaviour and sequence diagrams which ease the general understanding and flow of the system.

Chapter eight contains experiments designed to test the feasibility of the approaches proposed in chapter seven. As a prerequisite to these experiments, the H.323 domain had to be modelled and an OWL ontology of the domain is thus designed and presented. In order to specify which actions a unit needs to take in order to recover from a given error scenario, the services of a H.323 Terminal was also modelled using the OWL-S ontology.

In the ninth chapter of this thesis, an evaluation of the proposed architecture is given. This includes a discussion of its scalability, the architectural implications of using a rule-based way to classify the error scenarios as well as underlying technology weaknesses and limitations.

Finally the tenth, and last chapter, contains the conclusions of this thesis, which includes a presentation of the main contributions to the research area and suggestions for future work.

2 The H.323 Standard

H.323 is an umbrella standard which references many other ITU-T recommendations. It basically aggregates a set of standards for multimedia conferencing over packet switched networks such as the Internet [106]. As it is used for the scenarios and case throughout the thesis, this chapter will give a brief overview over the standard, its architecture and main components.

2.1 Components

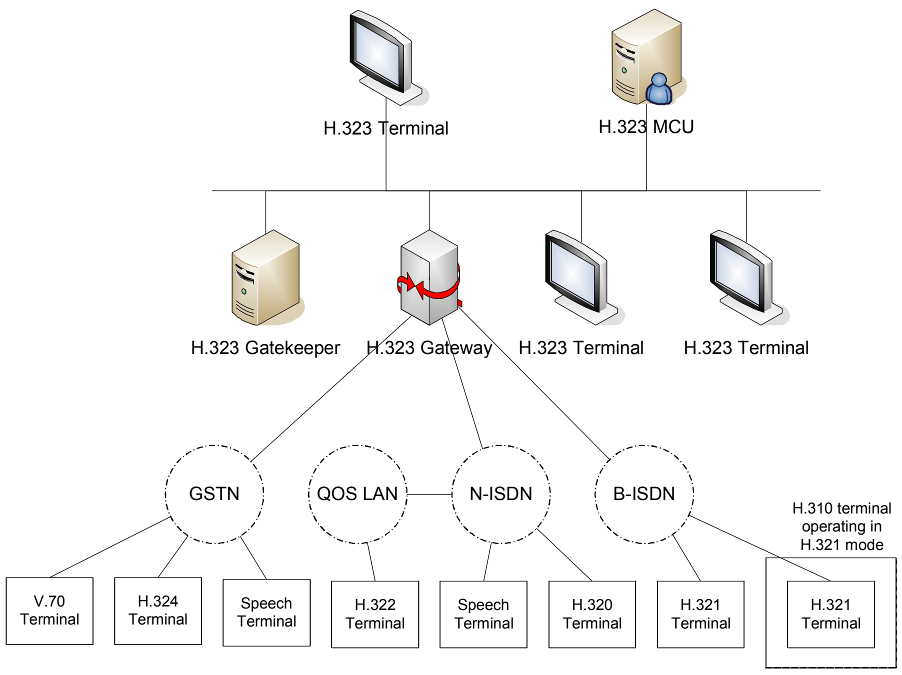


Figure 2-1 – Components in the H.323 Architecture

Shown in the top of Figure 2-1 are the major network components of the H.323 architecture: the mandatory terminal and the optional multipoint control unit (MCU), the gatekeeper and the gateway [106], [107]. Each of these will be further explained in the following.

2.1.1 Terminal

H.323 terminals are required support real-time two-way communication with other H.323 components through the use of some minimum signalling protocols (H.245, Q.931, and RAS) and audio codecs (minimum G.711). Optionally they can support video (at least QCIF H.261) and data communications (typically T.120) [107]. If the terminal does not support these minimum standards, the endpoint is not considered an H.323 standards-compliant endpoint [106]. The terminal is typically identified by H.323 IDs, which are arbitrary, case-sensitive text strings, or E.164 aliases, which are basically telephone numbers that may be local or global [106].

2.1.2 Gatekeeper

Although the gatekeeper is an optional element in an H.323 network, it is still a rather important component. It is basically responsible for managing all the other entities of an H.323 network (also known as a H.323 zone) and may provide many services such as call control [107]. In Table 2-1, the required and optional functions provided by the H.323 gatekeeper, is shown.

Required functions	
Address Translation	Translation of E.164 aliases or H.323 IDs to IP addresses using a table typically updated with registration messages.
Admission Control	Authorisation of LAN access based on call authorisation, bandwidth or other criteria. Support of Admission Request, Confirm and Reject (ARQ, ACF, ARJ)
Bandwidth Control	Support for Bandwidth Request, Confirm and Reject (BRQ, BCF, BRJ). This may be based on bandwidth management.
Zone Management	The gatekeeper will provide the functions mentioned to terminals, MCUs and gateways which has registered within its control zone.
Optional functions	
Call-control signalling	In a point-to-point conference, the may process call control signals or, alternatively, send them directly to each other
Call authorisation	The gatekeeper may reject a call from a terminal based on restricted access to particular terminals or gateways or restricted access during a certain period of time.
Bandwidth management	Places a limit on the amount of bandwidth the terminal may use on the network.
Call management	The gatekeeper may maintain a list of ongoing calls in order to indicate that a called terminal is busy or to provide information for the Bandwidth management function.

Table 2-1 – An H.323 gatekeeper's required and optional services [25]

2.1.3 Gateway

The gateway provides a support for interoperability with other communication networks as shown in the bottom part of Figure 2-1. More specifically H.323 gateways provide the following functionalities [64]:

- Translation between transmission formats, e.g. H.225.0 and H.221
- Translation between communication procedures, e.g. H.245 and H.242
- Translation between audio and video codecs
- Call setup and clearing on both the LAN side and the switched-circuit network side

Gateways are not required unless connections to other networks are needed.

2.1.4 Multiple Control Units

The Multi Control Unit (MCU) supports multipoint conferences, i.e., conferences between three or more endpoints. An MCU is required have a Multipoint Controller (MC) for signalling, and zero or more Multipoint Processors (MP) for mixing, switching and processing media streams and / or data bits. The MCU can be a single dedicated entity, or may be integrated in other components [64].

2.2 Security

Version 2 of the H.323 recommendation contains a number of improvements for IP telephony, including a completely new security recommendation, H.235, which was developed to provide a full security framework for H.323 and other multimedia systems. This recommendation provides services for authentication (which can be used for authorisation), privacy and integrity [63]. The services are accomplished through the use of so called security profiles which include [63]:

1) a simple, password-based profile
2) a profile using digital certificates and dependent on a fully-deployed public-key infrastructure
3) a combined use of both 1) and 2)

This can then be used in e.g. the encryption of a conference media stream. Use of these security profiles is optional and is activated by each unit. Endpoints may be required to use specific security features either by their gatekeeper, or by the recipient of any conference call.

3 Autonomic computing

The advances computing and telecommunication technologies have made in the last decades, have resulted in an explosive growth in computing systems and applications which impact all aspects of our daily life. However following the growth and vast possibilities of today, are problems. The applications are often highly advanced, heterogeneous and dynamic, and we can clearly see their configuration and management systems becoming more and more complex. This has led key firms and researchers to consider alternative approaches based on strategies used by biological systems. Autonomic Computing is one of these emerging new strategies. It was first proposed by Paul Horn, IBM's Senior Vice President of Research, in a keynote presentation at the AGENDA 2001 Conference and later published in [50].

This chapter will present the Autonomic Computing concept, and further elaborate on the important characteristics of an Autonomic System which this thesis will focus on.

3.1 Definition and Characteristics

Autonomic Computing has been inspired by the human autonomic nervous system and is aimed at designing and construction systems that are self-managing [99]. According to Horn, there are eight general elements, or characteristics, which define a true Autonomic Computing system, and thus define such a system [50]. From these characteristics, several terms to describe them have been suggested [74]:

- Self-aware
    - In order to be autonomic a computing system needs to "know itself". It must also consist of components having a form of system identity.
- Self-configuring
    - An autonomic computing system must be able to adapt dynamically to changes in the environment and configure and re-configure itself under varying and unpredictable conditions.
- Self-optimising
    - Being an autonomic computing system means never settling for status quo, but always looking for ways to optimise how it works.
- Self-healing
    - An autonomic system must also be able to perform some sort of healing, i.e. it must be able to recover from routine and extraordinary events which might cause some of its parts to malfunction. This must be done without any loss of data or noticeable delays in processing.

- Self-protecting
    - Since a virtual world is no less dangerous than the physical one, an autonomic system must therefore be an expert in self-protection.
- Environment-aware
    - An autonomic system is aware of the surroundings and context of its activity and acts accordingly.
- Openness
    - The autonomic system does not exist in a hermetic environment. This means to be able to function in a heterogeneous world based on open standards.
- Anticipatory
    - An autonomic system hides its complexity to the user while still anticipating the optimal resources which are needed in the future.

Any system which seeks to be a fully autonomous system must, according to Horn in [50] possess at least all these characteristics.

3.2 Architecture

Some of the characteristics mentioned in section 3.1 form a necessary basis which other autonomic functions rely on. For example, according to Albus in [3] any autonomic system needs an internal model of its self and its environment, and thus needs to be self and environment aware [3]. The other functions will, according to Ganek and Corbi ([39]), be accomplished by taking an appropriate action based on one or more situations that they sense in their environment. This is realised by combining the basis characteristics with the use of control loops that collects details from the system and acts accordingly [39]. This control loop is shown in Figure 3-1.

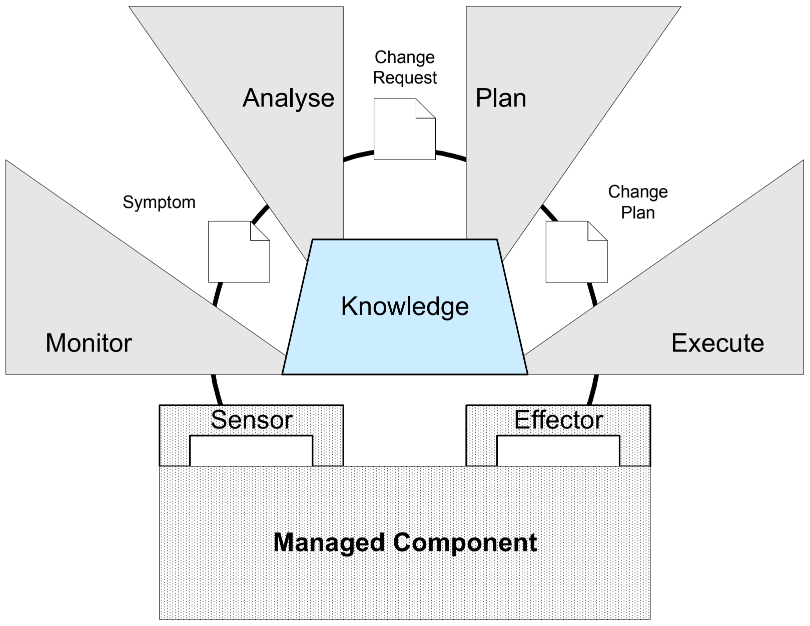


Figure 3-1 – The autonomic control loop [61]

As we see, the control loop revolve around knowledge, which, depending on which capability the control loop realises, may be information about the system itself, its environment or similar. Besides from the central knowledge component, the control loop consists of four separate parts that share knowledge [61]:

- The monitor function which provides mechanisms for collecting, aggregating, filtering and reporting details collected from a managed system.
- The analyse function provides the mechanisms that correlate and model complex situations.
- The plan function constructs actions needed to achieve goals and objectives.
- The execute function provides the system with mechanisms which control the execution of a given plan.

The four parts work together to provide the control loop functionality and communicate with one another and exchange appropriate knowledge and data. The components will be further described in the following sub-chapters. It is worth noticing that the autonomic computing architecture does not however prescribe the specific implementation choices for the autonomic manager, but rather specify how one should organise the internal structure [39].

3.2.1 Knowledge

Data used by the control loop's for functions are stored as shared knowledge. This includes data such as topology information, detailed knowledge of its components, the current status of different variables, capacities, interconnections with other systems and available resources.

The use of such knowledge is in accordance with Eracar which states in [31] that a key factor for achieving self-awareness, an essential element of any autonomic system, is self-modelling.

The knowledge used in the control loop can, according to a white paper published by IBM ([61]), be obtained in one of three ways [61]:

1. The knowledge is sent to the control loop through a pre-determined interface. Policies and analysing rules are examples of such knowledge.

2. The knowledge might be retrieved from an external information service. This might be world model details about units in its environment, or updates on other services.

3. The control loop might create the knowledge itself. This could be internal system information which is collected through sensors or logging of notifications. Another feasible scenario is that the execute part of a control loop might update the knowledge to indicate which actions that have been carried out.

The autonomic computing blueprint also identifies several types of knowledge which is needed depending on the control loop function. This includes topology knowledge, policy knowledge and problem determination knowledge [61].

3.2.2 Monitor

As previously mentioned the monitor function collects the details from the system and organise them. These details may include topology information, configuration properties and so on, and is used to keep the central knowledge base up-to-date. Whenever an error symptom is detected, this is passed to the analyse function [61] [39].

3.2.3 Analyse

The analyse function observe and analyse situations to determine whether changes need to be made. As an example, the requirement to enact a change in configuration might occur when the analyse function determine that some policy is not being met [39]. Initially, such responses will probably follow rules generated by human experts, but will probably over time will be supplemented by self-learning processes inherent in an autonomic system [74].

If there are any changes which are deemed necessary or desirable, the analyse component will send a change request to the plan function, including the prescribed modifications [61] [39].

3.2.4 Plan

When a change request is sent, it is the plan component which creates or selects a procedure which will enact the desired alteration. This plan function can take many forms, ranging from a single command to a complete work flow [39].

3.2.5 Execute

Finally, the execute function provides the mechanisms needed to schedule and perform the necessary changes to the system. It is responsible for carrying out the procedure generated by the plan component [61].

4 Ontologies

The word “ontology” has through the years been used in a number of contexts and interpreted in just as many ways. Originally, the term ontology came from philosophy and the study of metaphysics. Metaphysics tries to give a general and fundamental account of the way the world exists and how. In real-life, it is impossible to represent the real world with all its possible details. Therefore, to be able to represent a phenomena or a small part of the world, it is necessary to focus on a limited number of concepts which provide a sufficient and relevant abstraction [20]. This brings us closer to the modern concept of ontology normally used in computer science; a formal point of view where certain features and attributes of objects become more relevant than others. Guariano names in [48] several fields which have embraced the ontology concept including knowledge engineering, knowledge representation, qualitative modelling, language engineering, database design, information retrieval and extraction, and knowledge management and organisation. This concept will be presented closer in the following.

4.1 Definition of Ontologies

In some cases, the term “ontology” is just used as a fancy name for denoting the results of modelling activities carried out by means of standard methodologies. There are, however, numerous formal descriptions and definitions of what an ontology is. In [35], Fensel described it as:

“(...) a shared and common understanding of a domain that can be communicated between people and heterogeneous and distributed systems.” [35]

Although it captures most of the essence of the common concept of an ontology, Fensel’s definition captures also a lot of other concepts which are not an ontology like e.g. general UML models, purely textual descriptions of domains etc. Humns and Singh shared most of Fensel’s definition, but restricted the ontology concept to only deal with computerised versions when they in [59] defined an ontology as:

“(...) [an ontology is] a computer model of some portion of the world.” [59]

This is however also quite ambiguous as it can also be used to categorise e.g. a computerised 3D model of a landscape. Gruber’s definition as presented in [45], is more specific:

“(...) an ontology is an explicit specification of a conceptualisation.” [45]

This is the most referenced definition in the literature, but has later been considered as too broad. In 1998, Studer et al. slightly modified Gruber’s definition in [103] and stated:

“(...) an ontology is a formal and explicit specification of a shared conceptualisation.” [103]

This accentuates the need for a formalisation of the knowledge, as well as stress the fact that the ontology should be based on a shared, and thus consensual, conceptualisation of the domain. This means that a central aspect of the development of an ontology is the development of the conceptualisation. A conceptualisation represents a way to conceive the world and decide what to model in a knowledge representation. Studer et al. [103] defines a conceptualisation as:

> *"Conceptualization refers to an abstract model of some phenomenon in the world by having identifed the relevant concepts of that phenomenon." [103]*

In addition to this, both Gruber and Studer et al. based their work on a definition by Genesereth and Nilsson which stated:

> *"(...) [a conceptualisation is] a structure <D,**R**>, where D is a domain and **R** is a set or relevant relations on D" [42]*

According to this, an ontology is thus a form of knowledge representation and way to express an information model in an explicit and machine readable form. It contains the consensual knowledge of a domain and will contain all types of basic concepts with attributes as well as relationships and related constraints between different concepts.

When the terms ontology and conceptualisation is used throughout the rest of this thesis, they are referring to Studer's definitions.

4.2 Reasons for Use

Yang and Calmet state in [117] that a key reason for employing ontologies in intelligent systems is that they:

> *"(...) enable the representation of background knowledge about a domain in a machine understandable form." [117]*

This fits very well with the research statement of this thesis. In addition to this, Noy and McGuinness points in [84] out several other possible reasons for developing ontologies [84]:

- Share common domain information among humans and machine agents
    - Ontologies can be used to ensure that all participants of a domain share a common understanding of important terms and concepts. This can be utilised in e.g. a computer agent used for price searches after books if all the book shops share and publish the same ontology for all their terms.
- Enable reuse of domain knowledge
    - Once an ontology has been created, it can be reused in other ontologies and domains. This means that if one needed to build a large ontology, this could be put together by integrating several existing ontologies, each describing a small part of the complete ontology.
- Analyse domain knowledge and make it more explicit

  - Explicitly stating domain knowledge makes it easier for new users to learn what terms in a given domain mean. It also facilitate for formal analysis of a domain.

- Separate domain knowledge from the operational knowledge

  - By separating the domain knowledge the implementation e.g. by describing a task which put together components into a radio according to a required (ontological) specification and implementing an algorithm which does this assembly independently of the product and its components.

All these, combined or by them self, have led to rapid adoption from numerous research fields [48].

4.3 General Structure of an Ontology

An ontology has three basic elements [84]; description of concepts (classes) in a domain, properties (slots, also called roles) of each concept describing various features and attributes of the given concept and restrictions (facets, also called role restrictions) on these slots. These, combined with a set of individual instances of classes constitute what we call a knowledge base. There is often in reality a fine line between where the ontology end and where the knowledge base begins.

4.3.1 Concepts / Classes

Classes describe the concepts in a domain and are often the main focus point of most of today's ontologies. As an example, a class of animals represent all animals, while a class of plants represent all plants. Classes and concepts will usually constitute a taxonomic hierarchy where a class can also have subclasses which are more specific than a superclass [84]. This means that the example class of animals can be divided into e.g. fish, birds, mammals and reptiles. These subclasses can later be divided into further sub-subclasses, e.g. by dividing the mammal class into carnivore and herbivore and a necrophagous class to the carnivore class, see Figure 4-1. It worth noticing that ontology hierarchies are often depicted as "is-a" hierarchies, where the sub-concepts "is-a" super-concept.

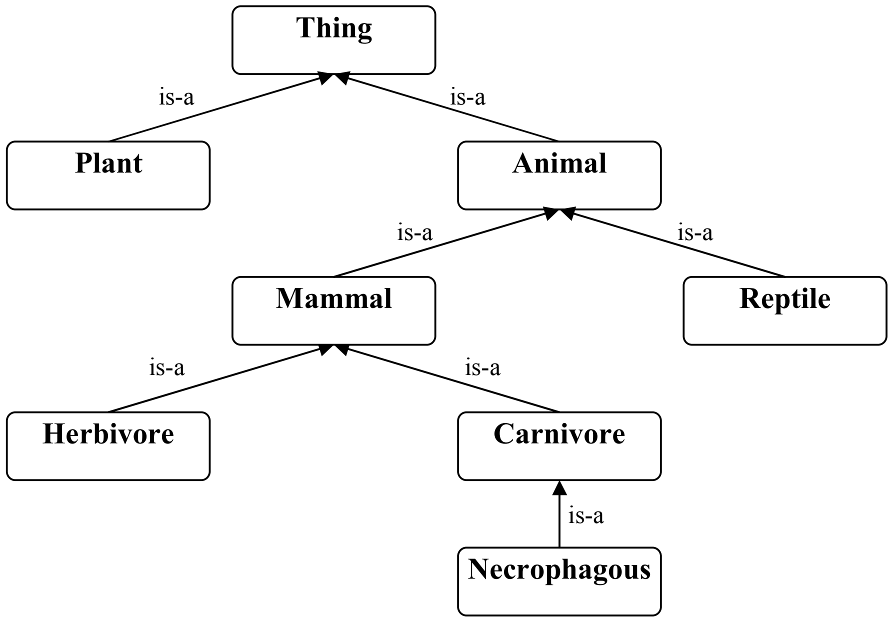


Figure 4-1 – An example ontology of animals

4.3.2 Slots / Properties

Each concept has different properties which describe various features and attributes. Continuing on the animal example from above; a mammal can have four legs and has a preference for what to eat (e.g. a plant or another animal). The latter being an object property, also called a relation, and the former a datatype property. An object property, which is also called a relation, relates individuals to other individuals. A datatype property relates individuals to datatype values such as booleans, integers, floats and strings.

In most ontology representation languages, a property is a binary relation. This means that it can only be used to link together two individuals or one individual and a value. There are however cases where the natural, and certainly most convenient, way to represent certain aspects is to link an individual to several individuals or values. These are called n-ary relations. One common solution to this problem, which lately also has been endorsed by the WC3 [85], is the introduction of a new class for the relation. This class will contain n new properties to represent the n-ary relation. This is illustrated in Figure 4-2.

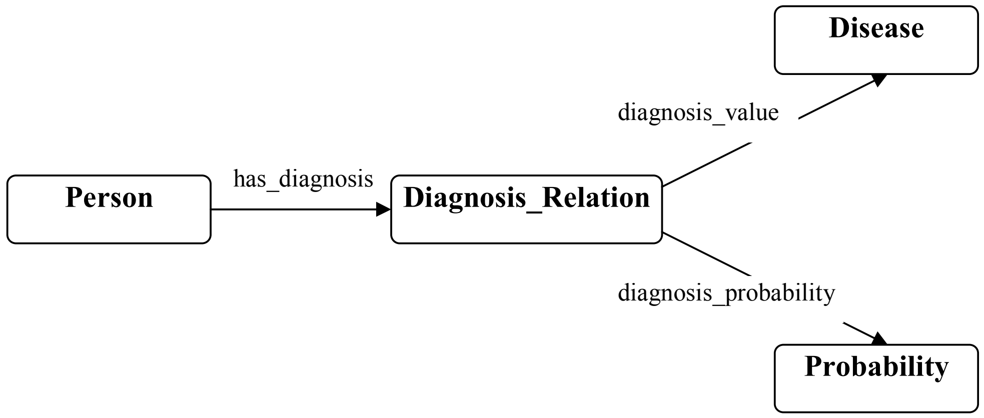


Figure 4-2 - Example of a n-ary relation [85]

This approach has been extensively used throughout the ontology modelling in chapter 8.1.

4.3.3 Restrictions / Facets

A property might have different restrictions describing the value type, allowed values, the number of values (cardinality) or other features of the values which the slot can take. As an example, the “name” property of a person is a string. This means that the only valid values of the property are a type of string. Common restrictions are [84]:

- Cardinality, i.e. how many values a property may have. Some systems distinguish only between *single* and *multiple* cardinalities, while others allow for a precise definition. It is also common for some to allow for a *minimum* and *maximum* cardinality restriction.

- The valid value types of a property, i.e. what kinds of values can fill a property (e.g. a Boolean).

- The domain and range of a property.

What restrictions that can be legally stated will depend on the chosen ontology representation language.

4.3.4 Individuals / Instances

Individuals are the actual things represented by a concept, such as Bob, being a instance of the concept of Man.

5 Semantic Web Technologies

The immense growth of the current World Wide Web which now contains billions of documents has transformed the concept of information. In response to its growing complexity, later initiatives by the World Wide Web Consortium point toward a new world where all the information will be machine processable and understandable; the world of the Semantic Web. Such technologies can, however, have several other uses which are interesting in self-configuring systems. As they may help machines understand arbitrary information, maybe it can also help them publish and process information about themselves and other units in the domain. In this chapter, a thorough introduction to the different technologies and parts of the Semantic Web is given.

5.1 The Semantic Web

"(...) the Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in co-operation." [12]

- Tim Berners-Lee et al.

In 2001 the inventor of the World Wide Web, Tim Berners-Lee, and some of his colleagues presented in [12] their vision of the future for the web. The vision was for a "web of meaning" which was designed to enable computers to help us seek out information and even derive information from the available sources that did not exist before. In this vision computers as well as people should be able to:

"(...) read, find, understand and use data over WWW to accomplish useful goals" [88]

To accomplish this, the information on the WWW needs to be understandable for computers. The concept "Semantic Web" was therefore not a vision where the computers tries to learn natural languages, but rather an extension of the current WWW where the available information is given a well-defined meaning or semantics. Thus, one of the key differences between the Semantic Web and the current Web lies how the information is represented:

"(...) the Semantic Web is supposed to make data located anywhere on the Web accessible and understandable, both to people and to machines." [88]

In the present human-oriented Web, machines are only expected to reason on a syntactical level. Their main focus is largely on how text and images should be rendered for human viewing. The Semantic Web however, is intended to use a representation of the information which will allow machines to process and reason about the information at the semantic level. Berners-Lee et al presented in vision ([12]) an advanced example of such sort, where a personal assistant automatically schedules tasks for his owner based on information which itself retrieves from the Semantic Web without requiring much human interaction.

As presented in [12] by Berners-Lee et al., the main components of the Semantic Web can be described as:

- Well-structured *annotations* (expressed in e.g. XML) on web pages which extend the traditional WWW and enable agents to capture some key concepts such as the author and his / her contact information. Adding *such meta-data* will allow us to e.g. search for documents created by different people.

- These key concepts can be linked to some further information contained within *ontologies*. This makes it possible for any agent to derive the *"meaning"* of the information and from this possibly deduce further information through the defined inference rules and restrictions.

- The concepts stored in these ontologies are *uniquely identified* by the use of URIs as identifiers. The use of URIs ensures also that the concepts are not only words in a document but are tied to a unique definition that everyone can find on the Web. This provides a way to create bridges between ontologies by *linking concepts* which exist in both ontologies with each other.

- The *Software agents* which access this information are computer programs or scripts which act on the behalf of a human (or organisation) and are therefore often considered as *"personal assistants"*. In contrast to other computer programs, software agents inhabit properties like the ability to adapt, autonomy of actions and learning aptitude. They are expected to carry out their tasks through communicating with each other, using ontologies and inference capabilities.

- A need for *trust* and *encryption* is also present, but this is not described in much detail.

5.1.1 Proposed Architecture

To realise the Semantic Web there are several different technologies which need to be in place. Its architecture can be represented in the form of a cake of specifications and languages layered out on top of each other. This is shown in Figure 5-1, where the main components as pointed out by Berners-Lee in [10] and later described by Matthews in [77] are depicted.

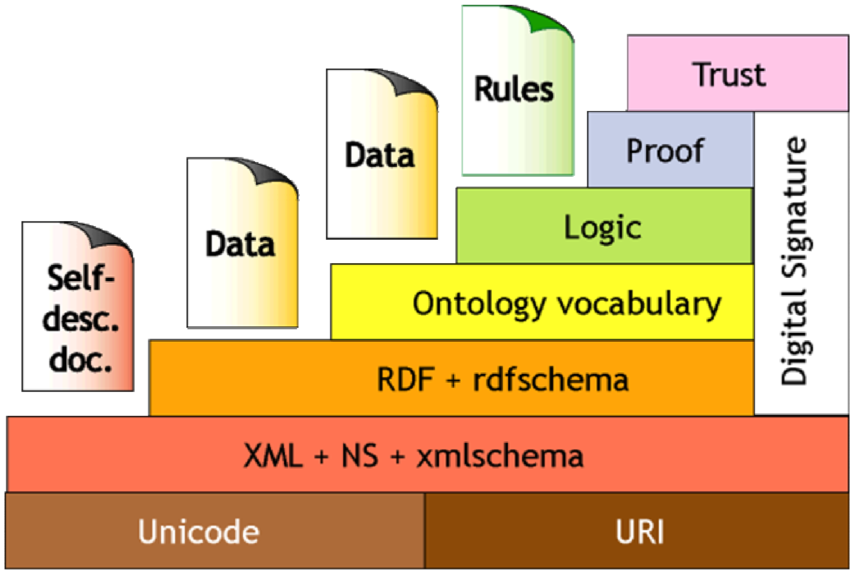


Figure 5-1 - The Layered Architecture of the Semantic Web [10]

5.1.1.1 Unicode and URI

The bottom layer of the Semantic Web architecture is made up by Unicode, which is the standard representation of characters ([105]), and Universal Resource Identifiers (URI, [11]), which is a generalised addressing mechanism for specifying a unique address for an item. A Uniform Resource Locator (URL) is a type of URI. This layer is responsible for representing characters and uniquely identifying web resources [77].

5.1.1.2 XML

The eXtensible Markup Language (XML) grew out of the demand to make HTML more flexible by allowing for the addition of arbitrary structures in web documents [32]. Its main advantage comes from the fact that it facilitates easy formation of a structured document. The technology itself has two levels. On the bottom level, it is an open standard which describe how to build a tree-based data structures using markup tags, while on a more conceptual level, it can be used as a strategy for information management. The structuring has no particular semantics to indicate what the structure means. XML plays only the role of a syntax carrier and thus corresponds to a basic syntax layer [77].

5.1.1.3 RDF

The Resource Description Framework (RDF) is a family of standards which extends the bottom two layers by allowing documents to be described in the form of metadata. This metadata is a mechanism to give meaning to the data [29].

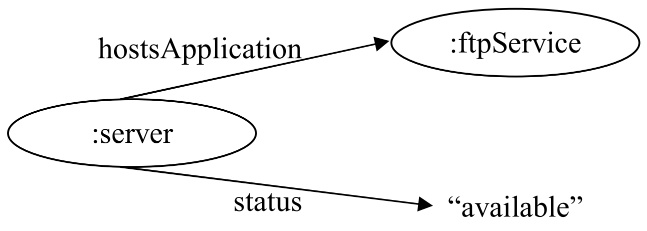


Figure 5-2 - RDF statement triple

The RDF data model is basically made up of different statements about resources. These statements are made through the use of object-attribute-value triples, see Figure 5-2. An object is a resource, i.e. anything which can be referred to using a URI. The attribute is, as one might expect, some attribute of the resource and the value is either another resource or a text string value. Through this, RDF allows one resource document to refer to and extend statements made in other resource documents [29].

5.1.1.4 RDF-S

RDF Schema (RDF-S) is an extension of RDF which defines a simple modelling language on the top of RDF and may be thought of as a simple type system for RDF [29].

RDF-S allows for the definition of classes of resources as well as the specification of domain-specific properties. These classes and properties may be further arranged in class- and property hierarchies, respectively. RDF-S also allows for an engineer to place statements to impose different restrictions on the model, e.g. range statements which restricts how the different classes and properties may be legally combined. Although this makes RDF-S seem like a likely candidate for a simple ontology language, it still has limitations. It only has a few modelling primitives and do not feature any exact, logical semantics (i.e. no precisely defined meaning) [29].

5.1.1.5 Ontologies (OWL)

The ontology layer provides even more meta-information such as relation cardinality, transitivity and so forth. This layer is extensively explained in chapters 4, 5.2 and 5.3.

5.1.1.6 Logic

This layer is supposed to add rule-based reasoning and is further explained in 5.4.

5.1.1.7 Proof

The proof layer will execute the use of rules and ontology reasoning, and evaluate these by cooperation with the trust layer which allow applications to decide whether a source should be trusted or not.

5.1.1.8 Trust

To include trust in the Semantic Web, digital signatures are used to detect unauthorised alterations to documents.

5.2 Description Logics

Description Logics (DL) are a family of knowledge representation languages which has long been advocated as a suitable tool for representing and reasoning about ontologies and have been heavily used in the Semantic Web. The Semantic Web is extensively presented in section 5.1. It has been shown that DLs can be viewed as fragments of First-Order Logic, but that it in some cases can exceed the standard First-Order Logic expressivity [13].

DLs use a way to structure knowledge which is formal, yet intuitive. The knowledge is represented by some basic building blocks which are classes, roles and individuals. Classes represent the ontology concepts, but can also be viewed as n-ary relations (or unary predicates in First-Order Logic terminology). Roles represent binary relations (binary predicates) between objects. Finally, individuals represent the concept instances. These can be viewed as constant symbols.

The allowed namespace for classes, roles and individuals form a namespace set which is usually represented as N_{DL}. This is made up of the namespaces for each disjoint set of classes, roles and individuals respectively, meaning:

$$N_{DL} = N_{DL}^{C} \cup N_{DL}^{R} \cup N_{DL}^{I}$$
$$N_{DL}^{C} \cap N_{DL}^{R} = N_{DL}^{C} \cap N_{DL}^{I} = N_{DL}^{R} \cap N_{DL}^{I} = \emptyset$$

These different elements in the namespace sets are combined with each other using operators and connectives to form DL formulas. There are several different operators and connectives that are allowed in DL. These are shown in Table 5-1.

Name	Operator
Intersection	$\cap$
Union	$\cup$
Negation	$\neg$
Exists	$\exists$
Value	$\forall$
Cardinality, min	$\leq$
Cardinality, max	$\geq$
Cardinalty, equal	$\equiv$
Set, enumeration of individuals	{...}
Inverse	$^-$
Transitive	*
Name	**Connective**
Subsumption	$\subseteq$
Defined as	$\equiv$

Table 5-1 – DL Operators and connectives [8]

Operators allow for the creation of more complex constructions called terms. As an example, we could represent the concept of a male human who has at least one human child by stating:

$$\mathbf{Human \cap Male \cap \exists\, hasChild.Human}$$

This term conceptualises the concept of a father. If such a term defines an important concept for us, we may want to give it a separate class name like Father. To express the fact that the concept of Father is the same as the term stated above, we need an axiom. Axioms are formed through the use of some connective. In this case, the axiom needed is:

$$\mathbf{Father \equiv Human \cap Male \cap \exists\, hasChild.Human}$$

All knowledge in DLs is represented through the use of such axioms. This means that every DL knowledge base is a set of DL axioms expressing knowledge about a given domain. This knowledge is made up of a terminology- and a world description, also called the *TBox* (meaning the *"Terminological Box"*) and the ABox (meaning the *"Assertional Box"*) [8].

A TBox consists of axioms dealing with concepts (e.g. Person) and roles only (hasChild, hasBrother) and are also called terminological axioms. Concepts denote a certain set of individuals, while roles represent a binary relationship between individuals (e.g. hasChild) [7]. The previously axiom presented for the concept of fatherhood (Father $\equiv$ Human $\cap$ Male $\cap$ $\exists$hasChild.Human) is a typical example of something which should go into the *TBox*.

The axioms of the ABox are also called assertional axioms and are used to assert concept and role properties of the individuals in the domain. These are typically called membership and role assertions. An example of the latter might be:

hasChild(Bob, Alice)

This is pretty straight forward and states that the individual *Bob* has a child, the individual *Alice*. Now, the next statement is a typical membership assertion:

Human∩ Male∩ Bob

This describes an individual *Bob* being a member of the intersection of the classes Male and Human (i.e. *Bob* is a male human).

5.2.1 DL Families

The higher expressive power a specific DL family possess makes its reasoning problems (5.2.3), unsurprisingly, more complex. Although a DL has to provide enough expressive power to describe the relevant properties for a certain application, it also needs to be somewhat "practical", i.e. run in realistic time and space. This means that one is faced with the classic trade-off between expressive power for computer efficiency and vice-versa [95]. The smallest propositionally closed DL is *ALC*, which are concepts constructed using $\cap$, $\cup$, $\neg$, $\exists$ and $\forall$. Examples of extensions to DL are shown in Table 5-2.

Extension	Description
S	ALC with transitive roles (R_+)
H	Role inclusion axioms (role hierarchy)
O	Nominals (singleton classes, written {x})
I	Inverse roles
N	Cardinality restrictions
F	Functional role restriction (cardinality of 1)
Q	Qualified cardinality restrictions
(D)	Datatypes

Table 5-2 –DL Extensions [8]

5.2.2 The Open World Assumption

Another feature and implication of using Description Logics is reasoning using the *open world assumption*. According to Baader et al. in [7], the open world assumption can be explained as follows:

"While a database instance represents exactly one interpretation, namely the one where classes and relations in the schema are interpreted by objects and tuples in the instance, an ABox represents many different interpretations, namely all its models. As a

consequence, absence of information in a database instance is interpreted as negative information, while absence of information in an ABox only indicates lack of knowledge." [7]

Thus, in an *open world reasoning* scheme it is assumed that unless explicitly stated otherwise, something is always possible. This means that even a failure to prove an assertion will leave us clueless about the assertion's truth of falsity, thus some statements simply remains unknown. This is in stark contrast to a *closed world reasoning* scheme where a failure to prove an assertion α leads to the conclusion that $\neg\alpha$ is the case. The use of *open world reasoning* makes DL *monotonic* which means that a DL processor should only draw conclusions that will remain valid even if additional statements were later added to the data set [7].

5.2.3 DL Reasoning

One of DL's strengths is the use of reasoning as a mechanism for inferring implicit knowledge from the explicit contained in a knowledge base, thus making it *declarative*. In [9], Baader and Sattler presented several typical reasoning tasks including subsumption, taxonomy construction, satisfiability and instance retrieval. According to Pires et al., the different reasoning tasks in DL can be divided into three general types [89]:

1. Class-level reasoning:

    - *Subsumption.* Finding subclass / superclass relations between concepts.

    - *Classification.* Concept classification of all defined concepts with respect to the subclass / superclass relation.

    - *Taxonomy construction.* Constructing the inferred class taxonomy.

    - *Satisfiability.* Checking whether there exists an interpretation of the terminology where a given concept has at least one instance.

2. Property-level reasoning:

    - *Sub-properties.* For all stated relationships, compute the implied relations.

3. Individual-level reasoning:

    - *Consistency check.* Check if an individual exist in a specific model.

    - *Realisation.* Given an individual, enumerate the most specific concept that describes it.

    - *Instance retrieval.* Enumerate all individuals who are instances of a given concept (retrieval)

Such automated reasoning support allows developers to check many more cases than could have been done manually. The mentioned checks are also valuable for designing large ontologies where multiple authors are involved as well as when integrating and sharing ontologies from various sources.

5.3 The Web Ontology Language – OWL

In 2001 the DAIM+OIL [20] language, a predecessor of DAML-ONT [72] and the OIL [34] language, was created as a web ontology language. This later formed the basis for the Web Ontology Language (OWL) which is now a WC3 recommendation [78]. It has received massive support and interest and is currently viewed upon as the official Semantic Web ontology language. OWL is built upon the RDF Schema enriched with the semantics from DAIM+OIL.

The OWL standard actually consists of three increasingly expressive dialects; OWL Lite, OWL DL and OWL Full. Each language is an extension of its predecessor and adds both to what one can legally express as well as in what one can conclude from the ontologies [78]. In [78] it is stated that the following set of relations hold, but are not symmetric:

- Every legal OWL Lite ontology is a legal OWL DL ontology.
- Every legal OWL DL ontology is a legal OWL Full ontology.
- Every valid OWL Lite conclusion is a valid OWL DL conclusion.
- Every valid OWL DL conclusion is a valid OWL Full conclusion.

Table 5-3 - Set of relations between OWL languages [78]

OWL Lite and OWL DL are both based on a logic framework called Description Logics (5.2) [7]. OWL Lite is a notational variant of Description Logics *SHIF*(D) while OWL DL is a notational variant of the Description Logic *SHOIN*(D) [56]. This is more closely explained in the following.

5.3.1 OWL Full

OWL Full is actually not a sublanguage of OWL as it contains all the OWL language constructs as well as free unconstrained use of the RDF constructs. It is meant used by designers who desire the maximum amount of expressiveness as well as the syntactic freedom of RDF. As an example, OWL Full allows for classes to be treated as individuals. This means that having the identifier "Fokker-100" act both as a class name (denoting the collection of Fokker-100 airplanes around the world) as well as an individual name (e.g. as an instance of the class AirplaneType) [27]. However, because this introduces enormous possibilities, it also introduce a serious drawback to using OWL Full as no computation guarantees for reasoning can be made. According to [27], it is also unlikely that any reasoning software will ever be able to support complete reasoning for every feature of OWL Full.

5.3.2 OWL DL

OWL DL is a sublanguage of OWL Full and received its name due to its close relation to Description Logics, a research field which has studied the logics that form the formal foundation of OWL (see 5.2). It includes all the OWL language constructs, but restricts some of their use. For example, unlike in OWL Full, a class in OWL DL may be a subclass of several other classes, but it cannot be an instance of another class. These restrictions help make OWL DL computationally decidable [78]. However, it has been shown that key inference problems have NEXPTIME complexity [56].

The OWL DL dialect is related to the *description logic* family *SHOIN*(D), which is a very expressive form of description logics. This includes transitive roles, inverse roles, role hierarchy (is-a), nominals, arbitrary cardinality restrictions and some datatypes. See section 5.2.1 for more details on these properties. The dialect may be somewhat difficult to present to new users since it is possible to build very complex class descriptions and restrictions by using boolean combinations of e.g. union, complement and intersection [8].

5.3.3 OWL Lite

In order to deal with the complications of OWL DL, OWL Lite, a subset of OWL DL, was created. OWL Lite is a variant of the description logics familiy *SHIF*(D) [56] which prohibits the use of union and complements as well as restricting cardinality constraints to be purely binary, i.e. only cardinality values of 0 or 1 are accepted [78].

OWL Lite is basically meant for those users who primarily need some sort of basic classification hierarchy and simple constraints. Because of its simpler nature and lower formal complexity, OWL Lite is easier to reason with than other OWL dialects and its inference complexity has been shown to be EXPTIME [56].

5.4 SWRL - Semantic Web Rule Language

Description Logics, and thus also OWL, allow for knowledge to be expressed using concepts and world descriptions. There are however several kinds of knowledge and restrictions which are impossible to express using Description Logics. Some of these are explained in section 9.3.2. These limitations could, at least to some extent, be overcome by establishing a rule layer which could complement OWL. Although there have been many proposals on how accomplish this and enhance OWL knowledge bases with rules, the Semantic Web Rule Language (SWRL) is probable the most established. SWRL is basically a combination of OWL and RuleML ([104]), and was originally proposed by Horrocks et al. in [54]. The rule extension is done by *rule axioms* to the OWL set of axioms. These axioms are a simple form of Horn-style rules.

A SWRL rule axiom is made up by an antecedent (body) and a consequent (head), which both consist of zero or more atoms. However, as rule axioms with zero atoms in either part are not adding any expressivity to OWL, these are better expressed in OWL directly. When there are multiple atoms in the head or the body, the atoms are treated in conjunction. The different possible rule atoms and their descriptions are shown in [54].

Atom	Description
C(x)	*C* stands for a given *OWL class* or *data range* while *x* is either a *variable*, *OWL individual* or *OWL data value*. This is often referred to as a *concept* or *class atom*.
P(x, y)	*P* is an *OWL property* and *x* is either a *variable* or an *OWL individual*. *y* can be a *variable*, *OWL individual* or *OWL data value*. The *P atom* is often referred as a *role* or *property atom*.
sameAs(x, y)	This states that *x* and *y* are the *same*, where *x* and *y* are *variables* or *OWL individuals*
differentFrom(x, y)	Similar to sameAs, this statement tells us that the provided *x* and *y* are *different* from each other. *x* and *y* are *variables* or *OWL individuals*
builtIn(r, x, …, z)	The *builtIn* atom is used whenever the engineer wants to use one of the *built-in relations* in SWRL. *r* is here the *built-in relation* and *x, …, z* are *OWL data values*. *Built-in relations* may be e.g. math built inns like *add* or *multiply*, or string built inn relations like *startsWith* etc. A complete list of the suggested built inn relations can be found in [54]

Table 5-4 - SWRL rule atoms [54]

A simple example of the SWRL rules, which also shows how it helps to overcome some the limitations described in 9.3.2, is to assert that the combination of the *hasParent* and *hasBrother* property implies the *hasUncle* property. This can be written as [54]:

$$\textbf{hasParent(?x, ?y) ^ hasBrother(?y, ?z)} \Rightarrow \textbf{hasUncle(?x, ?z)}$$

From this rule one can infer that if the individual *John* had the individual *Mary* as a *parent*, and that *Mary* had the individual *Bill* as a *brother*, then *John* would have *Bill* as an *uncle* [54].

A SWRL rule is treated as a logical implication between its head and its body. This means that whenever the conditions in the rule body hold, then the conditions in the rule head must also hold.

There are however, some problems related to the use of SWRL rules. The use of logical implication means also that the contraposition law holds for the rule. Although both OWL DL and SWRL are decidable logics on their own, this kind of rule-extension of OWL makes it undecidable [57]. Because of this, it is likely that many implementations will only provide partial support for SWRL, e.g. in the form of SRWL Lite & Full [87]. Currently, there are no SWRL rule engines on the market which has implemented all features of SWRL [92], but some provide sufficient functionality for preliminary testing (see section 8.4 for experimental data).

5.5 OWL-S: Service Descriptions using OWL

OWL-S is an ontology for describing services and is specified using OWL. Its long term goal is to aid service providers and consumers to describe services in a semantic way for automatic discovery, composition and consummation [26].

The OWL-S upper ontology consists of four major components (see Figure 5-3) [26]:

- A *Service*: This is the main concept and serves as an organisational point of reference when declaring services. Every service is declared by creating an instance of this concept.

- A *Service Profile*: The Profile concept provides a vocabulary used to characterise what the service does on a high level, so that an agent can determine if the service meets its needs. This includes describing both functional and non-functional properties which can be used for locating services on their semantic description.

- A *Service Process Model*: This is a general model of the service and describes how to ask for the service while detailing the semantic content of each request, thus including preconditions, results and processes which are carried out. These processes can be atomic or composite.

- A *Service Grounding*: The grounding describes in detail how an agent can access a service. Typically, this is done by specifying a communication protocol, the format of messages and other service-specific details.

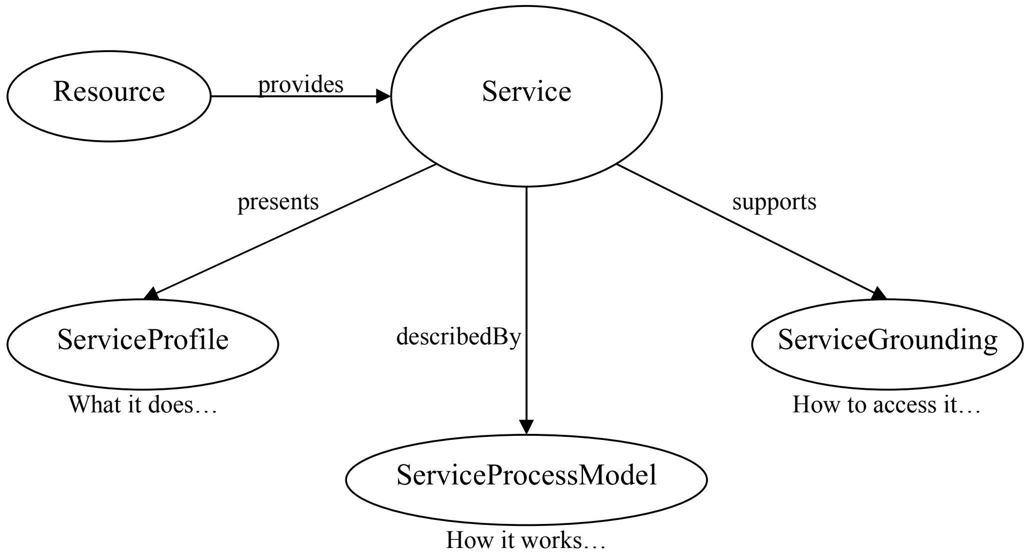


Figure 5-3 – OWL-S ontology [26]

Together, these three latter concepts are designed to give a total picture of the capabilities of a service. Being the most interesting for use in this thesis, which will be shown in chapter 7, the ServiceProcessModel is further presented in the following.

5.5.1 The OWL-S Service Process Model

The OWL-S Process Model is organised as a combined workflow of processes. Each of these processes is described by three separate components: inputs, preconditions and results. Results specify what the output of the service is under a given condition [26]. Processes in the work flow are related to each other by a data flow and by a control flow. The control flow allows for a specification of the temporal relation between processes [4]. OWL-S supports a wide range of control flow mechanisms including sequences, synchronisation points, loops, splits and conditional statements [26]. A simple example of a process utilising such a control flow can be seen in Figure 5-4. This is the UnregisterWithGatekeeper procedure of an H.323 endpoint. First the process checks if there are any active calls in the unit. If there are these are ended one by one until the unit have disengaged all its calls. When this is done, the process will unregister and exit. If there are no active calls, the unit will simply unregister and exit directly.

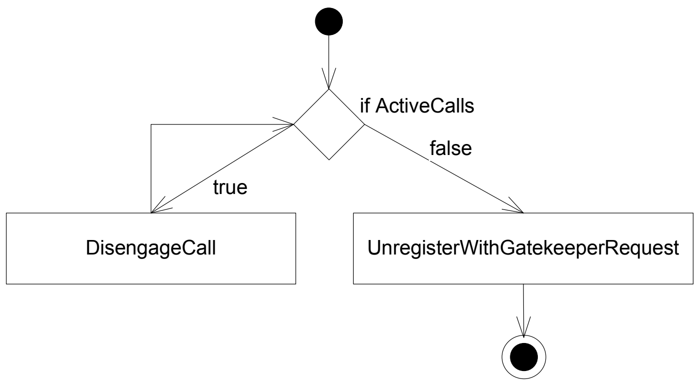


Figure 5-4 – OWL-S model of an H.323 Endpoint's UnregisterWithGatekeeper procedure

OWL-S distinguishes between atomic and composite processes. Atomic processes are, as the name indicates, indivisible services and will to the service requestor seem to be carried out in one single step. Composite processes however, are decomposable into other atomic and composite services. This decomposition may be further specified using control constructs to describe the relation between the inner processes.

As the OWL-S specification is provided using a machine-readable OWL knowledge base, itself is also machine readable. A specific model is created by inserting instances into the OWL-encoded scheme. Thus, any OWL-S process model can be read and in turn executed by a machine agent.

6 Thesis Methodologies

"If Edison had a needle to find in a haystack, he would proceed at once with the diligence of the bee to examine straw after straw until he found the object of his search... I was a sorry witness of such doings, knowing that a little theory and calculation would have saved him ninety percent of his labour."

- Nikola Tesla

A research process is a careful investigation which hopefully will lead to the discovery and new insight in a field of interest. In [18] Buchler refer to the expression "the power of a method" when he promotes the use of methods and methodologies in research. It is further emphasised that:

"(...) without methods, we are left with chance." [18]

Methods are used to facilitate better results which are achieved more efficiently [16]. This is rather nicely expressed in Tesla's introductory quote to this chapter. Throughout this thesis, several methodologies have been used to drive the research through the different phases. These phases have included: literature review, system development, ontology case modelling, experiments and finally evaluation. The different phases are shown in Figure 6-1.

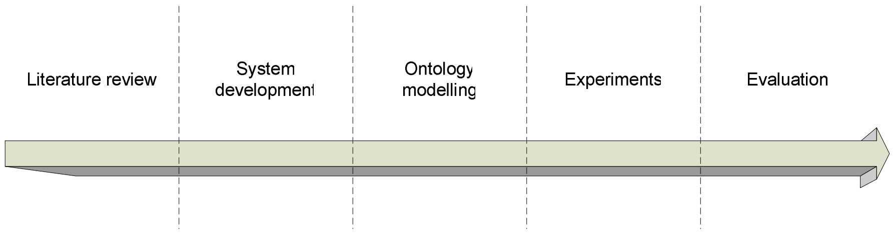


Figure 6-1 – The five different phases of the master's thesis research

In the following sections the research methodologies used throughout this thesis are presented.

6.1 Literature Review

In [30] Gay and Airasian describe the process of doing an analysis of previous literature as:

"(...) systematically identifying, locating, and analyzing documents containing information related to the research problem." [41]

This process is expected to produce a literature review which according to Creswell ([17]) can be defined as:

"(...) a written summary of articles, books, and other documents that describes the past and current state of knowledge about a topic." [23]

Other documents can include abstracts, reviews, monographs, dissertations, other research reports and electronic media [23].

There are, according to Cooper ([22]), two types of literature reviews – the integrative research review and the theoretical review. Theoretical reviews address and compare different studies with a focus on their width and consistency. Integrative research reviews, on the other hand, "summarize past research by drawing overall conclusions from many separate studies that are believed to address related or identical hypotheses". These reviews are also often done in order to compile the previous studies into a more complete contribution to a specific domain. Although most research papers usually have a section where previous research literature is discussed in relation to the project, the literature might also be used in other parts of the research projects, such as data analysis [16].

Since a single research study rarely will provide a general and absolute answer to a research question, the project will often be strengthened by combining its results with other studies [22]. However, unless a literature review is thorough and fair, it is of little scientific value [70]. This, being a question of reliability, can be seen in relation to the quality of the work behind the review [52]. It is an empirical question of whether the findings of two independent reviews will be roughly the same, i.e. that it is possible for someone else to reproduce the essence of the findings given that they use the same approach [51]. According to Dellinger [28], the "review of the literature is inherently an interpretive and value-driven process". Dellinger further notes that the problems associated with a poor literature review can culminate and affect the quality the single study, which may in turn manifest itself in poor conceptualisation, design, measurement and methodology [28]. Because of these problems, Kitchenham suggests in [70] procedures for performing systematic reviews including planning, conducting and reporting. Specifically this approach included constructing planning check lists and a review protocol used when reviewing a paper.

This thesis adopted and used the review protocol from Kitchenham's systematic approach to the literature review, to establish an overview of sources that were relevant to the research project. These sources were then used in an integrative research review approach to the literature survey.

6.2 Software Engineering

Computer science is still a young field and so is software engineering. According to IEEE ([62]) System Engineering is:

> *"(...) is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software." [62]*

This is quite similar to Bræk's definition of a methodology as a [16]:

> *"(...) system of methods and principles" [16]*

This section will present the methodology that the software development process in this project has focused on. The goal of such methodologies is to help developers make better systems more efficiently. In order to achieve this, Bræk stated in [16] that there is a need to separate the *functionality* in terms of logical behaviour from the way that it is implemented and that one should model the functionality using a suitable conceptual abstraction.

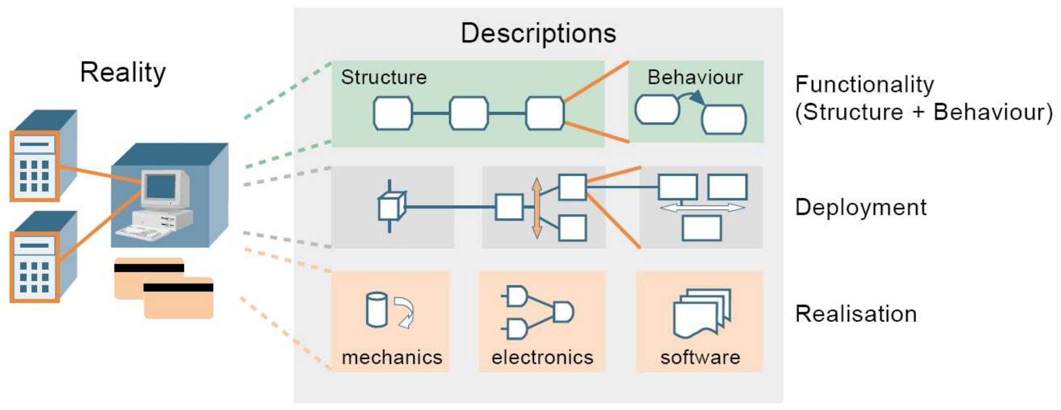


Figure 6-2 – Three viewpoints of reality [16]

It is suggested using three separate viewpoints of reality that are largely independent. These are, as shown in Figure 6-2 [16]:

- Functionality
    - A conceptual abstraction of the logical behaviour trying to describe this as clearly as possible in a way that is easily communicated between developers. It is supposed to provide a view where the system can understood independent of its realisation.
- Deployment
    - The deployment defines a mapping between the functionality and the realisation. This is done by describing the realisation on a high level identifying the technologies used and describing how and where the functionality is realised.
- Realisation
    - This is the technical definition of how the realisation is achieved in terms of technologies used, mechanics, hardware and electronics. Several realisations will probably be possible based on a given functionality.

When describing the functionality, Bræk advocated *"separation of concerns"*. This means identifying aspects that are as independent as possible and then describing them separately. By doing so, each description's complexity is reduced, thus allowing them to be expressed more clearly. Separation of concerns will also help to increase the modularity of description set since this allows descriptions concerning independent concepts to be changed independently of each other. Finally, if different aspects of the system require different skills or knowledge, separation will help to utilise different kinds of expertise better. It is also noted that the developer should be aware that separation is not a goal in itself. Very little is gained if the separated modules are too dependent on each other [16].

Although other modelling options are available, Bræk recommends further that the Specification and Description Language, SDL, together with Message Sequence Charts, MSCs, should be used to describe the functionality view. Not only do they provide the necessary clarity, but they are also formal languages suitable for use in the so called translation approach. This is one of the two different approaches to software engineering which Bræk presents in [16], where the other is the elaboration approach. In the latter, the functionality of the system is often described using more or less informal languages with incomplete semantics and is thus somewhat incomplete. In the former the functionality is, as previously presented, described as completely as possible [16].

If an elaboration approach is used, the actual realisation often ends up as up as the *"only complete view of the system"*, and is consequently often the only one being maintained. In other words, this means that the only up-to-date documentation of the system is the system itself. This is in stark contrast to the translation approach where the functionality can be completely defined, analysed and simulated before implementation. In addition to this, the functional description can remain valid for the realisation and serve as documentation [16].

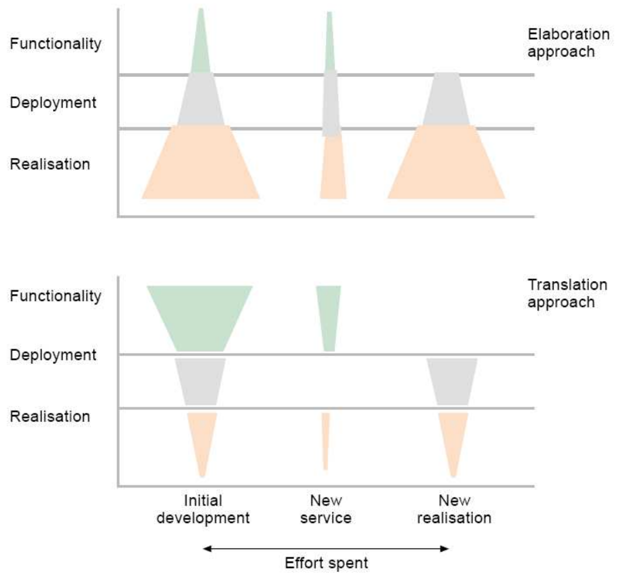


Figure 6-3 – Differences in where effort between the elaboration and the translation approach [16]

In Figure 6-3, the difference between these two approaches is shown. As depicted, the translation approach focuses on spending effort in modelling functionality, while the more common elaboration approach focuses its efforts on the realisation phase.

6.3 Ontological Engineering

Building an ontology means building and designing a model, thus, when we decide how to represent something in an ontology, we are making design decisions. Therefore, the strength of any ontology lies in its design [84]. According to Jakkilinki et al. in [64]:

“At present developing ontologies is an art rather than a science.” [64]

This statement support the fact that although ontological engineers have designed numerous ontologies, ontological engineering is a young field and no solid and complete methodologies for creating ontologies exist. This is however, continuously changing and according to Jones et al. in [67] there are now several promising ontology development methodologies (Uschold [109], Uschold and Gruninger [110], Gomez-Perez [44], Fernandez et al. [36]). One of them, and perhaps the most widely accepted is the Uschold & Gruninger methodology [110] which, as illustrated in Figure 6-4, consists of five separate stages. These are identification of purpose and scope, knowledge acquisition and conceptualisation, integrating reuse of other ontologies, formal specification and finally evaluation and documentation.

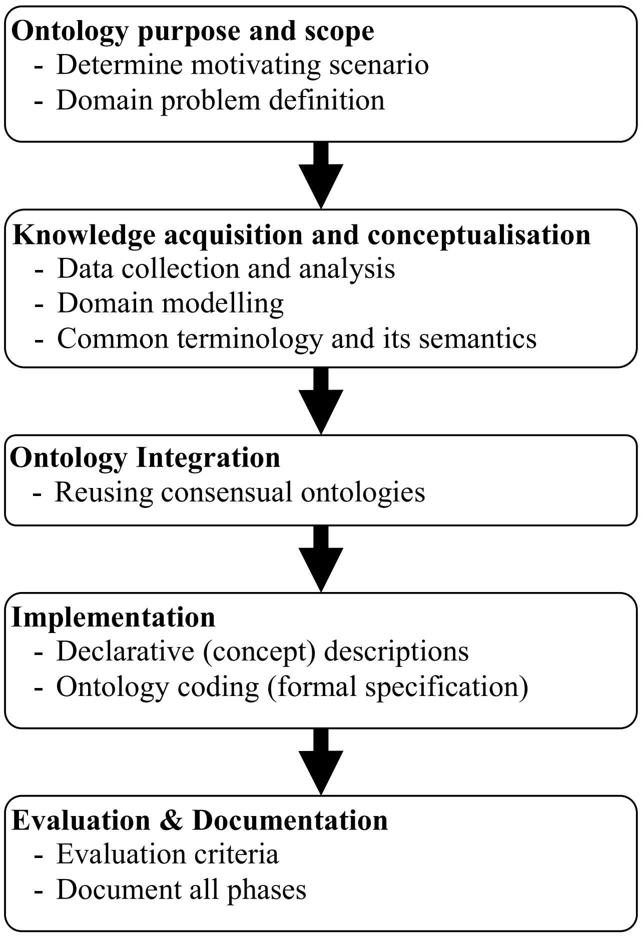

Ontology purpose and scope
- Determine motivating scenario
- Domain problem definition
Knowledge acquisition and conceptualisation
- Data collection and analysis
- Domain modelling
- Common terminology and its semantics
Ontology Integration
- Reusing consensual ontologies
Implementation
- Declarative (concept) descriptions
- Ontology coding (formal specification)
Evaluation & Documentation
- Evaluation criteria
- Document all phases

Figure 6-4 – The Uschold and Gruninger ontological engineering methodology [110]

This is the methodology used in this thesis when constructing ontologies. In the following, each stage is briefly presented and explained.

6.3.1 Ontology Purpose and Scope

At this stage of the process, the goal is to define the ontology's purpose and scope. This includes an informal description of how it will be used, who is supposed to use it i.e. its users as well as the scope of the ontology. Furthermore, there should be made a first attempt to define some of the domain's terminology, informal descriptions of the different entities, related terms, their properties and relationships.

6.3.2 Knowledge Acquisition and Conceptualisation

This stage focuses on acquiring different kinds of knowledge about the given domain. This means use of traditional methods such as domain text analysis and interviews with domain experts. At this point, other known techniques for knowledge elicitation and acquisition can, and probably should be, applied. Examples of such techniques can be found in [46], [103] and [110]. This process will lead to a domain conceptualisation. This will contain the identification and representation of the domain concepts, their properties, relations, instances and will associate them with the domain terms. Each term and its relations are represented in some sort of informal language. This stage is supposed to produce complete definitions of all the different terms used by the domain actors including their meanings and relationships.

6.3.3 Ontology Integration

Since designing ontologies is a very labour-intensive and time consuming work, ontologies usually only cover a given domain of interest in a detailed way. It is also difficult to construct ontologies from scratch. The ontology integration stage is concerned about the reuse of existing ontologies when building a new one. This is done in order to save time and effort, and to try to enforce some continuity across different ontologies. The reuse of ontologies can be accomplished e.g. by reusing terms from large and consensual ontologies. Examples of such published ontologies are the Bibliographic ontology, the Document ontology, the Enterprise ontology, the OpenCyC ontology and so forth.

Guarino classified in [48] ontologies based on their level of generality into *top-level ontologies, domain ontologies, task ontologies* and *application ontologies.* These can be seen in Figure 6-5.

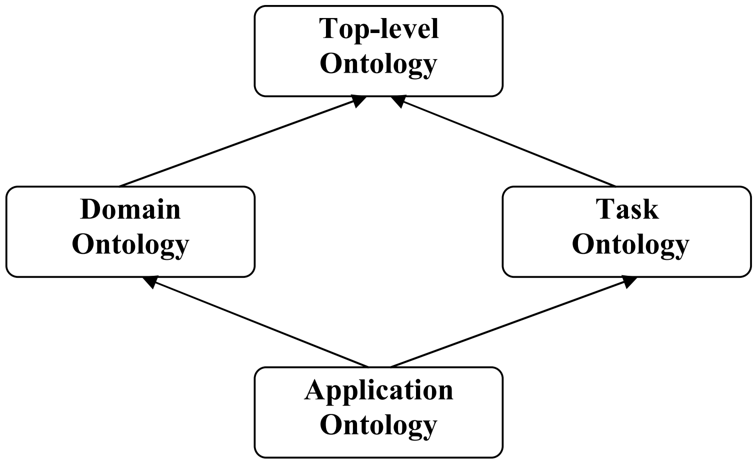


Figure 6-5 - The different kinds of ontologies, according to their level of dependence [48]

Top-level ontologies are used to describe very general concepts such as time, space, matter, object etc. which are completely independent of a particular domain. It therefore seems reasonable to have widely accepted and consensual top-level ontologies for large groups of users [48].

Domain and task ontologies describe the vocabulary related to a generic domain or task, by specialising the terms defined in the top-level ontology [48].

Application ontologies describe concepts depending on both a particular domain and task, and are often specialisations of both. These concepts often correspond to the different roles played by domain entities while performing a certain task, e.g. replaceable unit or spare component [48].

6.3.4 Implementation

During the implementation stage, the ontology should be represented formally by using some sort of formal representation language, such as *OWL* (see section 5.2). This can be done by hand (cumbersome) or through the use of ontology modelling tools such as Stanford's Protégé. This stage includes formalising each term as well as their constraints. The terms are represented through ontology concepts, relations, restrictions and instances. The different restrictions can be modelled using some form of logic, depending on the representation language. Most common are first-order logic and description logic.

6.3.5 Evaluation & Documentation

As an ontology often is used as an artefact for communication between its designer and its users it is essential that it communicates the intended meaning of the defined terms efficiently and with a high degree of clarity [46]. It should be well documented and its purpose should be clearly stated.

7 OSCOS – Ontology based Self-Configuration System

In this chapter the proposed architecture for a Semantic Web driven self-configuring system is presented along with a thorough discussion of each of the necessary parts for realising such a system. The architecture is visualised and explained to the reader through the use of block diagrams, behavioural state machines and sequence diagrams.

7.1 System Design

In [17] Bræk and Floch stated that there are two principal system architectures: the agent oriented and the server oriented. An agent oriented system follows a principle that the system should be structured so that it mirrors objects in the environment it serves. This is an approach known to give stable and adaptable designs [19]. According to Sanders [94]:

> *“These objects provide natural areas for adaptation of functionality (...) depending on what aspect is being treated.” [94]*

This fits well with this thesis and should therefore be taken into account when designing OSCOS. Consequently, the obvious design of OSCOS would be to place much of the functionality in the respective parts. However, due to the computational intensity of some of the necessary parts, it is evident that some functionality has to be centralised. This is discussed when presenting each unit, as well as in the evaluation chapter 9.

The proposed OSCOS system architecture is an event-based architecture. It is based on IBM’s previous work, but adapts this to an agent oriented, Semantic Web driven approach in accordance with the thesis’ research statement. Specifically, the architecture is based on a system where each entity publishes and maintains a knowledge base encoded in OWL of itself and its surroundings. Each system has an event handler component which acts on events and send these alongside the system’s knowledge base to a specified repair-service. In the repair service, SWRL rules are used to classify pre-defined error-scenarios, where each error-scenario has its own repair-procedure encoded using OWL-S. The classified error-scenario is then sent back to the unit’s self-reconfiguration component where the OWL-S procedure is executed.

The architecture is divided into three separate building blocks that can be distributed in a computer network. Each component is discussed in detail in section 7.1.8.

7.1.1 Choosing a Principle for System Monitoring

There are in essence two approaches to constructing a monitoring system for use in autonomic computing; the event-based architecture or the continuous-monitoring architecture [93]. As suggested by its name, the event-based architecture is founded around the event, which is an indication that something of significance has occurred. This indication is then sent further to an event handler which will then decide on future actions that need to be taken [24], typically making it a reactive system [115].The continuous-monitoring architecture however, uses a monitoring application which runs continuous analysis of the system regardless of any events.

This has some advantages as to responsiveness and solving problems both proactively as well as reactively [115], but requires massive amount of computing power [93]. Since most often expectations will be met and no problem will be detected, the continuous-monitor architecture will also introduce a massive overhead [71]. An event-based architecture offers, according to Dashofy et al. in [24]:

> *"(...)a significant degree of [...] autonomy of components which we feel is necessary for software repair 'without foresight' – that is without the types of repairs that can be performed being explicitly coded into the individual components" [24]*

It is further stated, both by Dashofy et al. in [24] and by Medvidovic et al in [79] that event-based wrappers can easily be developed for a large percentage of off-the-shelf components, even if they were not developed to take advantage of event-based communication.

Continuous-monitoring systems have an edge over event-based systems when it comes to solving problems proactively and when considering responsiveness. However, the advantages of event-based systems, as shown by Dashofy et al. ([24]) and Medvidovic ([79]), as well as the massive overhead of continuous-monitoring as shown by Laddaga ([71]) led to this thesis suggesting that the OSCOS system architecture should be constructed as an event-based one. This fits well into the suggested autonomic computing architecture published by IBM [61] ,which is thoroughly explained in chapter 3.2.

7.1.2 Making Units More Self-Aware

As mentioned both in chapters 1.1 and 3.2; one of the key components of the foundation in which any autonomic system can be based on is self-awareness. Eracar propose in [31] that in order for any system to become fully self-aware, it needs to implement and use self-modelling principles. This is in accordance with Albus which in [3] states that:

> *"The system must have an internal representation (world model) of what it feels and experiences as it perceives entities, events, and situations in the world. It must have an internal model that captures the richness of what it knows and learns, and a mechanism for computing values and priorities that enables it to decide what it wishes to do."[3]*

Amongst other knowledge representation forms, the ontology concept presented in chapter 4 has the required characteristics for such a representation. Ng states in [83] that:

> *"The use of formal knowledge representation can potentially lead to a cleaner paradigm to describe, maintain and control the detection and diagnostic processes." [83]*

The use of ontologies for formal knowledge representation in intelligent autonomous agents are supported by Yang and Calmet who state in [117] that one of the key reasons for using these are that they:

> *"(...) enable the representation of background knowledge about a domain in a machine understandable form. [...] Ontologies can [also] excellently represent the organizational structure of large complex domains" [117]*

On the background of this previous work and research, this thesis suggests the use of ontologies, or more specifically, ontologies encoded in OWL to provide a basis for modelling the knowledge a unit has about itself and others. OWL is chosen for several reasons: it supports the required formality to allow machine processing and reasoning, and it facilitates distributed development by solid consistency testing. It also provides easy interoperability between units. The generated files do typically not contain any definitions, but rather refers to the main domain ontology which contains the definitions and restrictions of the domain. As it is simply referred, this means that it does not have to be physically stored as a part of the unit's ontology. In the OSCOS architecture, this main ontology resides on a special ontology server, the ontology repository. This is a very good advantage since it, among other things, allows for easier maintenance of the domain ontology [68].

7.1.3 Unit Ontology Maintenance

Each unit will convert its knowledge into ontology instances and insert these into its own knowledge base. These knowledge bases will typically contain only these ontology instances, as the actual ontology specification schema can reside on a central server and later be simply referred to whenever needed.

For units to be able to store their knowledge as ontology instances in knowledge bases, they need some sort of mapping function. Amongst others, the Java Reflection API is one way of realising this component. The API enables java code to examine classes and objects at run time and allows for a generic mapping function to analyse and study the entire contents of the java virtual machine and then place this directly into the OWL knowledge base. This has been shown possible by several previous research projects (e.g. [1], [30] and [114]). However, it is stated by Ding et al in [30] that this mapping into ontology instances can be a very a time-consuming process. This means that constructing the entire unit knowledge base from scratch may, especially for complex units, take an unacceptable amount of time. As most real-time communication systems require short response time, this might make a complete update on demand unsuited, and thus making the incremental update scheme more desirable.

Another approach for realising this component is the one depicted in Figure 7-8 and Figure 7-10, where the component registers with the Ontology Object Mapper, attaches its data, and keeps on notifying it of the changes which occur in the component. However, not only could such a process be time-consuming, but requiring a unit to always update its knowledge base after some routine change might also cause unnecessary complications to its inner working. Because of this, it may not be desirable to constantly keep the unit's knowledge base up-to-date and basically trade-offs has to be made.

Real-life implementations needs to be aware of and fully assess all of these issues, before finding the correct trade off between computational overhead and complexity. This while keeping response times down as well as the unit's knowledge base coherent and up-to-date. The best trade-off will most likely vary based on the computational capacity of the unit, its complexity and finally the system requirements and sensitivity regarding response time and latency.

7.1.4 Choosing a Suitable OWL Dialect

System developers adopting Semantic Web technologies and OWL need to choose the sub-language that best fit their needs. The choice between OWL Lite and OWL DL generally depends on the extent to which the developers require the expressiveness of OWL DL or Full. If the ontology can be modelled without the use of concepts such as unions, complements or unary cardinality expressions, OWL Lite might suffice, and should then probably be chosen as it is easier to reason over.

Choosing between OWL DL and OWL Full will depend on whether the developer require the meta-modelling facilities of OWL Full, such as requiring classes to be treated as individuals in certain situations. However, when using OWL Full, reasoning support will be less predictable as no complete implementation of OWL Full reasoning currently exist [27].

In this thesis, OWL DL was chosen since OWL Lite did not meet some of the modelling requirements and as there was not sufficiently good reasons for using OWL Full.

7.1.5 Ontology Interoperability Approaches

Ciocoiu et al. proposes in [20] two approaches for how one can use ontologies in achieving interoperability between units from different vendors. Firstly, the *standardisation approach* propose using a single common, shared and standardised ontology which all producers use to represent all their different units. The second approach uses the common ontology as an *interlingua,* or *reference ontology*, and requires own mapping functions to be written between each local ontology and the common domain ontology [20].

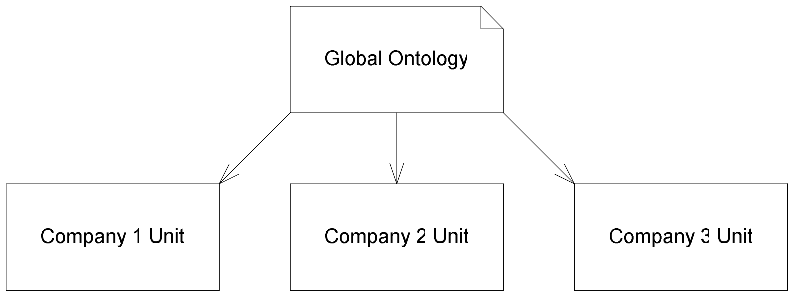


Figure 7-1 – Structure when using the single ontology *standardisation approach* [20]

The single ontology approach use, as shown in Figure 7-1, one global ontology which provides the different units with the semantic specification and must thus contain such a specification model for each of the units in the system. The ontology may be modularised, i.e. a combination of several specialised ontologies, to hinder it from becoming too large and monolithic. Although the *standardisation approach* works well in situations where all have nearly the same view of the domain, this is rarely the case and it often become hard to find the minimal ontology commitment [46]. Single ontology approaches are also more sensitive to changes in the different entities as changes to any producer's units can often affect the conceptualisation of the domain, and thus requiring changes to the ontology [20].

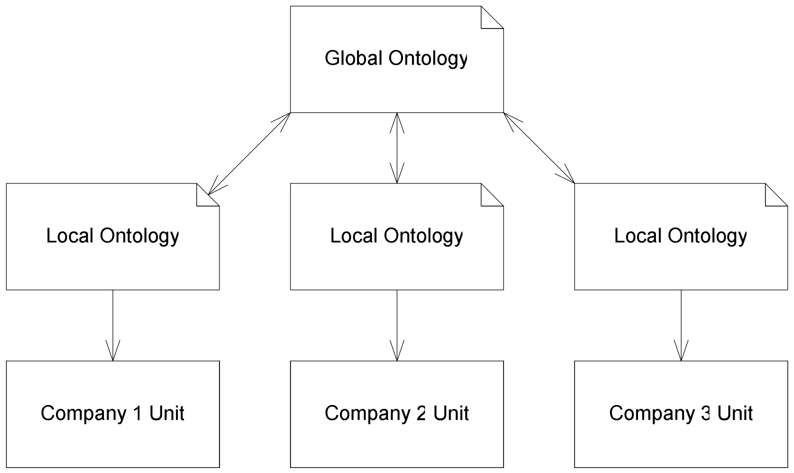


Figure 7-2 – Ontology structure when applying a *interlingua* approach [20]

In the *interlingua* approach, each unit may use a local ontology which is mapped onto the global ontology whenever needed. This approach is shown in Figure 7-2. The local ontologies may be refinement of the global ontology or an independent representation. Some important advantages of this approach are that new ontologies can be added without the need for modifications, and that the use of a common ontology makes the source ontologies easily comparable [20].

Based on the advantages and suitability of the *interlingua* approach, this thesis suggests using such a common global ontology. All units should map themselves or their ontology onto this whenever they communicate with other OSCOS enabled units. Doing so allows for an easier introduction of new units in new domains as the only mapping needed is between the new unit and the global ontology. It also support evolution of local ontologies and make the local ontologies easily comparable [20].

7.1.6 Classifying Error-Scenarios

After a unit has experienced an error and made sure that its knowledge base is up-to-date, it will need a reasoning process to determine what went wrong. Lin et al. suggests in [74] that early self-repair systems should use rules generated by human experts to determine error-scenarios. The unit error classification will therefore initially be carried out using a rule-based architecture. How these rules are derived is considered out of scope for this thesis and is left for future work.

While it is possible to express some of the rules necessary for classification directly in OWL by using *description logics* there are, as pointed out in section 9.3.2, many limitations in OWL's expressivity. It is stated by Horrocks and Patel-Scheider in [57] that many of the limitations of OWL stem to the fact that, while the language features a relatively rich set of class constructors, the language provided to assess properties is much weaker [57]. Since it is primarily the properties of a system which need to be assessed in the self-configuration architecture discussed in this thesis, it is therefore suggested using the SWRL rule-extension presented in section 5.4 to specify and carry out the actual classification.

This is specifically achieved by sending the unit's knowledge base to a classification service which will collect any lacking necessary information by examining the references annotated in the received knowledge base. After the collection is complete, the knowledge base is sent to a SWRL rule engine which checks it for any matches against the existing known error detections rules contained in the system rule base.

7.1.7 Resolving Error-Scenarios

When a specific error-scenario has been classified by the reasoning process, the unit still need to be told how it should resolve this scenario. This could range from a single command to a complex workflow [61]. OWL-S provide, as described in 5.5, a service process model which model and describe how a client can use a service including preconditions, results and the actual processes carried out by the service. These processes can be either atomic or composite. The latter contains detailed descriptions on how the service is executed following a set of control constructs, e.g. if-then-else, repeat-while etc.

An OWL-S process model includes both a formal definition of the needed processes, as well as how a managing application should execute this according to the process' control constructs. This thesis suggests therefore using OWL-S process model descriptions in runtime to explain to a unit how to recover from an error. This means that a generic managing application capable of processing OWL-S ontologies should be able to recover from the error simply by executing the process model as stated. This approach is used in a similar fashion by Vergara et al. in [112] where it was used to define management interfaces based on Web Services. It is stated [112]:

> *"The OWL-S ontology can be useful to self-describe how to manage a network resource, which can be important if the resource does not use standard management information. In this case, a manager can download the OWL-S description of the management interface and manage the resource by interpreting this description" [112]*

Specifically, the manager will process and execute the OWL-S encoded procedure as it is modelled. This includes parsing the OWL-S document, assessing its control structure and executing the services as specified. Such parsing and execution can be realised in Java e.g. by utilising the OWL-S API. This API provides a simple programmatic way to execute OWL-S process models. However, it does not currently have support for conditionals such as *if-then-else* and *repeat-until*, but these will be added in future releases according to its authors.

To be able to execute these procedures in a useful manner, the OSCOS architecture needs to have a complete OWL-S encoded view of the atomic and composite services provided by the unit. When a new error-scenario is identified, a fix for the recovery from this scenario will be put together using these services alongside the provided OWL-S control constructs.

7.1.8 Proposed Architecture

The OSCOS system contains, as shown in Figure 7-3, three main components: the terminal agent, the ontology repository and the repair service. The physical distribution of the components in an implementation will vary on the computational capacity of the entities. An obvious solution would be to assign an own computing unit for each software component.

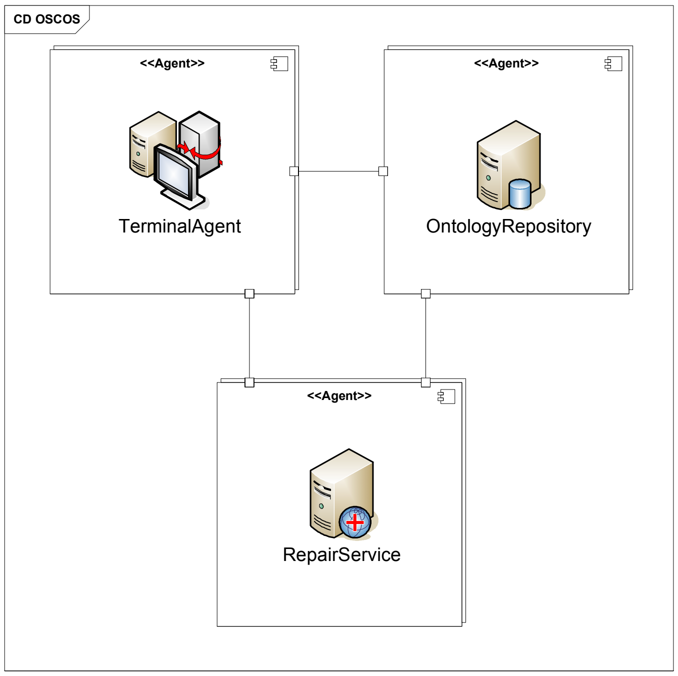


Figure 7-3 – OSCOS System Components

7.1.8.1 TerminalAgent

As shown in Figure 7-4, the TerminalAgent is made up of four inner parts; the SoftwareComponent, the OntologyMapper, the SelfReconfigurationEngine and the EventHandler.

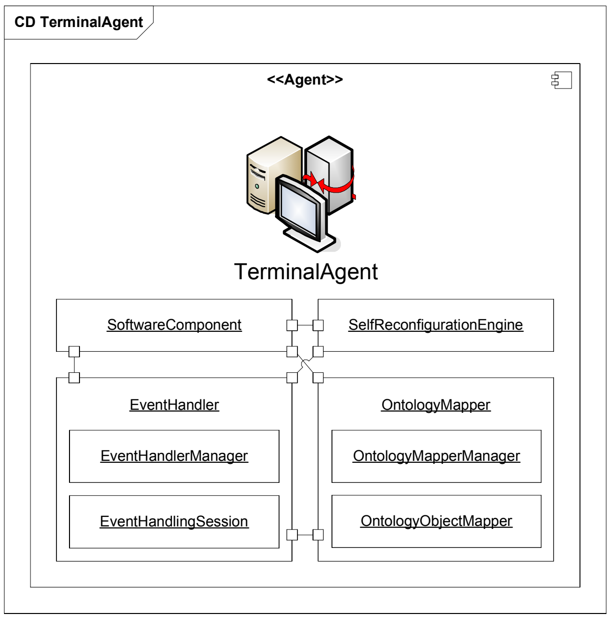


Figure 7-4 – Composition of the TerminalAgent component

7.1.8.1.1 Software Component

The software component has generally the main task of running the unit and might be e.g. the operating system of a cellular phone or any other computing unit. This component refers to the managed resource part of the IBM reference architecture and therefore needs to implement the sensor and effector interfaces so that it can communicate and share information with, as well as receive instructions from the control loop. Whenever the component experience an error, an event representing the error is sent to the EventHandler.

7.1.8.1.2 EventHandler

Connected to the sensor interfaces is the EventHandler (EH). This monitors the operation of the generic system, receiving and dealing with events as they come. It is constructed by two processes; the EventHandlerManager (EHM) and the EventHandlingSession (EHS).

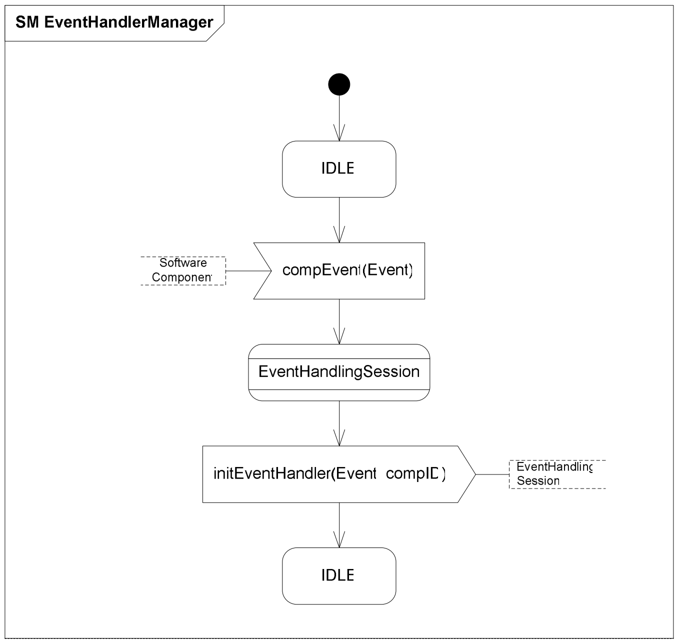


Figure 7-5 – The state machine describing the EventHandlerManager

As shown in Figure 7-5, whenever a component sends an event to the EH, it is first processed by the EHM, which then creates an EHS. The EHS will handle the rest of the error handling and its basic behaviour can be seen in Figure 7-6.

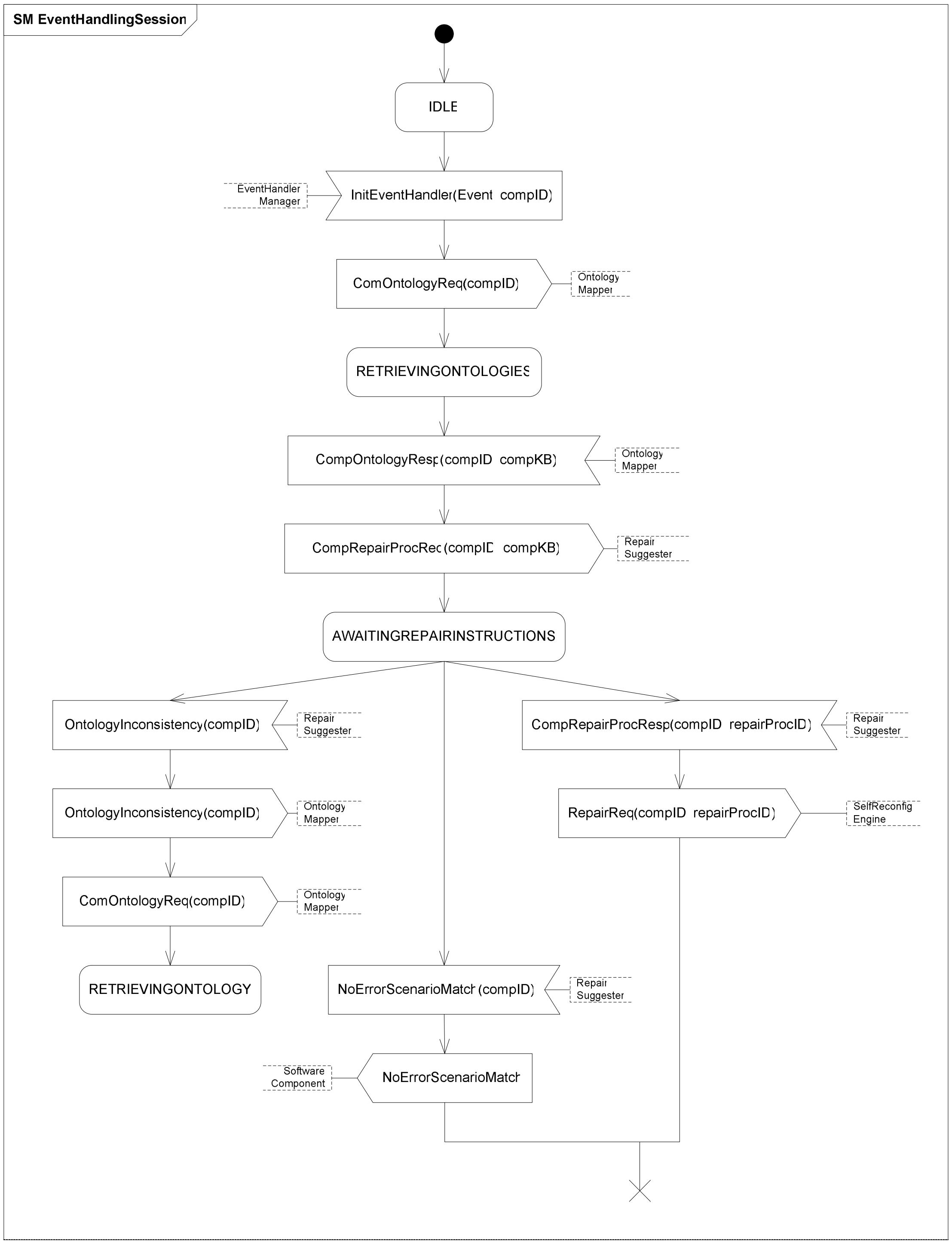


Figure 7-6 – The EventHandlingSession which is created to handle a event

After receiving the initial configuration data from the EHM, this process will send a request to the ontology mapping application asking for the latest OWL file representing the software component. When this is returned to the EHS, a request for a repair procedure for the unit is sent to the repair manager.

This request can either be answered with an indication of ontology inconsistency, the OntologyInconsistency message, a NoErrorScenarioMatch which means that no suitable recovery procedure was found, or the CompRepairProcResp, which contains a recovery procedure which is believed to solve the error. The first of the three, the inconsistency message, contains details about the inconsistency gathered from the OWL reasoner. It is forwarded directly to the ontology mapping application, thus asking it to construct a new ontology to represent the unit. In the case where no known scenarios match the described situation, a message has to be sent back to the software component from which the event originated from. The component may choose to deal with this as it pleases, but when the message has been sent, the EHS will terminate. In the last message, which is what we can call a successful handling of the event, the recovery procedure included is forwarded to the self reconfiguration engine which will execute the repair procedure. After the recovery procedure has been sent, the EHS will end. If any errors occur during the execution of the procedure, it is assumed that the component will fire another event which will be consumed by a new EHS. In Figure 7-7, the message sequence for a successful error handling is shown.

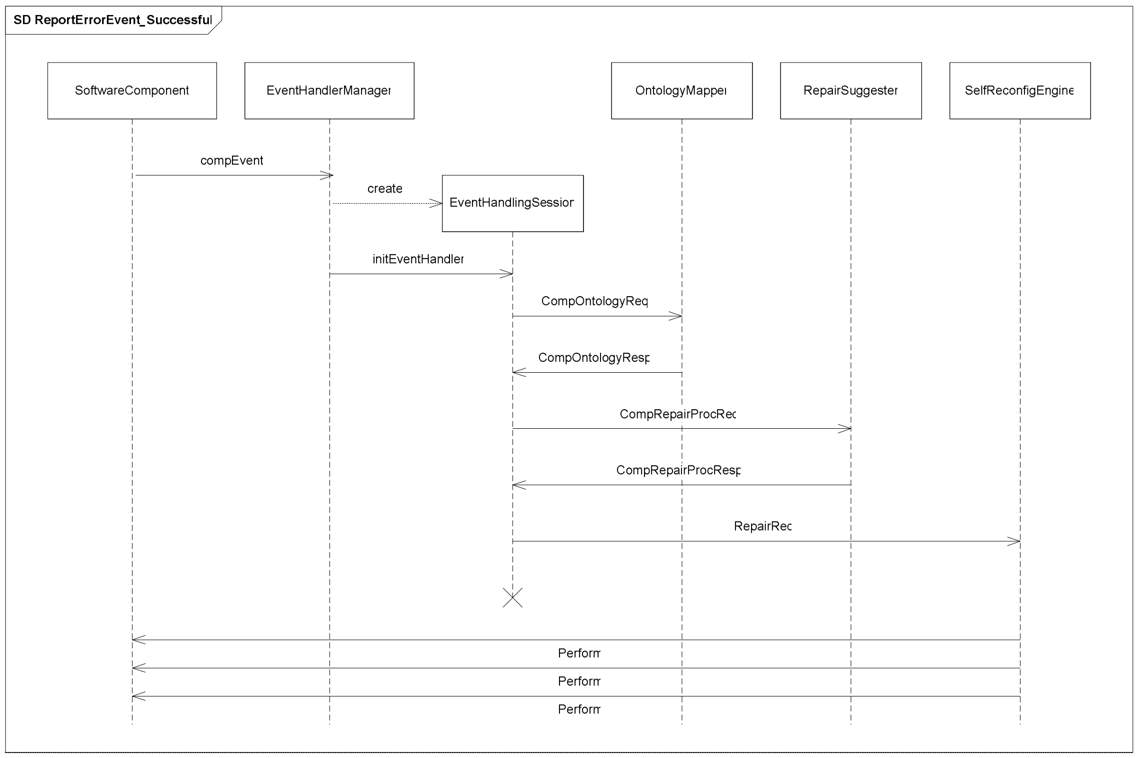


Figure 7-7 – Message Sequence Diagram for a successful handling of a reported error

7.1.8.1.3 OntologyMapper

The OntologyMapper (OM) has the task of communicating with the software component of the unit and then generating an OWL representation of the component. When implementing this part of the system, one generally has to decide whether to require that the mapper is situated on the same computing unit as the software component and thus being able to examine its memory and states, or to construct a process which communicates with the software component, querying it about its current condition.

If the mapper component is required to be physically situated on the same unit, the mapping can be realised through the use of e.g. Java's Reflection API [37]. This API enables java code to examine classes and objects at run time and allows for the mapper software to fully asses the contents of the java virtual machine. This approach has previously been used by Abela and Montebello in [1] as well as by Wang et al in [114] and Ding et al. in [30]. However, if the physical placement of the mapper is to be totally independent of the placement of the unit, which is in accordance with general telematics range of thought, another approach will have to be chosen. One way of accomplishing this is shown in Figure 7-8 and Figure 7-10, where the OM is divided into two separate parts; the OntologyMapperManager (OMM) and the OntologyObjectMapper (OOM). As it shown in Figure 7-8, the former receives registrations from the components and creates and designates a new instance of the OOM to each component.

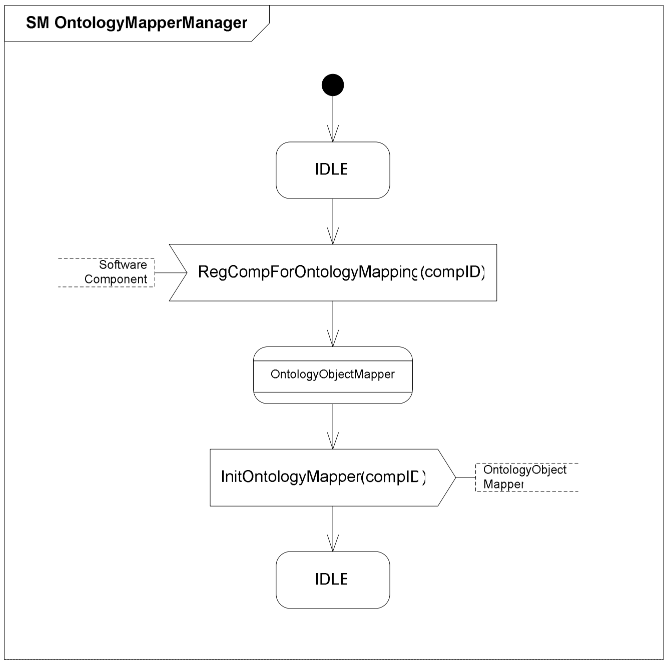


Figure 7-8 – The OntologyMapperManager's state machine

The OOM process will handle the actual construction of the OWL file, and begins with sending a request for the current system info to the component. When a component receives this, it should send its current system info (e.g. serialised classes, etc) to the process. While the OOM waits for this info, any requests for the ontology is stored, and handled later. When the system info arrives, the OOM creates and populate the OWL ontology file. After the initial construction, the component is expected to notify the mapper of changes in its configuration or variables. These transactions are shown in Figure 7-9. The OOM will also send out the OWL ontology on requests, but if an inconsistency is indicated it will attempt to build the ontology up from scratch as indicated in Figure 7-10.

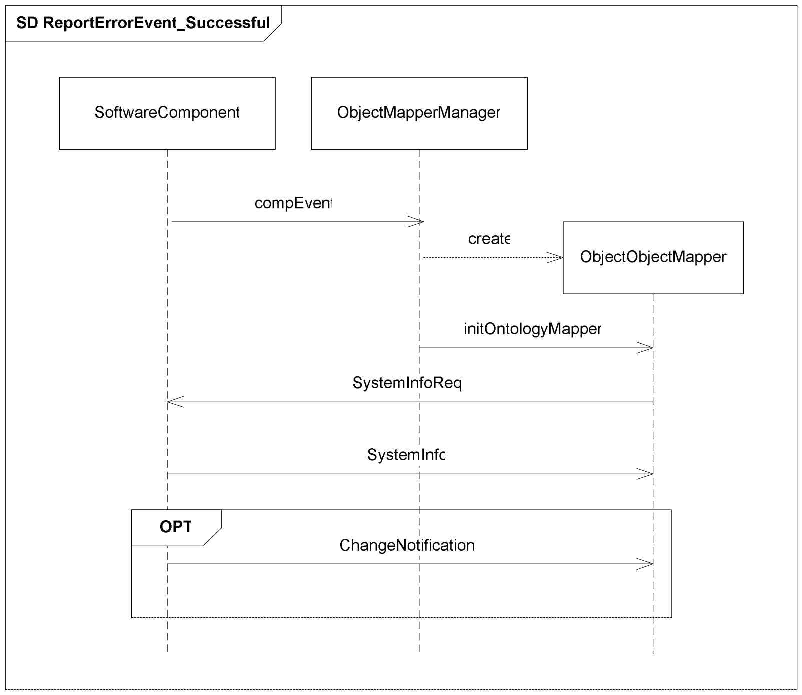


Figure 7-9 – MSC showing how a component registers with the OntologyMapper

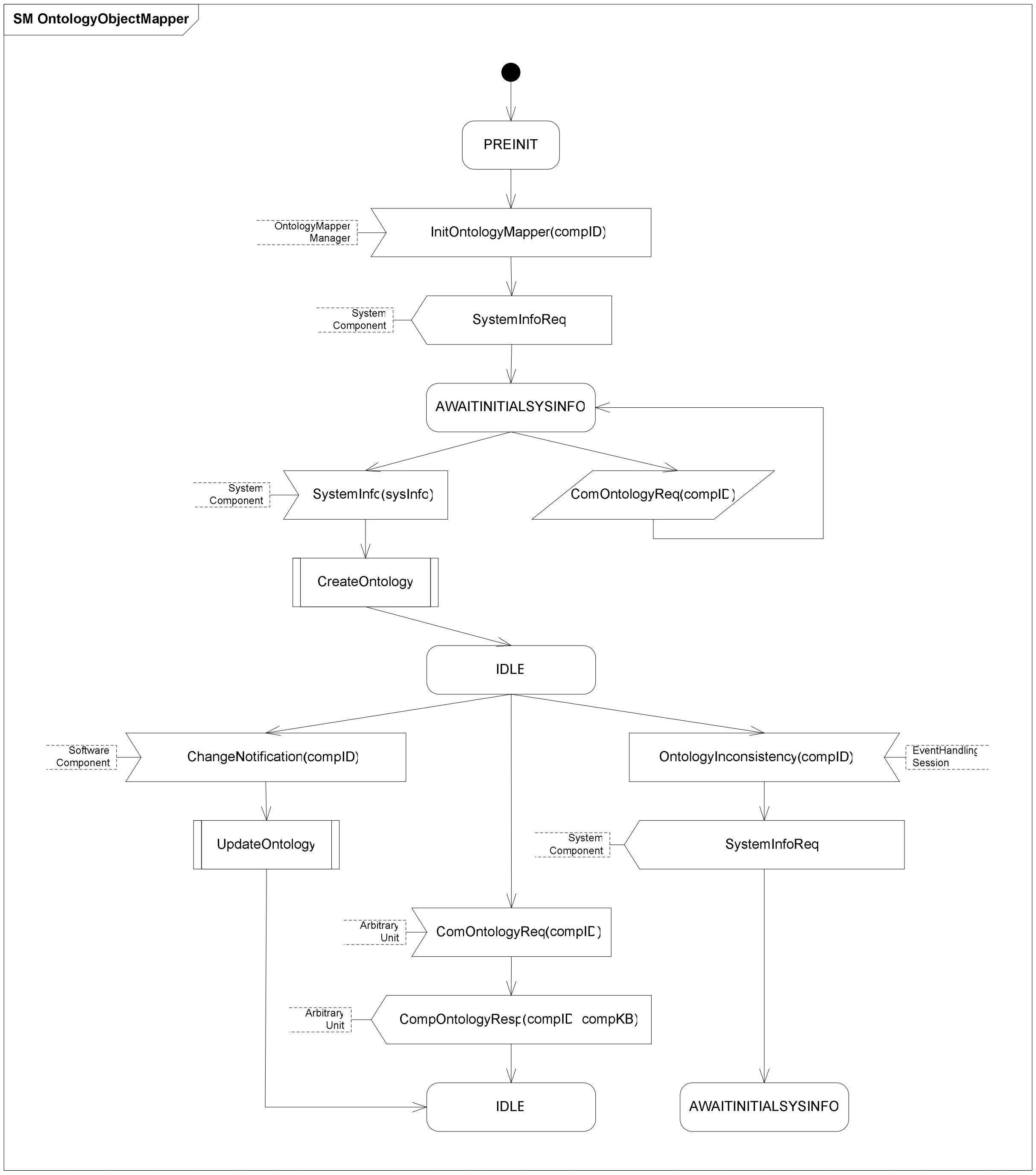


Figure 7-10 – The OntologyObjectMapper which keeps the component ontology up-to-date

7.1.8.1.4 SelfReconfigurationEngine

The last part of the TerminalAgent is the SelfReconfigurationEngine (SRE). This engine has the task of carrying out the defined procedure (or series of actions) that was generated by the self-repair service. These actions are performed through the use of the effector interface on the general software. In Figure 7-11, the state machine diagram of the SRE is shown.

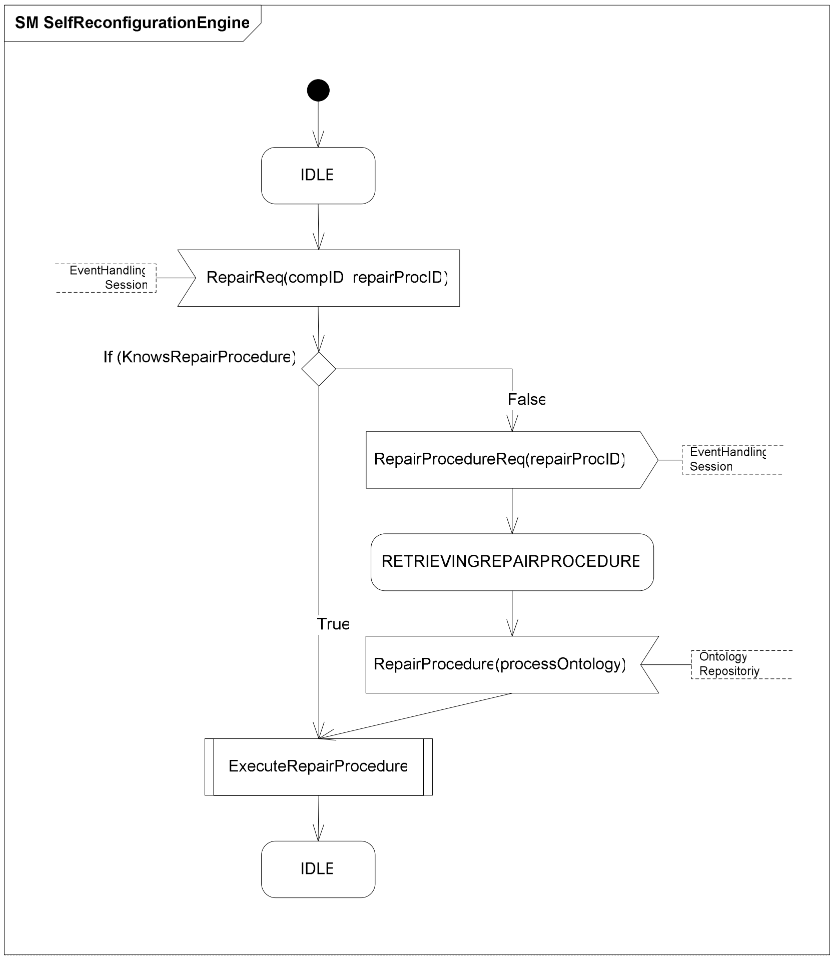

SM SelfReconfigurationEngine
IDLE
EventHandling Session
RepairReq(compID repairProcID)
If (KnowsRepairProcedure)
False
RepairProcedureReq(repairProcID)
EventHandling Session
RETRIEVINGREPAIRPROCEDURE
True
RepairProcedure(processOntology)
Ontology Repositoriy
ExecuteRepairProcedure
IDLE

Figure 7-11 –State machine of the SelfReconfigurationEngine

As we can see, after the engine receives the planned procedure, it must first assess whether it already has the procedure stored in its memory or if it has to be fetched from the Ontology Repository. After the necessary plans are in place, the self-reconfiguration engine will then execute the correct OWL-S encoded procedure.

7.1.8.2 RepairService

The OSCOS Repair service is made up of three inner components as shown Figure 7-12: the repair suggester, the SWRL rule engine and the OWL reasoning engine. This is by far the most computational intensive component in the system and should in some setups with e.g. large percentages of embedded systems, be considered placed on an own server. A discussion on this can be found in sections 9.1 and 9.3.1. The repair service will receive the service request from the event handler and then reply with the procedure which is believed to fix the encountered error after running the necessary tests.

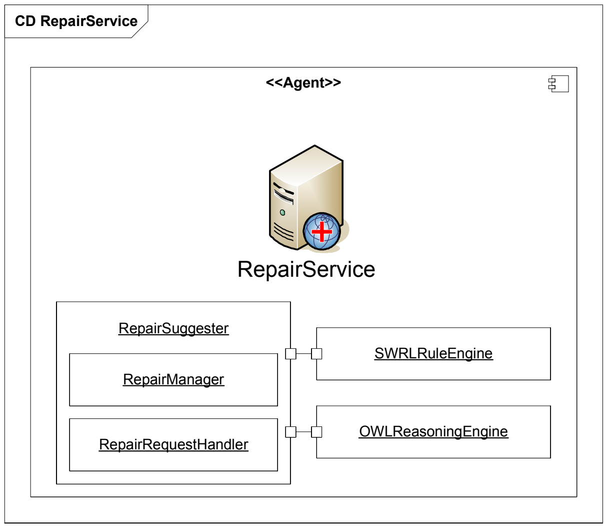


Figure 7-12 – The OSCOS Repair Service

7.1.8.2.1 RepairSuggester

The repair suggester is composed of two basic parts; the RepairManager (RM) and the RepairRequestHandler (RRH). Their behaviour is shown in Figure 7-13 and Figure 7-14 respectively.

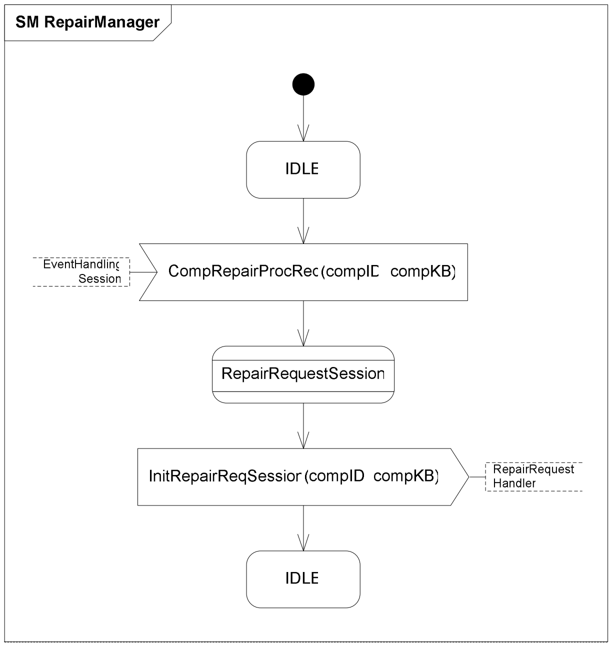


Figure 7-13 – The RepairManager's state machine

First, as we see in Figure 7-13, the RM receives the request from the event handler and creates a new instance of the RRH which will handle the rest of the request. This allows also for the manager to easier distribute the load on different computational units. The RRH state machine can bee seen in Figure 7-14 and Figure 7-16. A sequence chart showing a successful error handling is depicted in Figure 7-15.

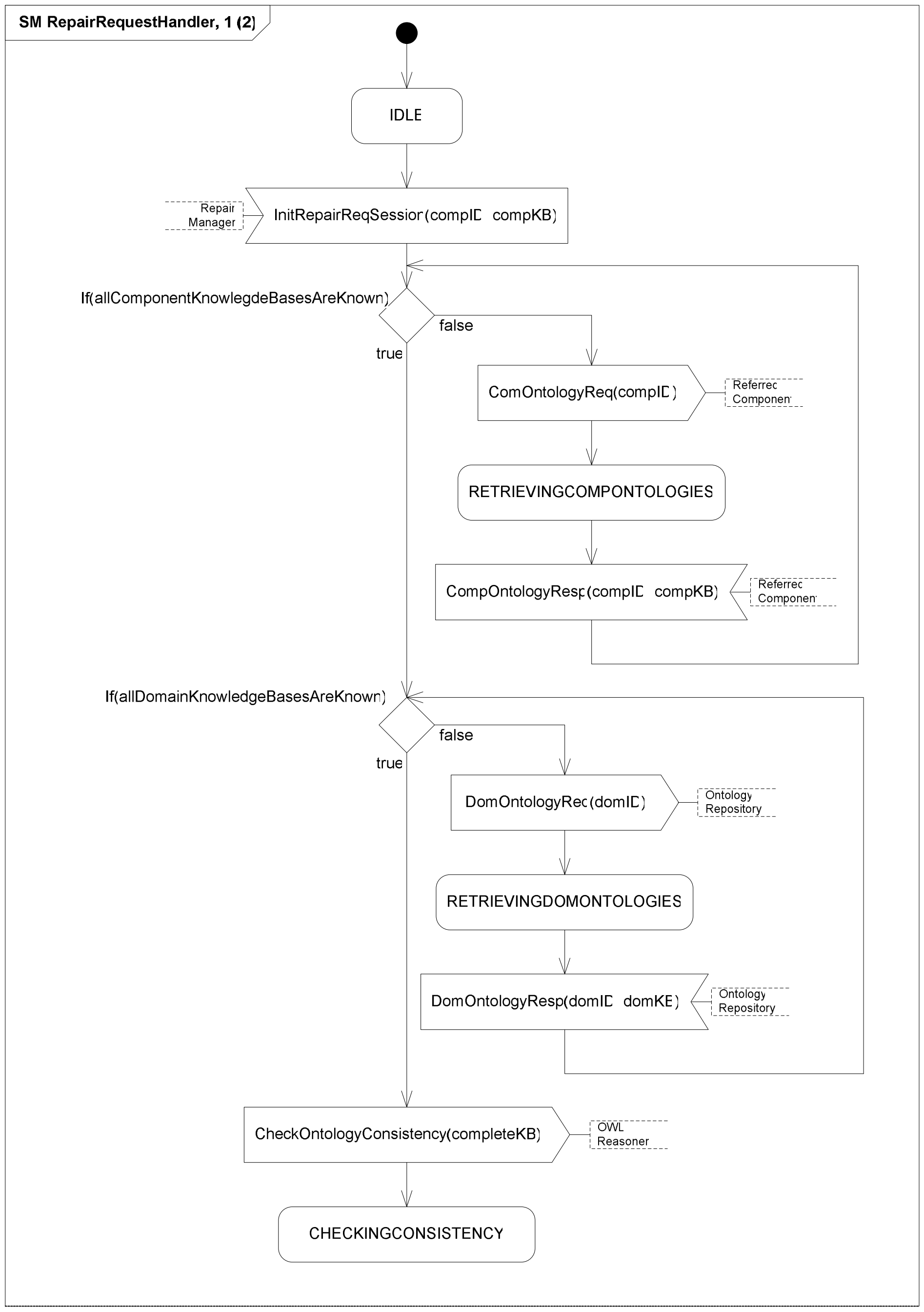

SM RepairRequestHandler, 1 (2)
IDLE
Repair Manager
InitRepairReqSession(compID, compKB)
If(allComponentKnowlegdeBasesAreKnown)
false
true
ComOntologyReq(compID)
Referred Component
RETRIEVINGCOMPONTOLOGIES
CompOntologyResp(compID, compKB)
Referred Component
If(allDomainKnowledgeBasesAreKnown)
false
true
DomOntologyReq(domID)
Ontology Repository
RETRIEVINGDOMONTOLOGIES
DomOntologyResp(domID, domKB)
Ontology Repository
CheckOntologyConsistency(completeKB)
OWL Reasoner
CHECKINGCONSISTENCY

Figure 7-14 – Part one of the behavioural state machine of the RepairRequestHandler

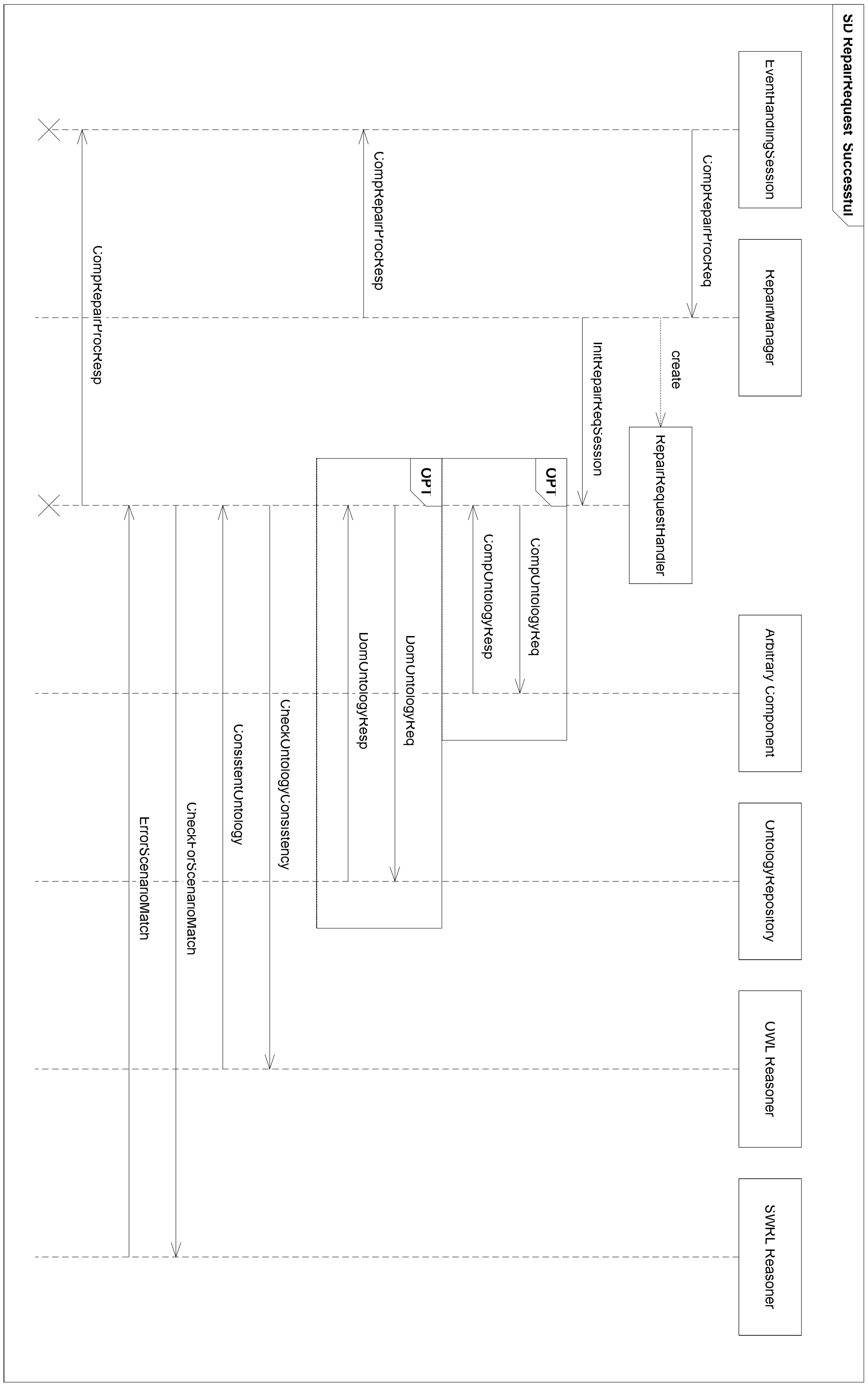


Figure 7-15 – Sequence diagram showing a successful request for a repair procedure

After the RRH is created and initialised, the first thing it will do is to check whether it has all the information which are referred to in the attached unit OWL file. This is shown in Figure 7-14 and includes both component instances and domain ontologies. If there are any files that are lacking, these are requested through the use of an Ontology Repository (e.g. if it is an ontology definition) or through a direct request to the unit believed to be holding the ontology (e.g. if the referred file is an instance file of another connected unit). After all the necessary OWL files are in place, the RRH will try to resolve the error.

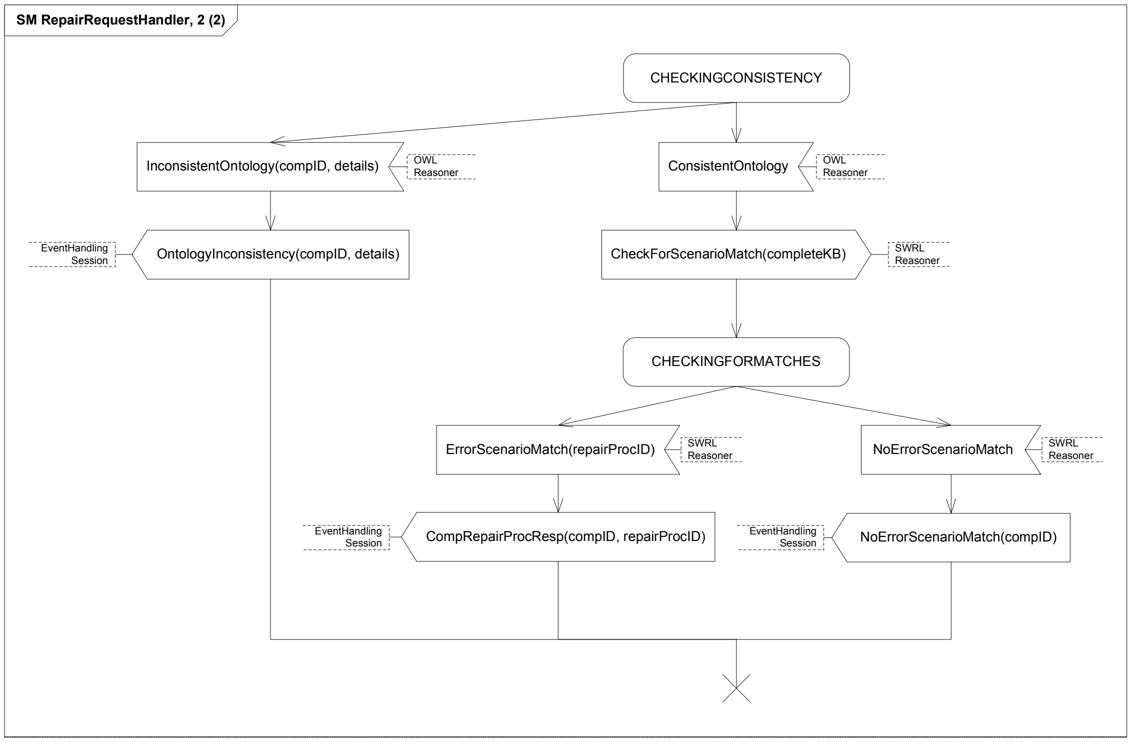

SM RepairRequestHandler, 2 (2)
CHECKINGCONSISTENCY
InconsistentOntology(compID, details)
OWL Reasoner
ConsistentOntology
OWL Reasoner
EventHandling Session
OntologyInconsistency(compID, details)
CheckForScenarioMatch(completeKB)
SWRL Reasoner
CHECKINGFORMATCHES
ErrorScenarioMatch(repairProcID)
SWRL Reasoner
NoErrorScenarioMatch
SWRL Reasoner
EventHandling Session
CompRepairProcResp(compID, repairProcID)
EventHandling Session
NoErrorScenarioMatch(compID)

Figure 7-16 - Part two of the behavioural state machine of the RepairRequestHandler

The reasoning process of the repair service is, as shown in Figure 7-16, composed of two phases; ontology consistency checking followed by checking for rule matches. The ontology consistency check is realised by sending the ontology to the OWL reasoning engine. This is to ensure that the unit has a consistent ontology which concurs with all the specified restrictions. Such a consistency check would also discover and make the unit able to possibly repair configuration errors like e.g. failed cardinality restrictions, data types etc. Repairing and sorting out inconsistencies in an ontology can however be a highly complex process which is not further assessed in this thesis but is rather left for future work. Readers interested in further literature on this topic are referred to [97] by Schlobach et al.

If the ontology is classified as inconsistent, this is reported back to the event handler and the RRH will end, requiring that a new RepairRequest has to be sent when the inconsistency has been worked out. If the consistency of the ontology is confirmed, all the OWL ontology files are sent to the SWRL Rule Engine where they will be checked for matches to the pre-defined scenarios. When the result from this matching process is received, it is forwarded to the event handler and the RRH will then end.

7.1.8.3 OntologyRepository

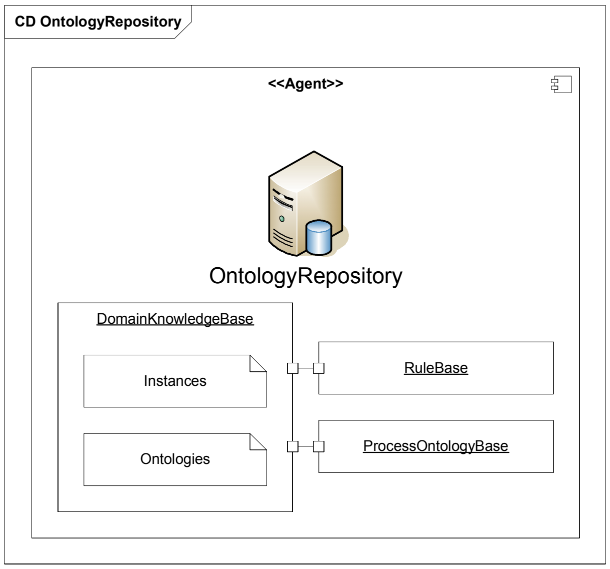


Figure 7-17 – The contents of the Ontology Repository

The Ontology Repository contains, as shown in Figure 7-17, three main components: the Domain Knowledge Base, the Rule Base and the Process Ontology Base. These are all pretty simple database components which work in a general insert and request pattern as shown in Figure 7-18.

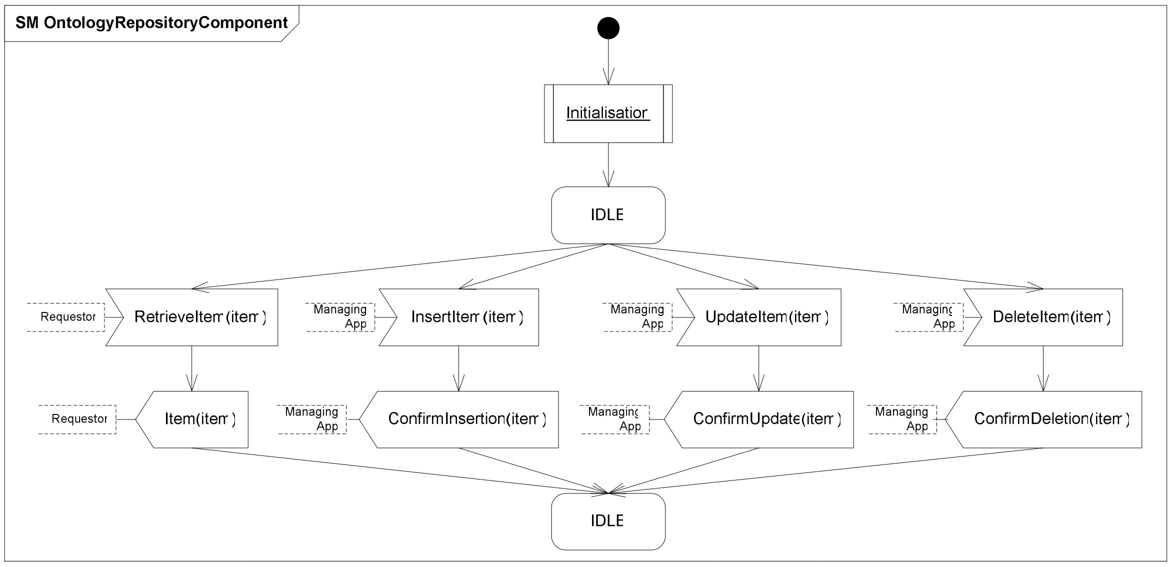


Figure 7-18 – The general state machine for the components of the OntologyRepository

7.1.8.3.1 DomainKnowledgeBase

The DomainKnowledgeBase (DKB) is itself composed of instances, or individuals, and ontologies as explained in section 4.3. The different instances are typically units which have reported their ontology to the knowledge base. Although it would be nice to always have an up-to-date knowledge base over a given sub-set of the available units, this would not be advisable, as it would probably scale rather poorly when applied on very large domains. It would also create a lot of network traffic, as the instances would have to be reported in on every change in the unit. It is therefore desirable that only units that have ontologies that are more or less in-accessible in some way, or rather static, are cached in the central knowledge base.

7.1.8.3.2 RuleBase

The RuleBase (RB) is simply a database containing SWRL rules which are used to classify error-events and assigning these to given error-scenarios. These rules are explained in section 7.1.6. The RB is the only component that differs from the general behaviour in Figure 7-18. It adds a new transition "GetGroup" to allow for a SWRL engine to fetch the group of rules that fits with the knowledge base which it is about to check. This extension is shown in Figure 7-19.

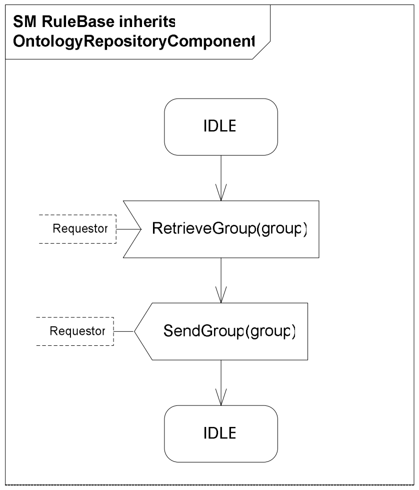


Figure 7-19 – The extension added to construct the RuleBase component

7.1.8.3.3 ProcessOntologyBase

The ProcessOntologyBase (POB) contains procedures encoded using OWL-S used for resolving error-scenarios. These procedures are described in more detail in 7.1.7, but are in essence OWL-S composite processes containing other, possibly both atomic and composite, processes alongside conditionals (e.g. if-then-else statements) which combined is thought to resolve a given error-scenario.

8 Experiments

In order to be able to fully assess whether the approach described in chapter 7 is feasible, two important experiments will have to be conducted; SWRL matching and OWL-S execution. These experiments will be described in this chapter alongside their results. The modelling of necessary ontology parts, such as the domain ontology and the process ontology of the mentioned case, is also presented.

8.1 Domain Ontology Design

The domain ontology in the OSCOS system is created to realise a foundation that the different units can use to create an ontological overview over themselves and others. The development of this ontology has been based on the ontological engineering methodology presented in section 6.3. Because one of its primary uses is to serve as a formal information model when setting up the rules used to classify different error scenarios, its design has been focused on modelling the domain in a way that facilitates easy rule development. The ontology does not completely represent the H.323 domain as it lacks complete representations of configuration files etc, but it serves well as an experimental basis for the OSCOS feasibility tests conducted in the end of this chapter (section 8.3).

This section will give a brief overview over the constructed domain ontology. It is based on the H.323 Recommendation from ITU-T ([64]) and other related documents ([15], [63], [106] & [107]). The entire ontology was modelled using the Protégé modelling environment (see appendix A.1) and is attached to this thesis both in an easy-to-browse form using OWLDoc as well a normal OWL-encoded version.

8.1.1 Top Level Concepts

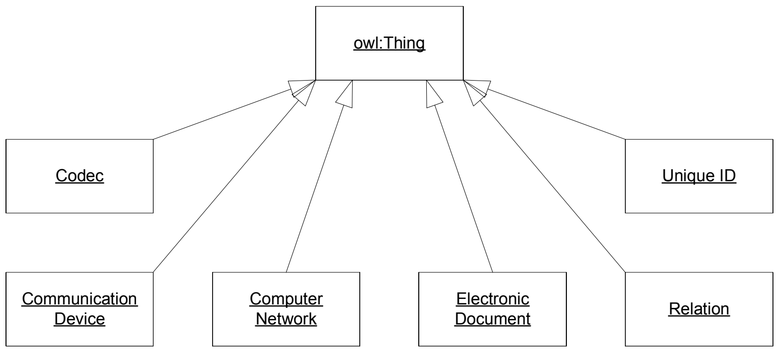


Figure 8-1 – The top-level of the engineered ontology

On the top level of the domain ontology, seven concepts were identified: Codec, Communication_Device, Computer_Network, Document, Relation and finally the Unique_ID concept. In accordance with the ontological engineering methodology presented in section 6.3, several concepts have been integrated from other ontologies. Three of the top concepts are from one of the best known upper ontologies, OpenCyc [86]. This upper-ontology project has a vision of creating an ontology of the entire world. The concepts adopted from OpenCyc are: Communication_Device, Computer_Network and Unique_ID. In addition to this, one concept, the Electronic_Document, has been integrated from the SATURN project [96], which is an effort to apply Semantic Web technologies to aid information sharing and usage in the Intelligence Community [69].

Each of the top level concepts contain sub-concepts that are presented alongside a description of the individual concepts in the following. Readers interested in detailed specifications of restrictions and properties are referred to appendix A.1.4, where it is explained how one can open and browse the entire ontology.

8.1.1.1 The Codec Concept

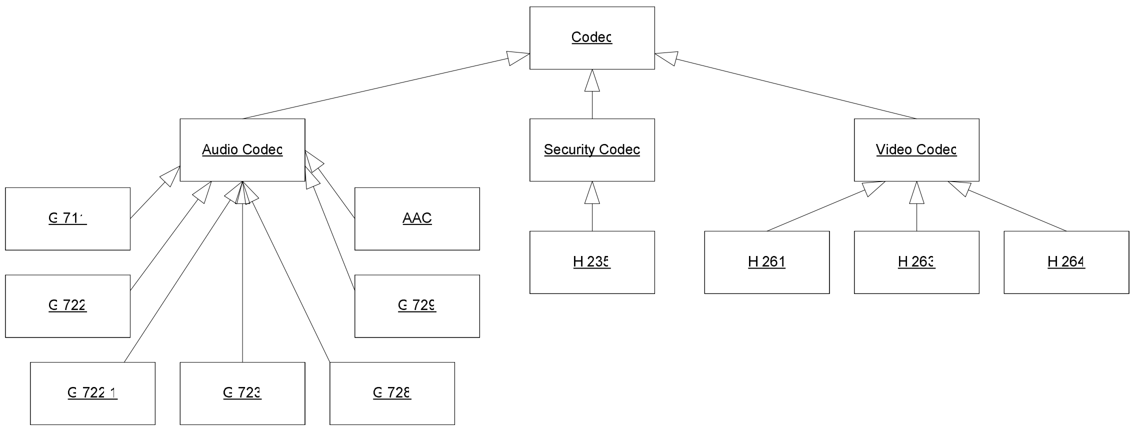


Figure 8-2 – The Codec concept representing different computer codecs

The Codec concept has been adapted from OpenCyc's encoder and decoder classes. This is a concept which represents an algorithm or code capable of performing some transformation on a data stream or signal. Codecs can both put the stream into an encoded form (e.g. compression or encryption) and later decode the stream back to its original form. This concept has been further divided into three sub-concepts: the Audio_Codec, the Security_Codec and the Video_Codec. These represent codecs for audio, security and video transformations respectively, and have several sub-concepts representing different standard codecs used in the H.323 standard.

8.1.1.2 The Communication_Device Concept

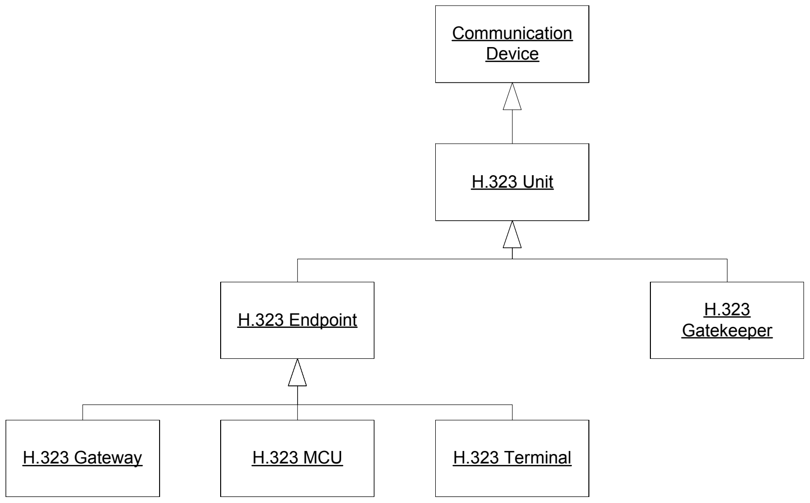


Figure 8-3 – The Communication_Device concept is at the heart of the proposed ontology

The Communication_Device is also adopted from the OpenCyc upper ontology, but has been extended to include the H.323_Unit sub-concept. This concept represents all the different types of H.323 units. The H.323_Unit has two sub-concepts of its own: the H.323_Endpoint, and the H.323_Gatekeeper. Since the H.323_Gateway, the H.323_MCU and the H.323_Terminal all can be endpoints in a conferencing call, they were placed as sub-concepts of the H.323_Endpoint. The different units represent their corresponding H.323 unit entity respectively.

8.1.1.3 The Computer_Network Concept

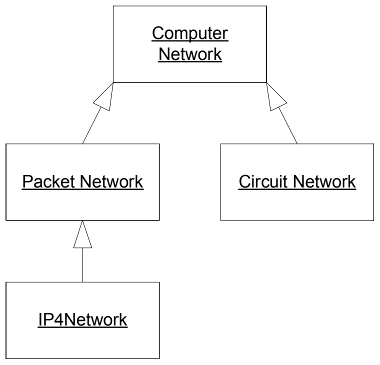


Figure 8-4 – The Computer Network concept with its sub-concepts

The Computer_Network was adopted from the upper ontology of OpenCyc and is a characterised as a “network used to link computational systems together to allow them to transfer information between each other” [86]. This has been further specialised to include Packet_Network and and Circuit_Network, which are two common network technologies. Packet_Network has also been extended with the IP4Network concept.

8.1.1.4 The Electronic_Document Concept

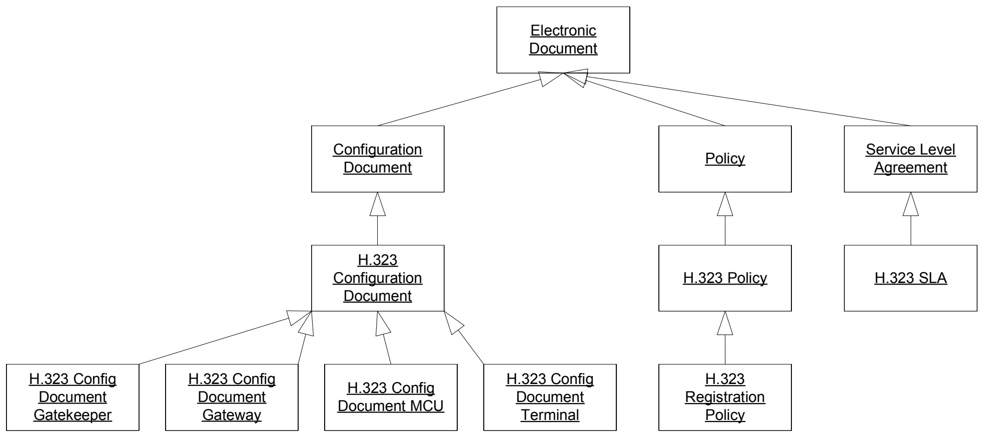

Electronic Document
Configuration Document
Policy
Service Level Agreement
H.323 Configuration Document
H.323 Policy
H.323 SLA
H.323 Config Document Gatekeeper
H.323 Config Document Gateway
H.323 Config Document MCU
H.323 Config Document Terminal
H.323 Registration Policy

Figure 8-5 – The Document concept

The Electronic_Document is a concept integrated from the SATURN project and usually represents an electronic document used to convey information. This was further extended further by adding three sub-concepts: Configuration_Document, Policy and Service_Level_Agreement.

The Configuration_Document represents documents that are used for configuring units and software. E.g. defining variables, system class path and similar. It has only one sub-concept, the H.323_Configuration_Document, which in turn has four sub-concepts of its own: H.323_Config_Document_Gatekeeper, H.323_Config_Document_Gateway, H.323_Config_Document_MCU and H.323_Config_Document_Terminal. Each concept specifies the configuration of a particular H.323 unit concept.

The Policy concept represents control documents used to ensure that systems confine to a set of policy rules. It has the H.323_Policy as a sub-concept, which in turn has another sub-concept; the H.323_Registration_Policy. The latter is a concept used to ensure that H.323 units registering with a H.323 gatekeeper follows certain rules, such as providing authentication or other application specific data.

The final sub-concept of the Electronic_Document is the Service_Level_Agreement. This represents a formal agreement between two parties, and defines the basis for the delivery of a service. Its only sub-concept, the H.323_SLA, represents an agreement between two H.323_Gatekeepers that allows for their users to communicate with each other.

8.1.1.5 The Relation Concept

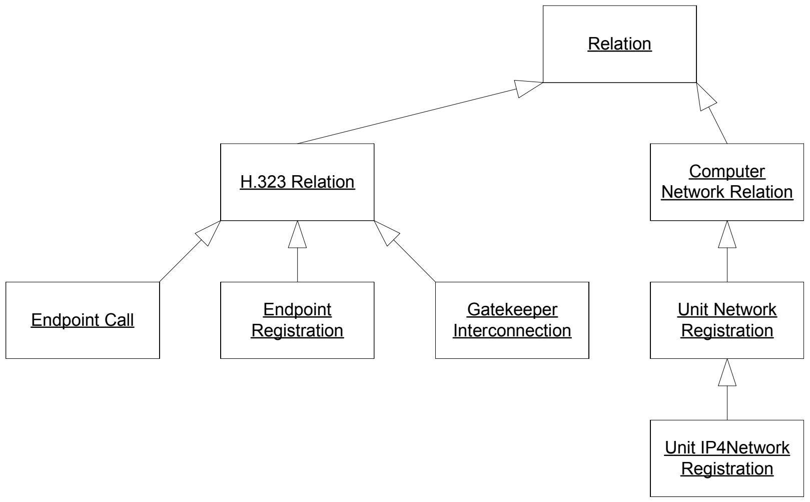

Relation
H.323 Relation
Computer Network Relation
Endpoint Call
Endpoint Registration
Gatekeeper Interconnection
Unit Network Registration
Unit IP4Network Registration

Figure 8-6 – The Relation concept which represents n-ary relations in the ontology

The Relation concept represents n-ary relations as explained in 4.3.2. It has two sub-concepts: the H.323_Relation and the Computer_Network_Relation, representing n-ary relations in H.323 and networks respectively.

Three H.323_Relations have been identified. The Endpoint_Call represents a call between two H.323_Endpoints, The Endpoint_Registration the registration of an H.323_Unit with a gatekeeper, whilst the Gatekeeper_Interconnection represents the interconnection of two gatekeepers.

Only one Network_Relation was identified as relevant; the Unit_Network_Registration which is the registration of a Communication_Device to a Computer_Network. This concept was also appended with the Unit_IP4Network_Registration concept, representing the unit's registration with an IP4Network. It is here worth noting that an H.323_Unit requires the use of such a network when communicating with others.

8.1.1.6 The Unique_ID Concept

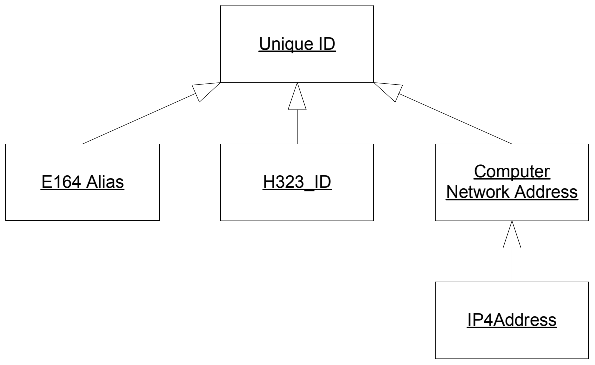


Figure 8-7 – The Unique_ID concept with its three sub-concepts

The last top-level concept of the OSCOS ontology is the Unique_ID. The concept is meant to represent "an object that can be used as a unique identification of a distinct entity" [86] and stems from the OpenCyc upper ontology. The Unique_ID concept was extended to include E164_Alias, H323_ID and Computer_Network_Address as shown in Figure 8-7.

The E164_Alias is the E.164 telephone number which has been assigned to identify a given H.323_Terminal. This can be a global or a local number. The H.323_ID is a concept similar to email-ids, and is also assigned to identify a given H.323_Terminal. The IP4Address was originally a direct sub-concept of OpenCyc's Unique_ID, but it was decided to reallocate this as a sub-concept of the newly introduced Computer_Network_Address instead as this better fitted the planned architecture. The IP4Address is made up of a string that represents a unique address of a network which conforms to InternetProtocol v4.

8.1.2 Important N-ary Relations

As explained in section 4.3.3, most ontology representation languages consider a property as a binary relation. It has been suggested by, amongst others, the WC3 to work around this by introducing a new concept to represent n-ary relations [85]. This section will present and describe some of the most important n-ary relations in the designed H.323 ontology.

8.1.2.1 Unit_Network_Registration

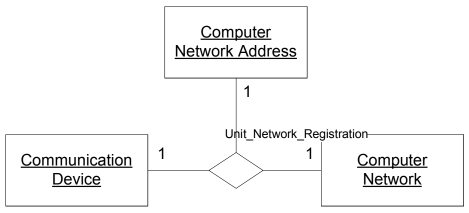


Figure 8-8 – The Unit_Network_Registration representing a device's registration to a network

Whenever a unit registers to a network, it needs an address uniquely identifying the unit. This address can be set either automatically or statically depending on both the units own capabilities as well as the network's capabilities. Shown in Figure 8-8 is the n-ary relation that represents the registration. As we can see, all parts of the relation is restricted to the cardinality of one, meaning that only one of each component may be used in a valid registration.

8.1.2.2 Endpoint_Call

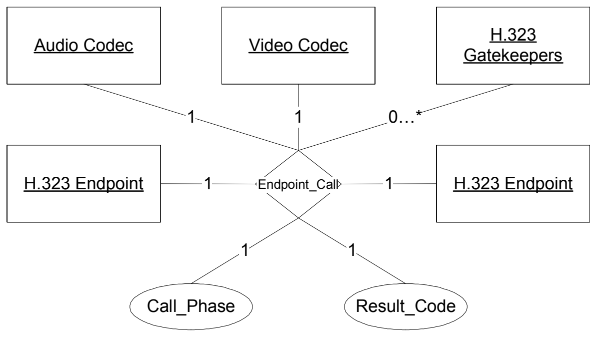


Figure 8-9 – The Endpoint_Call which represents a call between two H.323_Endpoints

The Endpoint_Call concept is an n-ary relation which describes the call between two H.323 Endpoints, and is shown in Figure 8-9. When setting up calls in an H.323 environment different schemes may be used. Messages may be exchanged either end-to-end between the calling party and the called party or through one or more gatekeepers, the most common being through one or two gatekeepers. Each call goes through several call phases including “setup”, “call”, “termination” and “terminated”, and can either be “ok” or “failed”. The primary audio and video codecs which are used in the call are also indicated.

8.1.2.3 Endpoint_Registration

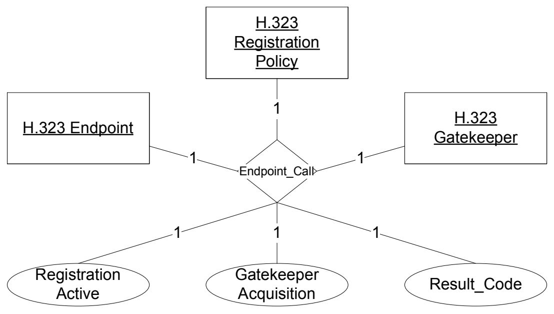


Figure 8-10 – Endpoint_Registration, representing a H.323 unit’s registration to a gatekeeper

The last n-ary relation presented in this chapter is the Endpoint_Registration which is shown in Figure 8-10. When an endpoint registers to a gatekeeper, the relation keeps track of whether the registration is active, how the unit got hold of the gatekeeper address, as well as the result code of the registration. At last, a registration policy is attached to the registration. This policy describes the properties which the endpoint must satisfy to be able to register with the gatekeeper. This is a common source to errors, and is therefore very useful in sorting out simple configuration problems.

8.2 Process Ontology Design

As mentioned in section 7.1.7, in order for OWL-S error-recovering procedures to be specified, the OSCOS architecture needs to have a complete OWL-S encoded view of both the atomic and composite services provided by the unit. In Table 8-1, modelled atomic and composite services for the H.323 terminal are briefly presented. This design has been based on the H.323 Recommendation from ITU-T ([64]) and other related documents ([15], [63], [106] & [107]). The process ontology is also visualised in Figure 8-11. Readers unfamiliar with the message name format are referred to appendix A.3.1.

Service name	**Description**
AutoGatekeeperDiscovery	The AutoGatekeeperDiscovery service is used by the H.323 unit to automatically find a gatekeeper which it can connect to. This is done by sending out multicast GRQ that is replied with a GCF / GRJ by Gatekeepers. A list over the GCF, i.e. the potential suitable gatekeepers is returned to the service user.
BandwidthRequest	After the initial call setup, this is a service which the H.323 unit can use to indicate that it wishes to use more or less bandwidth than previously indicated. Its result is returned to the user.
CallAdmissionRequest	When a call is being made, the CallAdmissionRequest service sends an ARQ to its gatekeeper, asking for permission to establish a call with another H.323 unit. The result is returned to the user.
DisengageCall	When the H.323 unit need to disengage a call, it sends a DRQ packet to indicate that it wishes to end its call.
ConfirmRemoteDisengagement	This is a service which confirms the remote disengagement of a call.
GenerateNewAlias	The GenerateNewAlias service will, as indicated, generate a new alias based on its input variables. This could be inserting a necessary prefix, or generating a totally new (and unused) alias. The generated alias is then returned to the user.
LocationRequestForUser	This service is used to look for users. The LRQ packet generated by this service may be sent directly to a known gatekeeper, or sent by multicast into the network. The result is returned to the user.

Service name	**Description**
NotifyUser	This is a simple service which sends textual descriptions of a problem to the user. This can be useful in situations where the procedure encounters situations which can not be easily fixed like e.g. a unit tries to reach another unit which does not even exist on the network.
ReceivedCallAdmissionRequest	Whenever a H.323 terminal which is connected to a gatekeeper receives a call, it needs to check with the gatekeeper whether it may accept the call or not. This is done by sending an ARQ to its own gatekeeper. The result is returned to the user.
ReceivedCallSetupConfirm	If the received call is accepted by the gatekeeper and the user, this service confirms the call setup and initiates the call with the calling unit.
ReceivedCallSetupDeny	If, however, the received call is not accepted by either its gatekeeper or its user, the call setup is denied and an indication of this is sent to the calling unit.
RetrieveAuthenticationInfo	Service used to retrieve authentication info from the user of the unit for further use in the security services of an H.323 unit.
SetNewAlias	This rather simple service which is just used to set a new alias for the endpoint.
SetupCall	When a call has been approved by the gatekeeper after sending an admission request, this service will then be used to establish a connection to the user which the unit wants to call.
StaticGatekeeperDiscovery	This service is used by the H.323 unit when a static gatekeeper address has been used to configure the unit. The service will attempt to send a GRQ packet to the gatekeeper and is replied with either a GCF or a GRJ. The result is returned to the service's user.
UnregisterWithGatekeeperRequest	As the name indicates, this service is used to send an unregister request to its gatekeeper.

Service name	**Description**
Dial	This composite service is visualised in Figure 8-11 and is the basic call service provided by the unit. It is made up of two atomic services; CallAdmissionRequest and SetupCall.
GatekeeperRegistration	The GatekeeperRegistration is also a composite service and utilises, as shown in Figure 8-11, three atomic services; AutoGatekeeperDiscovery, RequestGatekeeperRegistration and StaticGatekeeperDiscovery.
UnregisterWithGatekeeper	When a unit wishes to unregister with a gatekeeper, it uses this composite service which is made up of two atomic services; the DisengageCall and the UnregisterWithGatekeeperRequest.
ReceiveCall	The ReceiveCall composite service is used when a unit receives a call from another unit. Its composition, as shown in Figure 8-11, is made up of ReceiveCallAdmissionRequest, ReceiveCallSetupConfirm and ReceiveCallSetupDeny.

Table 8-1 – Identified atomic and composite services of a basic H.323 Terminal

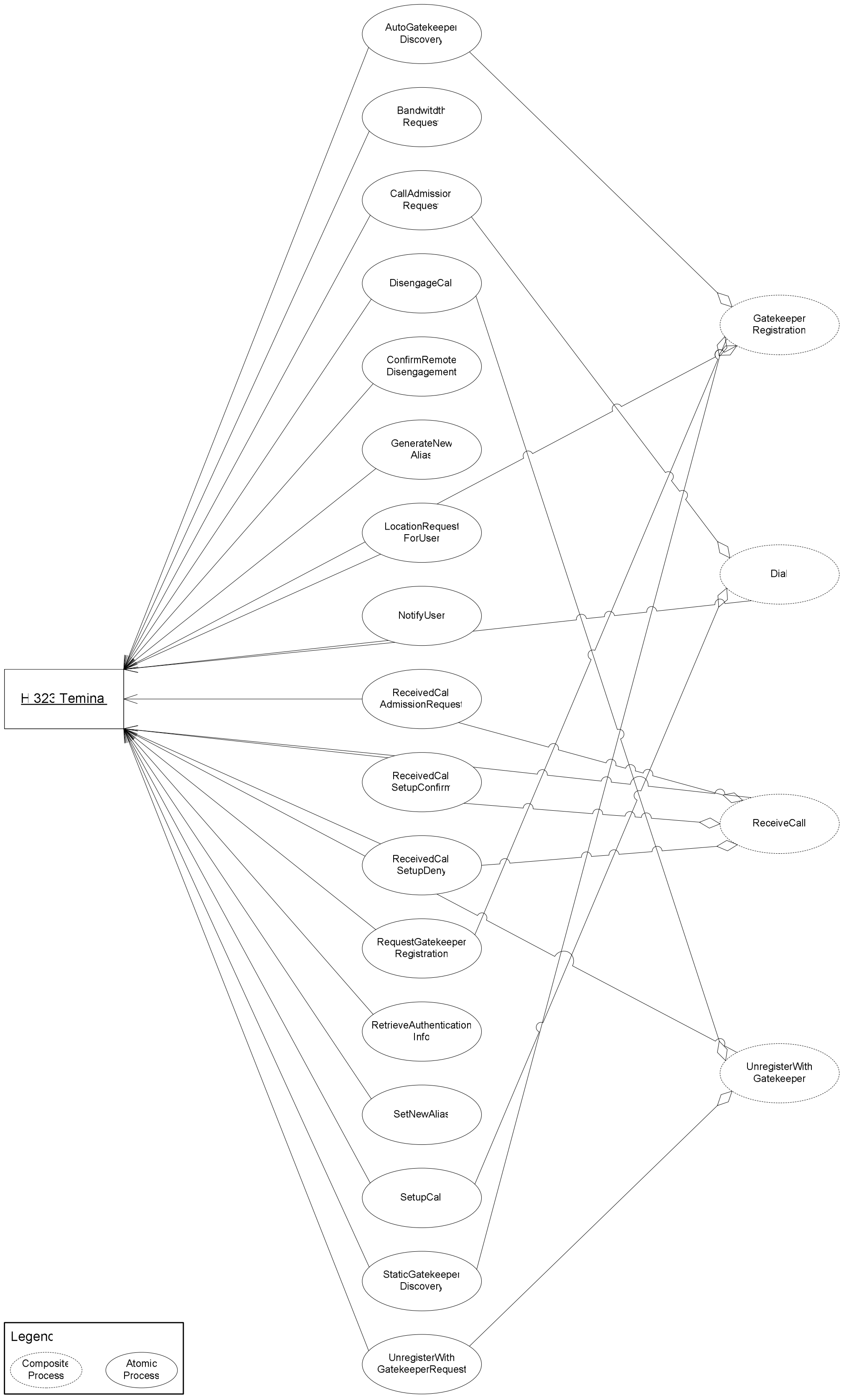


Figure 8-11 – Process ontology for the H.323 Terminal with its atomic and composite services

8.3 Experiment Descriptions

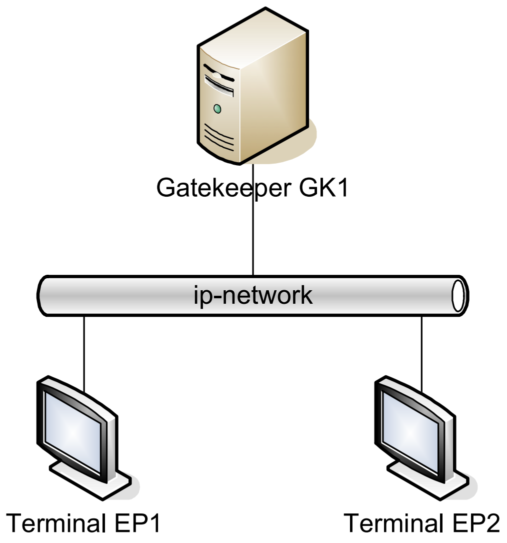


Figure 8-12 – Topology used for test cases

All test cases and experiments are based on a simple network as illustrated in Figure 8-12. This network was modelled by inserting instances into the domain ontology presented in section 8.1. These OWL files can be found in the file attachments of this thesis. The actual experiments were conducted using this modelled base together with SWRL reasoning engines and the OWL-S API. For each of the test cases, the instances' variables change, introducing new error scenarios. The SWRL rules used are presented for each case, along with the suggested OWL-S process which is thought to resolve the described problem. Both the SWRL rules and the OWL-S procedures were developed from informal descriptions in natural language from TANDBERG R&D domain experts.

The experiments were conducted as follows:

1) Gather all test case rules in a combined rule base and test whether the correct complete rule is triggered

2) Test whether it is possible to execute the OWL-S encoded repair procedure

When writing SWRL rules variables are used to bind together the different rules. If sufficient care is not shown, the set of variables might grow so large and complex that is becomes hard to get a complete overview. In Table 8-2 the variables used in the SWRL experiment rules are shown.

Variables	Description
`?ep1`	Endpoint 1
`?ep2`	Endpoint 2
`?gk1`	Gatekeeper 1
`?ep1configdoc`	The configuration document of Endpoint 1
`?gk1configdoc`	The configuration document of Gatekeeper 1
`?ep1callsep2`	The call relation between Endpoint 1 and Endpoint 2
`?ep1reggk1`	The registration relation between Endpoint 1 and Gatekeeper 1
`?ep2reggk1`	The registration relation between Endpoint 2 and Gatekeeper 1
`?gk1regpolicy`	The registration policy of Gatekeeper 1
`?gk1regpolicyAllowedAlias`	The regular expression which the aliases of endpoints connecting to Gatekeeper 1 need to satisfy. This is specified in the registration policy of Gatekeeper 1
`?ep1Alias`	The E164 alias of Endpoint 1

Table 8-2 – SWRL variables

8.3.1 Test Case 1

In the first test case the gatekeeper has set an option in its configuration file requiring every endpoint connecting to it to have to have an E164 alias which complies with a pre-specified regular expression. An example of such an expression could be *`6214*`*, which matches any expression beginning with 6214 and continuing with any arbitrary string of numbers or characters. Endpoint 1 has in this test case an alias which does not comply with the gatekeeper's and will therefore not be able to make any calls. When it then tries to call endpoint 2, the call fails. The SWRL rule used to identify the situation is shown in Table 8-3.

Textual description	SWRL Encoded Rule
When EP1 tries to initiate a call with EP2 using GK1, call setup fails.	`h323EndpointCallCaller (?ep1callsep2,?ep1)^` `h323EndpointCallCallee (?ep1callsep2,?ep2)^` `h323EndpointCallResultCode (?ep1callsep2,"failed")`
EP1 and EP2 share the H.323 Gatekeeper.	`h323EndPointRegistrationRegisteredEndpoint (?ep1reggk1,?ep1)^` `h323EndPointRegistrationRegisteredEndpoint (?ep2reggk1,?ep2)^` `H323EndpointRegistrationRegisteredWithGatekeeper (?ep1reggk1,?gk1)^` `H323EndpointRegistrationRegisteredWithGatekeeper (?ep2reggk1,?gk1)`
EP1 has an alias which does not conform to the Gatekeeper's registration policy.	`h323GatekeeperRegistrationPolicy (?gk1,?gk1reppolicy)^` `h323RegistrationPolicyAllowedAliases (?gk1reppolicy,?gk1reppolicyAllowedAlias)^` `h323TerminalE164Alias (?ep1,?ep1Alias)^` `¬swrlb:matches (?ep1Alias,?gk1reppolicyAllowedAlias)`

Table 8-3 – SWRL rules used for test case 1

In order to correct the situation without human intervention the unit will have to generate a new alias which complies with the gatekeeper's alias expression. The expression is therefore given as an input to the GenerateAlias service together with any previous alias. The service will then generate an alias which both complies with the expression, but that also is available within the gatekeeper's zone. This is then set as the new alias at the unit. The entire repair procedure is shown in Figure 8-13.

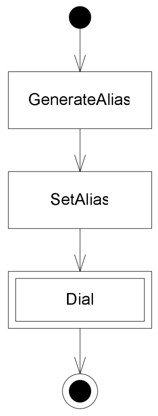


Figure 8-13 – Repair process for the first case scenario specified in OWL-S

8.3.2 Test Case 2

The second test case is a bit simpler than the first one. In this scenario, endpoint 1 tries to place a call to endpoint 2, but endpoint 1 is not registered with any gatekeeper. The SWRL rules which identify the scenario are shown in Table 8-4.

Textual description	SWRL Encoded Rule
When EP1 tries to initiate a call with EP2 using GK1, call setup fails.	`h323EndpointCallCaller (?ep1callsep2,?ep1)^` `h323EndpointCallCallee (?ep1callsep2,?ep2)^` `h323EndpointCallResultCode (?ep1callsep2,”failed”)`
EP1 is not registered with any gatekeeper.	`h323EndPointRegistrationRegisteredEndpoint (?ep1reggk1,?ep1)^` `h323EndPointRegistrationActive (?ep1reggk1,false)`

Table 8-4 – SWRL rules used for test case 2

In one of these SWRL rules, one of the limitations caused by the use of *open world assumption* in DL and thus OWL and SWRL is shown. Since it is not possible to assume anything about knowledge that is missing (or *null*), one have to implement workarounds. In this case, the endpoints need to model a registration to a gatekeeper even if it the registration is not used. The registration is instead tagged as inactive by the h323EndPointRegistrationActive relation.

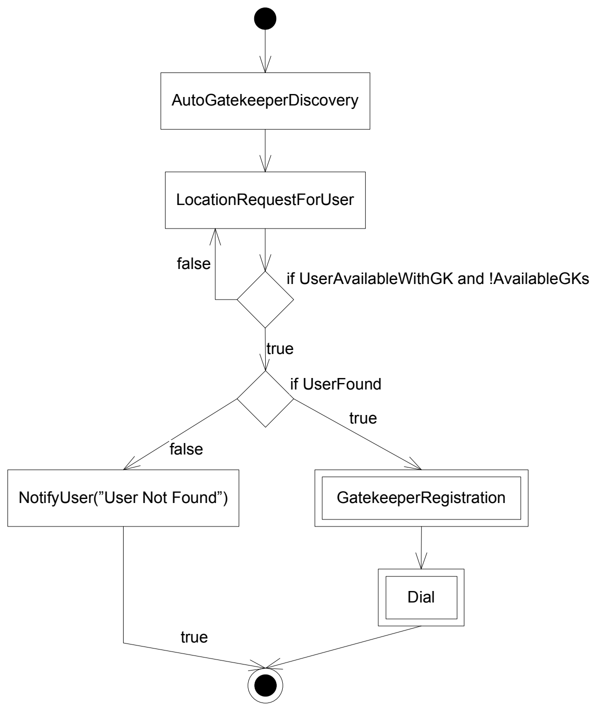


Figure 8-14 – Self repair process believed to correct the second case scenario

Although the error scenario is quite simple, the repair procedure is a bit more complicated as we can see in Figure 8-14. First all available gatekeepers are found using the AutoGatekeeperDiscovery. Then, the procedure connects and iterates through all these, asking each whether they know the user. This will end either when the user has been found or if there are no more available gatekeepers. If no user was found, there is sent out a notification before the procedure ends. Alternatively, the user is found and the unit will proceed to register itself with the gatekeeper which knew the user, and then redial.

8.3.3 Test Case 3

This last scenario is also quite simple. Here, the endpoint 1 tries to register with the gatekeeper but, as always, the registration fails. This time, the problem lays in the fact that endpoint 1 as disabled its authentication feature, while the gatekeeper requires it to be enabled.

Textual description	SWRL Encoded Rule
EP1 tries to register with GK1, but security-denial is returned	`h323EndPointRegistrationRegisteredEndpoint (?ep1reggk1,?ep1)^` `H323EndpointRegistrationRegisteredWithGatekeeper (?ep1reggk1,?gk1)^` `H323EndpointRegistrationResultCode (?ep1reggk1,"security-denial")`
GK1 requires authentication but EP1 has authentication turned off	`h323GatekeeperConfiguration (?gk1,?gk1configdoc)^` `authentication (?gk1configdoc,true)^` `h323TerminalConfiguration (?ep1,?ep1configdoc)^` `authentication (?ep1configdoc,false)`

Table 8-5 – SWRL rules used for test case 3

The difference between this repair procedure and the others is that it requests input from the user. Although self-configuring systems are supposed to act on their own, there are times where information must be acquired from the user. Sometimes, there might be time-constraints on how long the system can wait while the user types in the required information. In these cases, the procedure language should be able to insert timers on such operations. According to the OWL-S specification, this can be done using special timeout processes. These are however not currently implemented in any of the OWL-S editors. The procedure in Figure 8-15 is thought to solve the scenario, but is on the basis of the discussion in this paragraph not able to specify how long the user has to input the necessary authentication details.

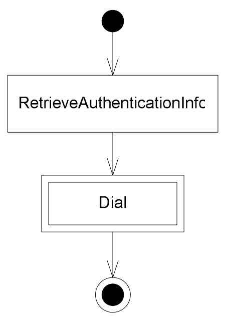


Figure 8-15 – Recovery process for test case 3 specified in OWL-S

8.4 Experiment Results

In this section, the results of experiments aimed at testing the feasibility of the proposed architecture are presented.

8.4.1 Experiment 1 – Complete Rule Base Firing

The first experiment was conducted to check how SWRL rules could be used to reason over the OWL domain ontology modelled during this thesis. This is necessary if the technology is to be used for classification of error scenarios. The actual experiment was carried out using the SWRL plug-in of Protégé (see appendix A.1) together with a custom translation function which translated the SWRL rules into executable rules for the Jess rule engine [38], a well-known open source rule engine. The Jess rule engine was then fed with the modelled domain ontology with instances having the right characteristics inserted for each test case. The results of the experiment can be seen in Table 8-6.

Test case	Description	Experiment result
Test case 1	Entire rule (Table 8-3) entered into plug-in, every horn clause element is checked before concluding the experiments results	Failed, built-in function is not implemented
Test case 2	Entire rule (Table 8-4) entered into plug-in, every horn clause element is checked before concluding the experiments results	Passed
Test case 3	Entire rule (Table 8-5) entered into plug-in, every horn clause element is checked before concluding the experiments results	Passed

Table 8-6 – Results from testing of SWRL on the domain ontology

As we can see, the second and last test case executed as expected, thereby passing the experiment. The first test case, failed its experiment as the SWRL reasoner lacked support for the necessary *matcher* built-in function. These will however be added to the plugin in the near future according to the developer (Table A-1). Based on these simple results, it seems feasible to use SWRL in order to classify error scenarios.

8.4.2 Experiment 2 – OWL-S Execution

This experiment will test whether the correct services will be executed when using process models combined with the OWL-S API. However, as the API unfortunately does not currently support any conditionals, the only thing that the code was able to verify, was that the API correctly parsed the OWL-S file and tried to execute some of the scenarios in correct order. First, the entire process ontology presented in section 8.2 was entered into the OWL-S plug-in of Protégé, and then the error scenarios was created as own services. The resulting OWL files were then fed into the OWL-S test parser.

Test case	Description	Experiment result
Test case 1	Entire OWL-S encoded model of Figure 8-13 was fed into OWL-S API and the output log was inspected to ensure that the procedure was presented in the right order and way.	Passed
Test case 2	Entire OWL-S encoded model of Figure 8-14 was fed into OWL-S API and the output log was inspected to ensure that the procedure was presented in the right order and way.	Failed, conditionals not implemented
Test case 3	Entire OWL-S encoded model of Figure 8-15 was fed into OWL-S API and the output log was inspected to ensure that the procedure was presented in the right order and way.	Passed

Table 8-7 – The results from experiments on OWL-S parsing

As we can see, test case 2 failed as it was expected to do, since the current version of the OWL-S API does not support control structures. However, the other test scenario files were easily parsed and executed as they were meant to.

9 Evaluation

Whenever proposing, designing or implementing a computer application, great care should be taken into a critical evaluation of the result. Such evaluations are often used to point out critical issues in the application and sum up its weaknesses, as well as propose how it can be made better. In this chapter, an evaluation of the proposed OSCOS architecture including its underlying technologies is presented.

9.1 Scalability

A definition of scalability is:

> *"(...) the ability of a solution to a problem to work when the size of the problem increases." [90]*

The proposed OSCOS architecture follows an agent-oriented approach which is known to give stable, reliable and adaptable systems [19]. This means that much of the system's functionality is placed in the different agents. By doing so, and also requiring each event to have its own handling process both in the event handler as well as in the repair service, facilitates parallel behaviour, distribution of the processing load, and also allows for an easier physical distribution of its objects. There is however areas in the OSCOS architecture that needs to be assessed as to its scalability in further work.

First of all it is somewhat unclear what the best way for units to map their knowledge into ontology-driven knowledge bases is. One solution to this is to implement a monitoring component which needs to have direct access to the computational environment of the software component, and use this information to construct the knowledge base. This has however been shown in previous research (e.g. [1], [30] and [114]) to be potentially quite time-consuming and complex [30]. It may also be shown how such a solution could quickly grow rigid and be hard to adapt to changing environments with many software components. The portrayed solution in this thesis was therefore to provide each software component with an own monitoring process to which the component would report changes in its runtime environment too. This is a computationally cheap solution, but might cause complications to the system's inner working as any change would have to be reported to the mapping process. However, considering the other options, it seems to be a more scalable solution than a realisation depending on memory access.

Although it is possible to implement OSCOS in single self-contained systems, most realisations will most likely use a distributed architecture. Because the system is designed to handle real-time analysis of data, the repair service also has to be reasonably fast. This means that it has to be fast enough to satisfy the delay requirements of a system while performing real-time classification of error scenarios. As the repair service is the heart of the system, such issues may also cause it to become the bottleneck if enough care is not taken during realisation and design. If the reasoning process uses too much time and processing power, this will severely hurt the general performance and in consequence the user's willingness to depend on the system. To aid this issue, the repair service was designed to split each request for a repair procedure into an own process and thus allowing for an easy physical distribution of the requests onto several computational devices. This does not however solve the entire scalability problem of this component. The OWL and SWRL reasoning engines are truly the biggest computational drains in the entire system. Even if several engines were deployed on different computation devices, checking an ontology for e.g. consistency is quite the challenge as shown by Haarslev et al. in [50]. In addition to this, the SWRL – OWL combination may in fact be undecidable with the wrong inputs [57]. To aid in both of these issues, the OSCOS architecture proposes to limit the information which is checked by the reasoning engines to the minimum necessary to solve the problem. Specifically, this is accomplished by requiring each unit to only contain and maintain their own local information in their respective knowledge bases. When the unit later request assistance from the repair service, only this local information is sent. The repair service will acquire the rest of the necessary information from other units and ontology repositories. By doing so, only small sub-domains of information will be fed into the engines and this should thus help in achieving fast and less computationally intensive queries.

9.2 Rule-Based Systems

The OSCOS architecture uses a rule-based way of identifying error scenarios. According to Garga et al. in [40], the main limitations of rule-based systems are:

> *"(...) [a] combinatorial explosion and consistency maintenance" [40]*

This mean that not only do rule sets in such systems have a tendency to grow larger and more complex over time [40], but if the administrative users do not have an easy-to-asses overview of the existing rules and a simple way to configure them, the system might render itself useless [89].

Walker elaborates on these issues when he in [113] further states that rule-based expert systems are only as good as the declarative knowledge that they contain [113]. This means that if an expert system is to function properly it has to be updated frequently to keep up with the development in its error domain [60]. However, although the system may be effective and accurate in some given situations, there are times in which the conditions reflected in a rule are not absolutely certain [102]. There might be situations where it is not clear how to derive the classifying conditions for a given error. In such cases it is often suggested to append a certainty measure to the classifying rules [118]. This uncertainty could also be modelled by assigning a certainty metric to the conclusion, thus suggesting that a conclusion is given with e.g. 90% certainty [118]. The SWRL standard used in this thesis does not currently include support for expressing such rules but it has been suggested, and also expected, to be added in later releases [87].

Checking rule sets for consistency generally means determining whether a given rule set contains rules that are either overlapping or conflicting. Rule overlaps occur when more than one rule can lead to the same action. This is not necessarily a mistake, as many error-scenarios have the same solution. Conflicting rules are rules that trigger in the same situation, but lead to different actions [60]. Although this checking can be done at run-time, it is generally preferred to detect these as soon as possible. The rule sets should therefore be checked for consistency after inserting a new rule. This can be done by using logical techniques like forward or backward chaining [65], by utilising expert system verification tools such as SYCOJET or SACCO [6], or visual inspection. The latter is easier to do if the rules are presented to the administrative user in a way which makes it easier to keep track of the rule base [60]. Hutschemaekers et al. suggest in [60] such a rule grouping based on:

- the events triggering the rule
- the action that is taken

As we see, this grouping is actually a way to make the previously mentioned logical techniques of forward and backward chaining easier and more intuitive [60]. However, not all overlaps or conflicts can be detected at design time, so some will only be discovered when the system is put into real world action. In these cases, the system designer needs to make a choice of what should be the default action in such situations. This decision can either be made by rule sorting algorithm like shown by Lindgren [75] or on the basis of rule priority, age, source, most specific, least recently used, etc, as shown by Lee et al. in [73].

9.3 Technology Issues

There are some issues regarding the underlying technology used in the system proposed in this thesis. This section presents these and other related limitations.

9.3.1 OWL Reasoning

Even if both OWL DL and OWL Lite guarantee completeness and decidability, there still exist performance issues in OWL reasoning. The actual performing of reasoning tasks on an OWL ontology could easily become computationally infeasible when used in e.g. an interactive service with a large number of users [2]. It has, however, been shown that some of the performance issues may be reduced by adding a disproportionate amount of memory to the reasoning unit [58]. Still, this need for either a high computational strength or a very large memory bank, makes any use of built-in OWL reasoning on small computation devices such as mobile phones etc., unfeasible. This implies that at least the ontology consistency checking, and probably also the error scenario matching services should be placed on central servers.

9.3.2 Limitations in OWL Expressivity

As mentioned in section 5.2, OWL DL, as any other particular DL, allow only for a subset of the given operators and connectives. This means that one may trade expressive power for computer efficiency and vice-versa. When OWL DL was created, a certain blend of connectives and operators was chosen. The blend was, within the available space of the DL languages, as expressive as possible without becoming undecidable [55]. This, combined with the use of a logic-based knowledge representation approach, makes OWL walk a fine line between expressivity and tractability [92]. As it shows, there are several expressivity limitations in OWL DL which now affect the modelling power of the language [92]. Many of the limitations of OWL stem to the fact that while the language features a relatively rich set of class constructors, the language provided to assess properties is much weaker [57].

9.3.2.1 Cross-slot Constraints and Operations

OWL is only able to place restrictions on one single property at a time. This means that one is allowed to state that a given class *C*'s property *p* has a certain value *"v"*, or that it may only have a value chosen from the set *{"v", "a", "l", "u", "e"}*. It is, however, not possible to express that the value of property *p1* must be greater that the value of property *p2*, or that the value of property *p3* is the sum of the previously mentioned properties *p1* and *p2*.

9.3.2.2 Identity Criteria

A somewhat related consequence of the ability to only place and assess restrictions on a single property at a time is that the resource identity criteria are also limited to one slot at a time. This means that is not possible to express anything equivalent to composite keys in ER-modelling.

9.3.2.3 Property Composition

OWL is unable to express the fact that one property may be the composition of two other properties. This means that it is e.g. impossible to define the relation *uncle* as a direct composition of the properties *brother* and *parent* [57].

9.3.2.4 Defaults

It is also impossible to describe default values for a property in OWL. This is another consequence of the *open world assumption* (5.2). If for instance a reasoner assumed that because it had not yet seen a value for a given property belonging to an individual, it should use the default instead, then this might become a problem if the correct value later arrived.

9.3.3 SWRL Reasoning

Although the SWRL layer alleviates many of the limitations associated with OWL, it has some limitations of its own:

- The syntax is a bit too low-level to be efficient for human processing and it quickly becomes cumbersome and hard to write large and complex rules.

- Since SWRL, as OWL, only support binary relations, n-ary relations are needed as shown in section 4.3.2. This not only necessitates greater care in ontology and rule design, but also complicates the rules as they now have to first identify the n-ary relation instance for all members before any further reasoning can be done. An example of such a rule can be seen in Table 8-3 where the call relation between two endpoints is modelled.

- SWRL inherits the *open world assumption* and assumes thus always that something is just lacking rather than false. Because of this, work-arounds for cases were one would typically use *null,* has to be made. This was encountered in the second test case where a unit was not connected to any gatekeeper. In the test case, this normal *null* relation instead had to be realised by adding a new boolean property indicating whether the registration relation was active or not.

- Dealing with OWL constraints within a rule engine is not trivial to say the least and as this is written, no available rule engines fully support the handling of OWL constraints. This might in consequence lead to conflicts and inconsistencies as shown by Grosof et al in [47].

There are also issues which seem to stem in large from SWRL's lack of maturity. There are currently no available graphical editors which are able to store SWRL rules directly in the defined SWRL/XML format, thus requiring the insertion of rules to be written manually. This makes the management of SWRL rule bases even harder. Also, none of the SWRL reasoning engines available today support many any the built-in functions suggested in the SWRL specification ([54]). This is disappointing as several of these add useful functionality to the reasoning scheme such as matcher(?a, ?b). This function checks whether two string variables or regular expressions matches with each other.

9.3.4 OWL-S Process Models

The OWL-S Process models gave the OSCOS architecture a simple way to specify recovery procedures for pre-defined error scenarios. However, during the experiments conducted in this thesis, some limitations of today's situation were found. Firstly, there is currently no way to specify time-constraints on services or actions. Such constraints could be useful e.g. in situations where a user input is needed, but the system does not have an unlimited amount of time available. Another limitation is the fact that there is no way to indicate system states. The use of system states would make several operations easier as it probably could reduce situation reasoning and process choices. Finally, not only are there currently few available OWL-S executors, but these also have several limitations themselves as they do not fully support all the OWL-S control mechanisms. This stem in large from the OWL-S being a very new standard, and will probably be improved in near future.

9.3.5 Ontology Inconsistencies

As mentioned in section 5.2.2, *description logics*, and thus OWL DL make the so called *open world assumption* which states that something that is not explicitly expressed denotes lack of knowledge. This stands in contrast to systems that make the *closed world assumption,* where a lack of knowledge is assumed to indicate false. Such lack of knowledge requires for the ontology designer to always remember defining explicitly what can and can not be done in the ontology. As an example of a direct consequence of the *open world assumption* is e.g. that for classes not explicitly defined as disjoint, it is perfectly legal to share individuals [7]. This means that if classes are not explicitly stated as disjoint, the instance Mike can be a member of both the human ontology classes Man and Woman. This is obviously an inconsistency in most cases, but would be considered legal by any OWL reasoner. However, if the classes were defined as disjoint, the instance of Mike would in the same situation cause an inconsistency as a human has to be a man or a woman.

Another important issue regarding OWL reasoning is the fact that, as explained in section 5.2, one of the OWL dialects, OWL Full, does not have computation guarantees of its solvability. Because of this, OWL Full ontologies using elements such as metaclasses are currently not supported by any available OWL reasoners. It is therefore vital that any domain ontologies confine to either the OWL DL or the OWL Lite dialect.

Because of these problems, any ontology needs both during and after the design to be checked for inconsistencies and for dialect compliance [44]. This is most efficiently done by employing an ontology reasoner (e.g. Pellet [99]) which is able to detect both of these issues.

10 Conclusions and Future Work

During the last century, advances in technology have brought on many revolutionary changes. Amongst others, the century saw the dawn of the information society, driven by increasingly advanced computing systems. Vast possibilities are now provided to its users. However, there are also problems related to this explosive growth. As the complexity of modern communication systems increases, so does the need for proper configuration and management. The problem is that in the future it will not only be a shortage for people with the right knowledge and skills to manage computer and telecommunications systems, but the complexity will have grown beyond any human's ability to manage it. It is therefore desirable to develop self-configuring systems which can govern themselves without human intervention.

This thesis has explored how such self-configuring systems can be constructed using Semantic Web technologies. The Semantic Web is an effort to give meaning to information in a machine-readable way. This is done using a knowledge representation technique called ontologies. Ontologies are formal and explicit specifications of shared conceptualisations [103]. Specifically, it is investigated how one can utilise such ontologies in order to classify pre-defined error scenarios using real-time data. This is achieved by using the Web Ontology Language (OWL) combined with another Semantic Web technology - the Semantic Web Rule Language (SWRL). Such classification is suggested realised using three separate steps: first by modelling an ontology of the given domain using OWL, then inserting the necessary information using ontology instances and finally to use SWRL rules to determine which error scenario the given situation should be classified as. In order to determine when such an analysis is needed, it is suggested to use error events. Such events are the unit's way of notifying monitoring services that something of significance has occurred. The thesis has also shown how one can use the OWL service specification ontology, OWL-S, to specify and describe which actions the unit should take in order to recover from a given scenario.

In addition to the theoretical work, this thesis also proposes an architecture for realising a Semantic Web driven self-configuring system. This includes block diagrams, complete behavioural models and message sequence charts. Because of the formal modelling languages used during the modelling of this architecture, one is able to quite easily transform these using a model-driven approach into a framework of executable code to which the more specific functionality may be added (including OWL-S execution etc).

To show that the suggested approach is sound and feasible, a test domain was modelled and two experiments were conducted. Although some of the sub-tests failed due to the immaturity of the Semantic Web field and its technologies, the approach seemed to be promising. A complete realisation of the architecture will, however, require that these underlying technology issues are resolved.

10.1 Contributions

Throughout this thesis, many lessons were learned and discoveries made. This section will present the main contributions to the research area.

10.1.1 Theoretical Foundation for Semantic Web Driven Self-Configuration

Although other research studies have explored how Semantic Web technologies can be used in areas such as network management, this thesis is, to the authors' knowledge, the first which propose to employ these technologies in an autonomic self-configuring scheme.

In order for such a Semantic Web driven self-configuration system to be realised, many underlying technologies have to be assessed. This thesis has presented and described the relevant and enabling technologies required for such an approach. These include autonomic computing, ontologies, description logics and Semantic Web technologies which are all presented and discussed in relation to this thesis. Several weaknesses, limitations and immaturity issues of the same technologies have also been pointed out. The goal to all this has been to provide a clear, structured and solid theoretical foundation for which further research studies and work can be based on.

10.1.2 Proposed Architecture

OSCOS is a proposed architecture for how a Semantic Web driven self-configuring system can be realised. This proposal includes thorough discussion on most of the principal architectural choices, such as how to specify what a unit should do to recover from a given error scenario. The architecture is constructed around the ontology concept and requires each unit to create and maintain an ontological representation of itself using OWL-encoded instances. Such a representation is also called world-modelling and is a prerequisite for self-aware systems. Whenever a unit experiences an error, it will dispatch an error event. This will initiate a classification procedure driven by rules expressed in SWRL. If any scenarios match the unit's current situation as modelled in its representation, this will then execute a recovery procedure modelled in OWL-S. Although it is possible to implement OSCOS in single self-contained systems, most realisations will most likely use a distributed architecture.

The OSCOS specification includes:

- Software block diagrams with specified communication routes between the different components

- Complete behavioural models expressed by state machines

- Data flow shown through message sequence charts

Because of the formal languages (UML2.0) used during the modelling of this architecture, one is able to quite easily transform these using a model-driven approach into an executable framework to which the more specific functionality may be added (including OWL-S execution etc).

10.1.3 Feasibility Studies

In order to show that the technologies suggested in the proposed architecture could handle the tasks which they were expected to carry out, experiments were designed and executed. The first experiment tested whether SWRL rules could be used to classify pre-defined error scenarios using an ontological representation of a unit's current situation. The tests were successful in two of the test cases, but failed in one. This was due to the inability to use SWRL built-in *matcher* function.

In the second experiment, the proposed error recovery approach was tested. This meant testing if a machine could parse and execute correctly a recovery procedure specified using OWL-S. Just as the first experiment, two of the tests were successful while one failed. Again, this was due to an immature technology as the required flow control conditionals were not yet implemented.

Although one sub-test in both the experiments failed due to immaturity of the Semantic Web field and its technologies, the approach seemed to be promising and certainly realisable. However, the experiments also showed that a complete realisation of the OSCOS architecture will require that these underlying technology issues are resolved.

10.1.4 H.323 Ontologies

The prerequisites of the experiments carried out in this thesis were: an OWL-encoded domain ontology and a process ontology which showed the action capabilities of a unit specified in OWL-S. Because of this, an ontology of the chosen test domain, the H.323 standard, was modelled. This ontology is not complete as it, amongst other things, lacks complete specifications of the possible configuration files of its units, but is a good framework for later case studies in the H.323 domain. A process ontology for the H.323 Terminal was also designed.

10.2 Future Work

As stated in this thesis' research statement, its purpose was to produce a solid foundation in which further research and realisations can be based on. This means however that there is still work to be done in order for a Semantic Web-driven self-configuring architecture to be realised.

In future studies a demonstrator service needs to be implemented in order to fully show the feasibility of the approach. Such a demonstrator's framework can be generated from the behavioural models in this thesis. However, in order for such a demonstrator to be realised, several currently underlying technology issues which need to be resolved. This includes establishing and deploying SWRL's built-in functions as well as enabling ways to correctly parse conditionals (if, while, etc.) during OWL-S process execution.

The focus of this thesis has been on a rule-based approach to the classification of error scenarios. It is however a well researched view that even if such rules may be effective and accurate in some given situations, they are not enough to provide full autonomic function [102]. Another and more dynamic approach is classification through machine learning which helps us circumvent this and other issues such as the rigidness and inability to adapt to unknown environments. Machine learning is a term for algorithms and techniques used to help computers "learn" from its experiences [81]. It has been shown that the predictions of such approaches often have a high degree of accuracy with the right training data [81]. Future research should therefore investigate and assess how machine learning techniques can be used to further improve on the OSCOS system’s error classification.

In the distant future where technologies such as the Semantic Web are implemented, one can only imagine the possibilities. A scenario where the OSCOS units themselves decides to scrap the central repair suggester and choose to search for error solution procedures through e.g. web searching engines may be one of them…

11 References

[1] Abela, C., Montebello, M., “DAML enabled Agents and Workflow Components Integration”, Proc. of the IADIS International Conference WWW/Internet 2002, November 2002

[2] Agostini, A., Bettini, C. and Riboni, D., “Experience Report: Ontological Reasoning for Context-aware Internet Services”, Proc. of the 4th Annual IEEE International Conference on Pervasive Computing and Communications, March 2006, pp. 1-8

[3] Albus, J. and Meystel, A., “Engineering of Mind”, John Wiley & Sons Inc., 2001, ISBN 0-471-43854-5

[4] Ankolekar, A., Paolucci, M. and Sycara, K., “Spinning the OWL-S Process Model - Toward the Verification of the OWL-S Process Models”, The 3rd International Semantic Web Conference (ISWC 2004), Semantic Web Services workshop, 2004

[5] Appleby, K., Goldszmidt, G. and Steinder, M., “Yemanja – A Layer Event Correlation Engine for Multi-domain Server Farms”, IEEE/IFIP International Symposium on Integrated Network Management Proceedings, 2001, pp. 329-344

[6] Ayel, M. and Vignollet, L., "SYCOJET and SACCO, two tools for verifying expert systems", International Journal of Expert Systems: Research and Applications, vol. 6, 1993, pp. 357-382

[7] Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D. and Patel-Schneider, P, “The Description Logic Handbook: Theory, Implementation and Applications”, Cambridge University Press, 2003, ISBN-13 9780521781763

[8] Baader, F., Horrocks, I. and Sattler, U., "Description logics as ontology languages for the Semantic Web", Mechanizing Mathematical Reasoning: Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday, vol. 2605 of Lecture Notes in Computer Science, January 2005, pp 228-248

[9] Baader, F. and Sattler, U., “Description Logics with Symbolic Number Restrictions”, Proc. of the 12th European Conference on Artificial Intelligence, 1996, pp. 283-287

[10] Berners-Lee, T., “Semantic Web on XML”, XML 2000, 2000, http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide1-0.html, accessed 27th February 2006

[11] Berners-Lee, T., Fielding, R. and Masinter, L., “Uniform Resource Identifier (URI): Generic Syntax”, IETF Network Working Group, January 2005

[12] Berners-Lee, T., Hendler, J. and Lassila, O., “The Semantic Web”, Scientific American, May 2001

[13] Borgida, A., “On the Relative Expressiveness of Description Logics and Predicate Logics”, Artificial Intelligence, vol. 82, 1996, pp. 353-367

[14] Borst, W.N., “Construction of Engineering Ontologies”, PhD thesis, University of Twente, Enschede, 1997

[15] Brandl, M., Franzens, K., Daskopoulos, D., Dobbelsteijn, E., Garroppo, R. G., Janak, J., Kuthan, J., Niccolini, S., Ott, J., Prelle, S., Ubikk, S. and Verharen, E., ”IP Telephony Cookbook”, TERENA Secretariat, 2004, ISBN 90-7759-08-6

[16] Bræk, R., “On Methodology Using the ITU-T Languages and UML”, Telektronikk, vol. 4, 2000, pp. 96-106

[17] Bræk, R. and Floch, J., “ICT convergence: Modeling issues”, System Analysis and Modeling (SAM), 4th International SDL and MSC Workshop, 2004, pp. 237-256

[18] Buchler, J., “Concept of Method”, University Press of America, 1985, ISBN 0819146714

[19] Burness, A.L., Titmuss, R., Lebre, C., Brown, K. and Brookland, A., “Scalability evaluation of a distributed agent system”, Distributed Systems Engineering, vol. 6, 1999, pp. 129-134

[20] Ciocoiu, M., Nau, D. S. and Grüninger, M., “Ontologies for Integrating Engineering Applications”, Journal of Computer and Information Science and Engineering, vol. 1, March 2001, pp. 12-22

[21] Connolly, D., Harmelen, van F., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F. and Stein, L. A., “DAML+OIL (March 2001) Reference Description”, Technical Report, DAML Program, December 2001

[22] Cooper, H. M., “Integrating research: A guide for literature reviews”, Sage Publications, 2nd edition, 1989, ISBN 0803934300

[23] Creswell, J. W., “Educational research: Planning, conducting, and evaluating quantitative and qualitative research”, Prentice Hall, 2nd edition, 2005, ISBN 013112790X

[24] Dashofy, E. M., Hoek, A. and Taylor, R., “Towards Architecture-based Self-Healing Systems”, Proc. of the first workshop on Self-healing systems, 2002, pp. 21-26

[25] DataBeam Corporation, “A Primer on the H.323 Series Standard”, DataBeam Corporation Whitepaper (available from http://www.databeam.com/, accessed 23rd March 2006) , May 15, 1998

[26] Dean, M. (editor), "OWL-S: Semantic Markup for Web Services", WC3 Member Submission 22 November 2004, WC3, 2004

[27] Dean, M., Schreiber, G., "OWL Web Ontology Language Reference", W3C Recommendation, WC3, 2004

[28] Dellinger, A., "Exploring Relationships Between the Validity and the Literature Review", The South Western Educational Research Association, February 2003

[29] Ding, Y., Fensel, D., Klein, M., Omelayenko, B., "The Semantic Web: Yet Another Hip?", Data and Knowledge Engineering, vol. 41, 2002, pp. 205-227

[30] Ding, Y., Litz, H., Malaka, R., and Pfisterer, D, "On Programming Information Agent Systems-An Integrated Hotel Reservation Service as Case Study", Proc. of the first German Conference on Multiagent System Technologies, 2003, pp. 50-61

[31] Eracar, Y. A., "Negotiating solutions to multi-objective combinatorial optimization problems", Ph.D. Thesis, Northeastern University, April, 2005

[32] Erdmann, M., and Studer, R., "How to structure and access xml documents with ontologies", Data and Knowledge Engineering, vol 36, 2001, pp.317–335

[33] Falbo, R. A., Menez, C. S. and Rocha, A. R. C., "Using Ontologies to Improve Knowledge Integration in Software Engineering Environments", ICAS'98, vol. 1, 1998, pp. 1-10

[34] Fensel, D., Harmelen, van F., Horrocks, I., McGuinness, D.L. and Patel-Schneider, P.F, "OIL: An Ontology Infrastructure for the Semantic Web", IEEE Intelligent Systems, vol. 16, 2001, pp. 38-45

[35] Fensel, D., "2000: Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce", Springer Verlag, 2001, ISBN 3540003029

[36] Fernandez, M., Gomez-Perez, A. and Juristo, N., "METHONTOLOGY: From ontological Art towards ontological engineering", Proc AAAI-97 Spring Symposium Series on Ontological Engineering, pp. 33-40

[37] Flanagan, D., "Java in a Nutshell", O'Reilly Media Inc, 2005, ISBN 156592262X

[38] Friedman-Hill, E., "Jess in Action: Java Rule-Based Systems", 2003, ISBN 1930110898

[39] Ganek, A. G. and Corbi, T. A., "The dawning of the autonomic computing era", IBM Systems Journal, vol. 43, 2003, pp. 5-18

[40] Garga, A. K., McClintic, K. T., Campbell, R. L., Chih-Chung Y., Lebold, M. S., Hay, T. A. and Byington, C.S., "Hybrid reasoning for prognostic learning in CBM systems", Proc. of the 2001 Aerospace Conference, vol. 6, March 2001, pp. 2957-2969

[41] Gay, L. R. and Airasian, P., "Educational research: Competencies for analysis and application", Pearson Education, 2003, 8th edition, ISBN 0131185349

[42] Genesereth, M. R. and Nilsson, N. J., "Logical Foundation of Artificial Intelligence", Morgan Kaufmann, 1987, ISBN 0-934613-31-1

[43] Gennari, J., Musen, M., Fergerson, R., Grosso, W., Crubézy, M., Eriksson, H., Noy, N. and S. Tu, "The evolution of Protégé-2000: An environment for knowledge-based systems development", International Journal of Human-Computer Studies, vol. 58, 2003, pp. 89–123

[44] Gomez-Perez., A., "Some Ideas and Examples to Evaluate Ontologies", the 11th Conference on Artificial Intelligence for Applications, 1995, pp 299-305

[45] Gruber, T. R., "A translation approach to portable ontology specifications", Knowledge Acquisition, vol. 5, 1993, pp. 199-220

[46] Gruber, T. R., "Toward principles for the design of Ontologies used for knowledge sharing", International Journal Human-Computer Studies, vol. 43, 1993, pp. 907-928

[47] Grosof, B., Horrock, I., Volz, R. and Decker, S., "Description Logic Programs: Combining Logic Programs with Description Logic", Proc. of the 12th International Conference on the World Wide Web (WWW 2003), 2003

[48] Guarino, N., "Formal Ontology and Information Systems", Proceedings of international conference on formal ontology in information systems (FOIS'98), 1998, pp. 3-15

[49] Guerrero, A., Villagrá, V. A. and Vergara, J. E. L., " Including management behavior defined with SWRL rules in an Ontology-based management framework, Proc. of the 12th Annual Workshop of HP Openview University Association (HP-OVUA'2005), July 2005

[50] Haarslev, V., Möller, R. and Wessel, M., "On the Scalability of Description Logic Instance Retrieval", Proc. of the 29th Annual German Conference on Artificial Intelligence, LNCS, 2006

[51] Halvorsen, K., "Å forske på samfunnet", Bedriftsøkonomens forlag, 1993, ISBN 8202226546

[52] Hellevik, O., " Forskningsmetode i sosiologi og statsvitenskap", Universitetsforlaget, 1999, ISBN 82-00-12992-6

[53] Horn, P., “Autonomic Computing: IBM perspective on the state of the information technology”, Presented at AGENDA 2001 (available from http://www.research.ibm.com/autonomic/), 2001

[54] Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B. and Dean, M., “SWRL: A Semantic Web rule language combining OWL and RuleML”, W3C Member Submission, 21 May 2004, WC3, 2004

[55] Horrocks, I., Patel-Schneider, P. F. and Harmelen van F., “From SHIQ and RDF to OWL: The Making of a Web Ontology Language”, Journal of Web Semantics, 2003, pp. 7-26

[56] Horrocks, I. and Patel-Scheider, P. F., “Reducing OWL entailment to description logic satisfiability”, Proc. of the 2003 International Semantic Web Conference (ISWC 2003), 2003

[57] Horrocks, I. and Patel-Scheider, P. F., “A proposal for an OWL rules language”, In WWW ’04: Proc. of the 13th international conference on World Wide Web, ACM Press, 2004, pp. 723–731

[58] Horrocks, I., “DAML+OIL and Description Logic Reasoning”, Talk given at HP Labs, October, 2001, http://www.cs.man.ac.uk/~horrocks/Slides/hp-labs-prn.pdf, accessed 10th June 2006

[59] Humns, N. and Singh, M., “Ontologies for Agents”, IEEE internet Computing, Nov – Dec 1997

[60] Hutschemaekers, M., Plas, D. J., Verheijen, M. and Zwaal, H., ”Manipulating context information with SWRL”, Freeband A-Muse Project, Deliverable D3.12, March, 2006

[61] IBM Corporation, “An Architectural Blueprint for Autonomic Computing”, IBM Technical White Paper, IBM Press, 2004

[62] IEEE, “IEEE Standard Glossary of Software Engineering Terminology”, IEEE std. 610.12-1990, 1990

[63] International Telecommunication Union (ITU), “Security and encryption for H-series (H.323 and other H.245-based) multimedia terminals”, ITU-T Recommendation H.235, ITU-T, 2003

[64] International Telecommunication Union (ITU), “Visual Telephone Systems and Equipment for Local Area Networks which Provide a Non-Guaranteed Quality of Service”, ITU-T Recommendation H.323, ITU-T, 1996

[65] Ioannidis, Y. and Sellis, T. K., “Conflict Resolution of Rules Assigning Values to Virtual Attributes”, Proc. of the 1989 ACM SIGMOD international conference on Management of data, 1989, pp. 205-214

[66] Jakkilinki, R., Sharda, N., Georgievski, M., “Developing an Ontology for Teaching Multimedia Design and Planning”, M2USIC 2005: MMU International Symposium on Information and Communication Technologies

[67] Jones, D.M., Bench-Capon, T.J.M. and Visser, P.R.S, “Methodologies for Ontology Development”, Proc. of IT&KNOWS Conference for the 15th IFIP World Computer Congress, 1998

[68] Kalyanpur, A., Pastor, D. J., Battle, S. and Padget J., “Automatic Mapping of OWL Ontologies into Java, Proc. of the Sixteenth International Conference on Software Engineering and Knowledge Engineering, June 2004, pp. 98-103

[69] Kettler, B., Starz, J., Miller, W. and Haglich, P., "A Template-based Markup Tool for Semantic Web Content", Proc. of the 4th International Semantic Web Conference (ISWC 2005), November 2005

[70] Kitchenham, B., “Procedures for Performing Systematic Reviews”, Keele University Technical Report, Keele University Press, ISSN 1353-7776

[71] Laddaga, R., “Active Software”, Proc. of the First International Workshop on Self-Adaptive Software (IWSAS’00), 2000, pp. 11-26

[72] Lara, R., Lausen, H., Arroyo, S., Bruijn, de J. and Fensel, D., “Semantic Web Services: description requirements and current technologies”, In Semantic Web Services for Enterprise Application Integration and e-Commerce workshop (SWSEE03), in conjunction with ICEC 2003

[73] Lee, K. M., Sohn, B. K., Kim, J. T., Lee, S. W., Lee, J. H., Jeon, J. W. and Cho, J., “An SoC-Based Context-Aware System Architecture”, Proc. of the 8th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES 2004), Lecture Notes in Computer Science, vol. 3215, 2004, pp. 573-580

[74] Lin, P., MacArthur, A. and Leaney, J., “Defining Autonomic Computing: A Software Engineering Perspective”, Australian Software Engineering Conference, 2005, pp. 88-97

[75] Lindgren, T., “Methods for Rule Conflict Resolution”, Proc. of 15th European Conference on Machine Learning (ECML 2004), Lecture Notes in Computer Science, vol. 3201, 2004, pp. 262-274

[76] Martin-Flatin, J. P. and Znaty, S., “Annotated Typology of Distributed Network Management Paradigms”, Proc. of the 8 IFIP/IEEE Int. Workshop on Distributed Systems Operations and Management (DSOM'97), Oct. 1997

[77] Matthews, B., “Semantic Web Technologies”, CCLRC Rutherford Appleton Laboratory Technical Report, JISC Technology and Standards Watch, 2005

[78] McGuinness, D. L. and Harmelen F., “OWL Web Ontology Language - Overview”, WC3 Recommendation 10 February 2004, WC3, 2004

[79] Medvidovic, N., Oreizy, P. and Taylor, R., “Reuse of Off-the-Shelf Components in C2-Style Architectures”, Proc. of the 1997 International Conference on Software Engineering (ICSE’97), 1997, pp. 692-700

[80] Merriam-Webster Online Dictionary, http://www.m-w.com/

[81] Mitchell, T.M., “Machine learning”, McGraw Hill, 1997, ISBN 0070428077

[82] Muller, M. J., “Catalogue of Scenario-Based Methods and Methodologies”, Lotus Research, Lotus Technical report, vol. 6, 1999

[83] Ng, G., “Open vs. Closed world, Rules vs Queries: Use Cases from Industry”, Proc. of the OWL Workshop at the International Conference on Rules and Rule Markup Languages for the Semantic Web, 2005

[84] Noy, N. F. and McGuinness, D. L., “Ontology development 101: A guide to creating your first ontology”, Stanford technical report, Stanford University, 2001

[85] Noy, N. F., Rector, A., “Defining N-ary Relations on the Semantic Web”, WC3 Working Group Note 12 April 2006, WC3, 2006

[86] OpenCyc Upper Ontology, http://www.opencyc.org/

[87] Pan, J. Z., “Requirements for a Semantic Web Rule Language”, Proc. of W3C Workshop on Rule Languages for Interoperability, 2005, http://www.w3.org/2004/12/rules-ws/paper/51, accessed 15th April 2006

[88] Passin, T.B., “Explorer’s Guide to the Semantic Web”, Manning Publications Co., Aug. 2004, ISBN 1932394206

[89] Pires, L. F., Sinderen, M. van, Munthe-Kaas, E., Pokraev, S., Hutschemaekers, M. and Plas, D. J., ”Techniques for describing and manipulating context information”, Freeband A-Muse Project, Deliverable D3.5, October, 2005

[90] Rana, O.F. and Stout, K., “What is Scalability in Multi-Agent Systems?”, Proc. of the 4th International Conference on Autonomous Agents, 2000, pp. 56-63

[91] Regnell, B., Kimbler, K. and Wesslen, A., ” Improving the Use Case Driven Approach to Requirements Engineering”, Proc. of Second IEEE International Symposium on Requirements Engineering (RE’95), March 1995, pp. 40-47

[92] Reynolds, D., Thompson, C., Mukerji, J., Coleman, D., “An assessment of RDF / OWL modelling”, HP technical report, HP Laboratories Bristol, Hewlett-Packard, HPL-2005-189, 2005

[93] Robinson, W. N., “Implementing Rule-Based Monitors within a Framework for Continuous Requirements Monitoring”, Proc. of the Hawaii International Conference on System Sciences, 2005

[94] Sanders, R. T., “Service-Centred Approach to Telecom Service Development”, Proc. of 8th EUNICE and IFIP Workshop on Adaptable Networks and Teleservices, September 2002

[95] Sattler, U., “Description Logics for Ontologies”, Proc. of the International Conference on Conceptual Structures (ICCS 2003), Lecture Notes in Artificial Intelligence, volume 2746, Springer Verlag, 2003

[96] SATURN project, http://semanticobjectweb.isx.com/

[97] Schlobach, S., Huang, Z. and Cornet, R., “Inconsistent Ontology Diagnosis: Evaluation”, SEKT: Semantically Enabled Knowledge Technologies, Deliverable D3.6.2 (WP3.6), January 2006

[98] Schultz, D.J., “IEEE standard for developing software life cycle processes. IEEE Std 1074-1997”, The Institute of Electrical and Electronics Engineers, 1997

[99] Sirin, E. and Parsia, B., “Pellet: An OWL DL Reasoner”, Proc. of the 2004 International Workshop on Description Logics (DL2004), CEUR Workshop Proceedings, vol. 104, 2004

[100] Sterritt, R., Parashar, M., Tianfield, H. and Unland, R., “A concise introduction to autonomic computing”, Advanced Engineering Informatics, vol. 19, pp. 181-187, 2005

[101] Sterritt, R. and Bustard, D. W., “Autonomic computing — a means of achieving dependability?”, Proc. of IEEE international conference on the engineering of computer based systems (ECBS’03), pp. 247-251

[102] Sterritt, R., “Towards Autonomic Computing: Effective Event Management”, Proc. of the 27th Annual IEEE/NASA Software Engineering Workshop, Dec. 2002, pp. 40-47

[103] Studer, R., Benjamins, R. V. and Fensel, D., "Knowledge engineering: Principles and methods", Data and Knowledge Engineering, vol. 25, 1998, pp. 161–197

[104] Tabet, S., Boley, H. and Wagner, G., "Design rationale of RuleML: A markup language for Semantic Web rules". Proc. Semantic Web Working Symposium, 2001, pp. 381–402

[105] The Unicode Consortium, "The Unicode Standard, Version 4.1.0, defined by: The Unicode Standard, Version 4.0", Addison-Wesley, 2003, ISBN 0-321-18578-1

[106] Thom, G. A., "H.323: the multimedia communications standard for local area networks", IEEE Communications Magazine, Vol. 34, December 1996, pp. 52-56

[107] Toga, J. and El Gebaly, H., "Demystifying Multimedia Conferencing over the Internet Using the H.323 Set of Standards", Intel Technology Journal, vol. Q2, 1998

[108] Tosic, V. and Djordjevic-Kajan, S., "The Common Information Model (CIM) Standard - An Analysis of Features and Open Issues", Proc. of the 4th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services (TELSIKS '99), vol. 2, Oct. 1999, pp. 677-680

[109] Uschold, M., "Towards a Methodology for Building Ontologies", Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95 in Montreal, 1995, July

[110] Uschold, M. and Gruninger, M., "Ontologies: Principles, methods and applications", Knowledge Engineering Review, vol. 11, 1996, pp. 93–155

[111] Vergara, J. E. L., Villagrá, V. A. and Berrocal, "Semantic Management: Advantages of Using an Ontology-Based Management Information Model", Proc. of the HP Openview University Association (HP-OVUA'2002), June 2002

[112] Vergara, J. E. L, Villagra, V. A. and Berrocal, J., "Application of OWL-S to define management interfaces based on Web Services", Proc. of the 8th IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS 2005), Lecture Notes in Computer Science, vol. 3754, October 2005, pp. 242-253

[113] Walker, A., "Knowledge systems: Principles and practice", IBM Journal of Research and Development, vol.30, January 1986, pp. 2-13

[114] Wang, J., Baclawski, K., Brady, D., Kokar, M. M. and Lechowicz, L., "The Use of Ontologies for the Self-Awareness of Communication Nodes", Proc. of the Software Defined Radio Technical Conference, November 2003

[115] Weatherspoon, H., Moscovitz, T. and Kubiatowicz., J., "Introspective Failure Analysis: Avoiding Correlated Failures in Peer-to-Peer Systems", Proc. of International Workshop on Reliable Peer-to-Peer Distributed Systems, October 2002

[116] Wilson, R. A. and Keil, F. C., "The MIT Encyclopedia of the Cognitive Sciences", MIT Press, 1999, ISBN 0-262-23200-6

[117] Yang, Y. and Calmet, J., “OntoBayes: An Ontology-Driven Uncertainty Model”, Proc. of the International Conference on Intelligent Agents, Web Technologies and Internet Commerce (IAWTIC’05), vol. 1, pp. 457-464

[118] Zadeh, L. A., "The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems", Fuzzy Sets and Systems, 1983, vol. 11, pp. 199-227

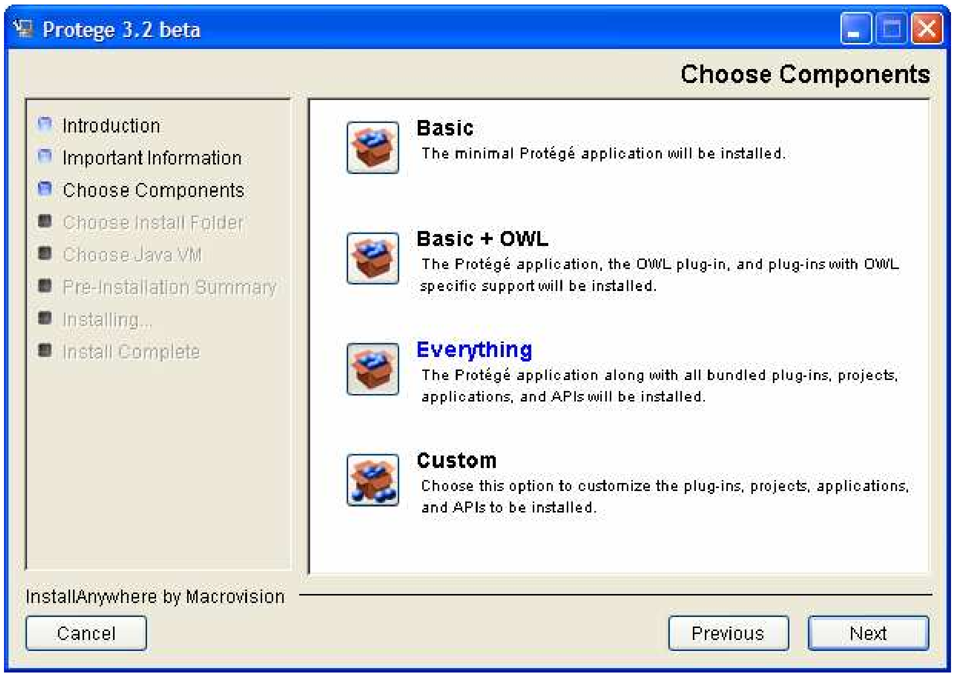
A Appendix

A.1 Ontology Browsing and Editing Related

In this section, explanations on how to open, browse and edit the ontologies created in this thesis are given.

A.1.1 How to Install Protégé

Protégé [43] is an open platform for ontology modelling and were used for all ontology development throughout this thesis. This section will explain how to install the editor. First, Protégé installation files need to be downloaded from its project site which is located at http://protege.stanford.edu/. In this thesis, the Protégé version 3.2 beta has been used, and in order to open the attached ontology files it is recommended that it only this or a later version is used. The editor is available both with and without the Java Virtual Machine. When going through the installation process, it is wise to control that "Everything" is chosen when asked which components that should be installed, as shown in Figure A-1.



Basic
The minimal Protégé application will be installed.

Basic + OWL
The Protégé application, the OWL plug-in, and plug-ins with OWL specific support will be installed.

Everything
The Protégé application along with all bundled plug-ins, projects, applications, and APIs will be installed.

Custom
Choose this option to customize the plug-ins, projects, applications, and APIs to be installed.

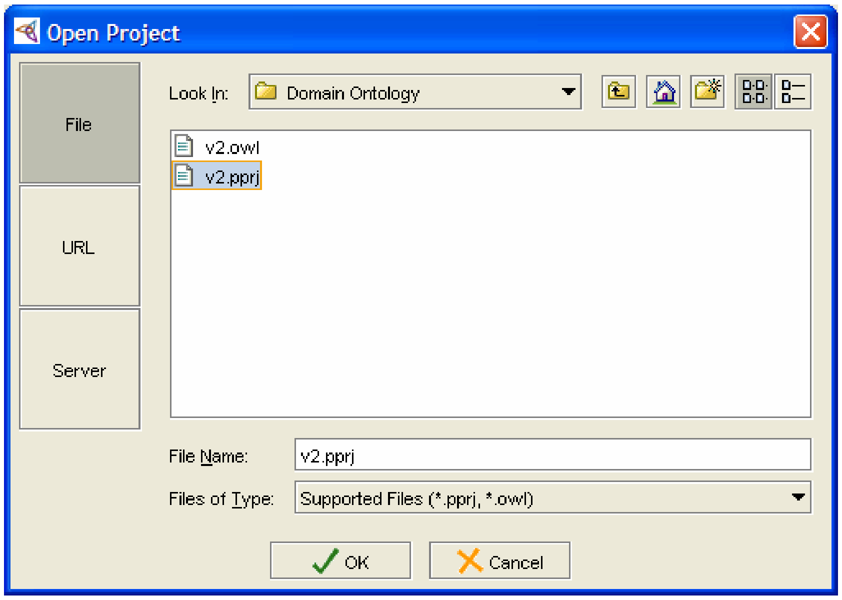
Figure A-1 – Choosing which components of Protégé that should be installed

When this has been done, the remainder is pretty straight forward and basically consists of choosing an appropriate installation directory.

In order to later use the SWRL editor and reasoning, the "jess.jar" file enclosed with this thesis must be inserted into the following Protégé sub-dir: "plugins\edu.stanford.smi.protegex.owl".

A.1.2 How to Open Ontology in Protégé

When starting Protégé, a welcome dialog is opened. This will present several options, including “Open Existing File”. Press this, and navigate your way to where you have extracted the attached file of this thesis. Here, you can easily navigate and open the desired ontology. Just remember to open the .pprj file as shown in Figure A-2.

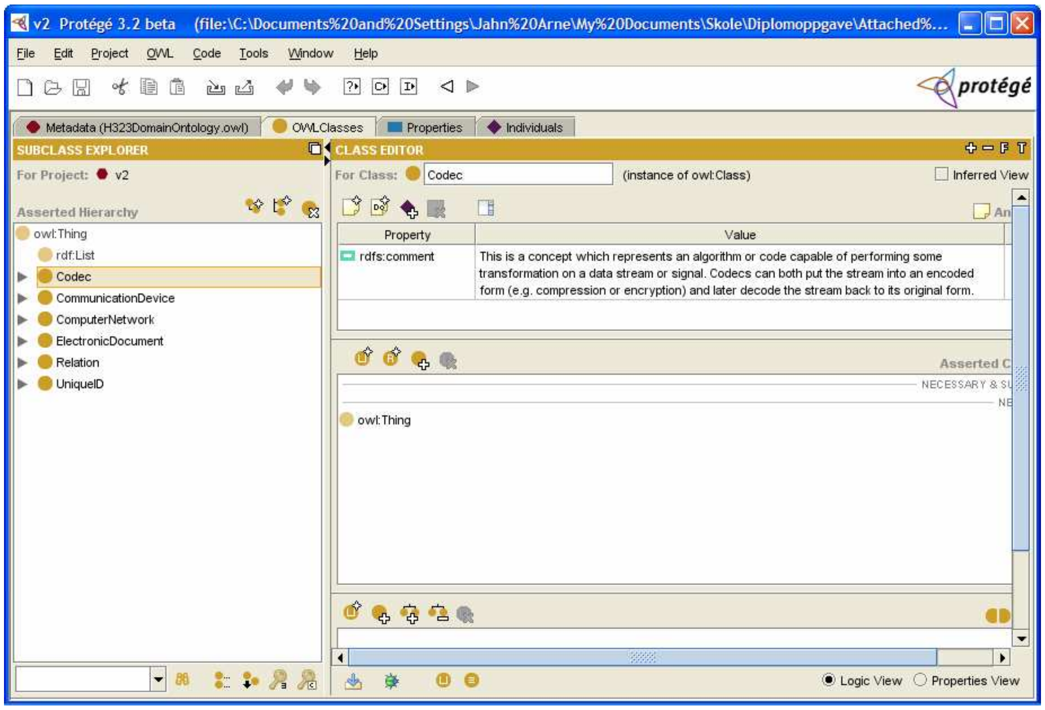


File Name: v2.pprj

Files of Type: Supported Files (*.pprj, *.owl)

Figure A-2 - Showing which ontology file to open

When this is done, the entire ontology should be shown as depicted in Figure A-3.



This is a concept which represents an algorithm or code capable of performing some transformation on a data stream or signal. Codecs can both put the stream into an encoded form (e.g. compression or encryption) and later decode the stream back to its original form.

Figure A-3 – Ontology opened in Protégé

A.1.3 How to Install Pellet OWL Reasoner

In this project, the open-source java based OWL DL reasoner, Pellet, has been used. It can be downloaded from its web site at http://www.mindswap.org/2003/pellet/download.shtml. This thesis has primarily used the 1.3 beta 2 version, but recommends the 1.3 version or later due to several bug fixes and better stability. The downloaded software is enclosed in a zip file which needs to be extracted to a suitable directory. After it has been extracted, it has to be integrated with Protégé. This is done as follows:

- Open the thesis project using Protégé as shown in A.1.2
- Using the menu, choose OWL → Preferences as shown in Figure A-4
- Make sure the “Reasoner URL” is set to “http://localhost:8081

After this has been done, the Pellet reasoner should be started by using the pellet-dig.bat file which lies in the Pellet directory. Now, everything should be in place to start using the Pellet reasoner. This can easily be confirmed by checking the consistency of the ontology using the “Check Consistency” – option under the OWL menu bar. See Figure A-4.

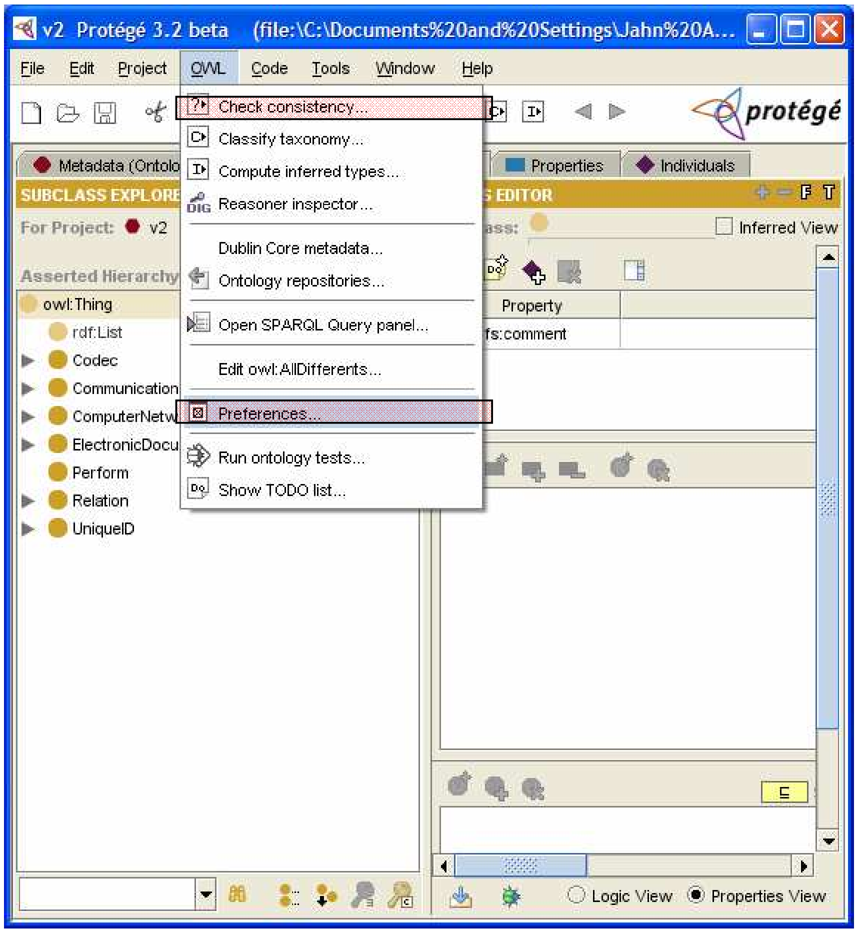
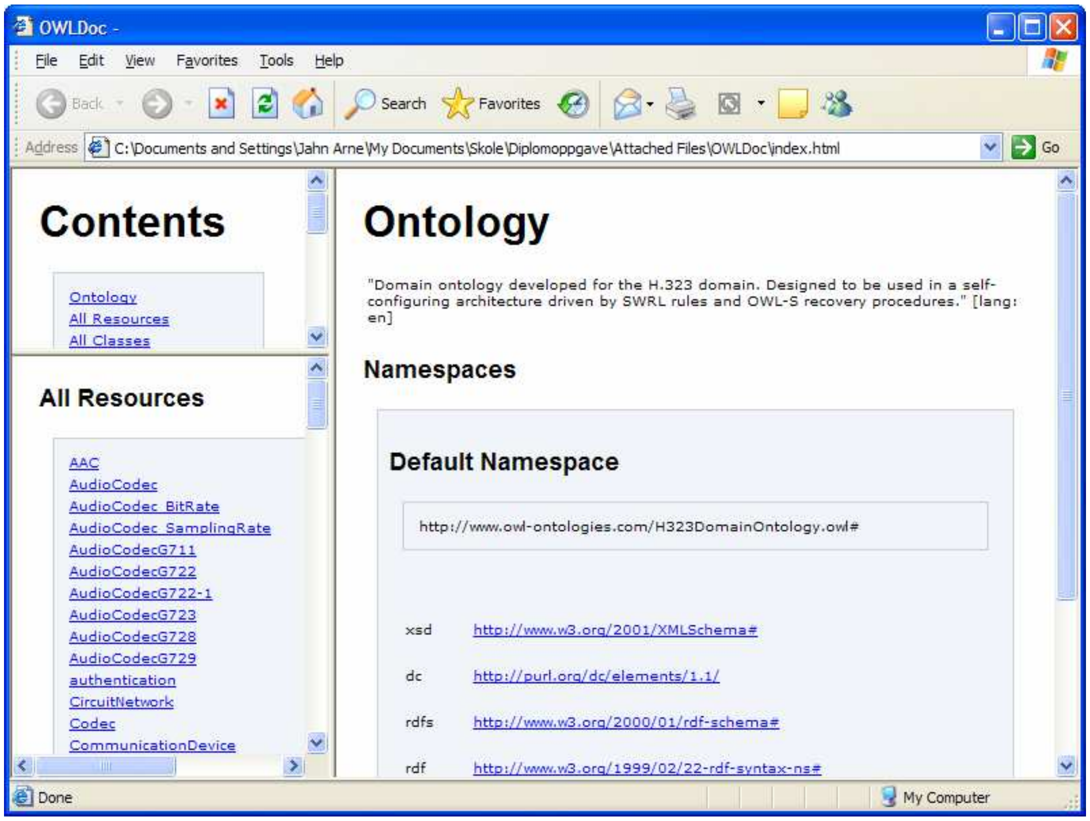


Figure A-4 – Navigating Protégé menus

A.1.4 How to Browse Ontology in OWLDoc

In the enclosed files of this thesis is an easy-to-browse version generated by OWLDoc. Its main file is index.html and can be opened by any HTML browser such as FireFox, Internet Explorer or Opera. When opened, it should look something like Figure A-5.


OWLDoc -
File Edit View Favorites Tools Help
Back Search Favorites
Address C:\Documents and Settings\Jahn Arne\My Documents\Skole\Diplomoppgave\Attached Files\OWLDoc\index.html Go
Done
My Computer

Contents

Ontology
All Resources
All Classes

All Resources

AAC
AudioCodec
AudioCodec_BitRate
AudioCodec_SamplingRate
AudioCodecG711
AudioCodecG722
AudioCodecG722-1
AudioCodecG723
AudioCodecG728
AudioCodecG729
authentication
CircuitNetwork
Codec
CommunicationDevice

Ontology

"Domain ontology developed for the H.323 domain. Designed to be used in a self-configuring architecture driven by SWRL rules and OWL-S recovery procedures." [lang: en]

Namespaces

Default Namespace

http://www.owl-ontologies.com/H323DomainOntology.owl#

xsd	http://www.w3.org/2001/XMLSchema#
dc	http://purl.org/dc/elements/1.1/
rdfs	http://www.w3.org/2000/01/rdf-schema#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#

Figure A-5 – Screenshot of web browser showing OWLDoc generated files

Included in this representation are all the instances used in the SWRL experiments of section 8.4.1, as well as the entire domain ontology and its relations.

A.2 Attached E-mails

This section contains only an e-mail received on the Protégé OWL mailing list, showing the author of the SWRLJessTab, software which allows users to write and execute SWRL rules using the open source Jess rule engine.

```
From: protégé-owl-bounce@crg-gw.Stanford.EDU on behalf of Martin
O'Connor [martin.oconnor@stanford.edu]
Sent: 19. juni 2006 21:20
To: protege-owl@SMI.Stanford.EDU
Subject: [protege-owl] Re: Reasoning with SWRL and Jess in Protege
3.2


Luis,


I looked at this and built-ins are not working property in the
current release. I am away most of this week at a conference so the
fix will probably not be available until next week's build.


Martin
```

Table A-1 – E-mail from the SWRL plugin developer regarding built-in functions

A.3 More on the H.323 Standard

This section will provide interested readers with some more details on important transactions, messages and codecs of the H.323 standard. Readers are further refereed to Brandl et al. ([15]) if the provided information is still not sufficient.

A.3.1 RAS Messages

Messages of the Registration Admission Status standard are generally presented on the form xRQ for requests, xRJ for rejects and xCF for messages confirming others. In , all known messages are presented alongside

Message	Name	Description
GRQ	Gatekeeper Request	Gatekeeper discovery message
RRQ	Registration Request	Registration of terminals, gateways and MCUs
URQ	Unregister Request	Unregistration of terminals, gateways and MCUs
ARQ	Admission Request	Request for admission to make a call
BRQ	Bandwidth Request	Request for more / less bandwidth
DRQ	Disengage Request	Disengage call
LRQ	Location Request	Request for contact information from a directory gatekeeper
IRQ	Info Request	Info request
IRR	Info Request Responce	Info responce
RIP	Request in Progress	Request in progress

Table A-2 – Messages used in the Registration Admission Status standard [15]

A.3.2 H.323 Message Sequence Charts

In this section, some message sequence charts of important H.323 transactions are presented.

A.3.2.1 Gatekeeper Discovery

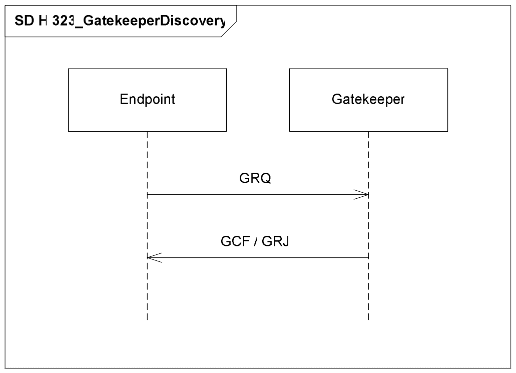


Figure A-6 – H.323 Gatekeeper discovery

When a H.323 endpoint needs to connect to a H.323 Gatekeeper, it sends out Gatekeeper Requests (GRQs). There are two possibilities for an endpoint to find its gatekeeper:

- **Multicast discovery:** The endpoint sends a Gatekeeper Request to a multicast address and port (224.0.1.41:1718). Gatekeepers receiving such requests may either confirm their responsibility with a GCF, or simply reject the request using GRJ.

- **Static configuration:** The endpoint may already know the IP address of the gatekeeper by manual configuration.

A.3.2.2 Registration

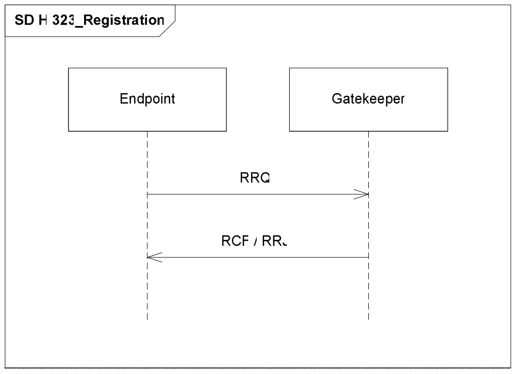


Figure A-7 – H.323 Endpoint Registration

After the endpoint has learned the address of a gatekeeper willing to accept its registration, it may send a registration request (RRQ) to the gatekeeper. This request may either be accepted (RCF) or rejected (RRJ).

A.3.2.3 Unregistration

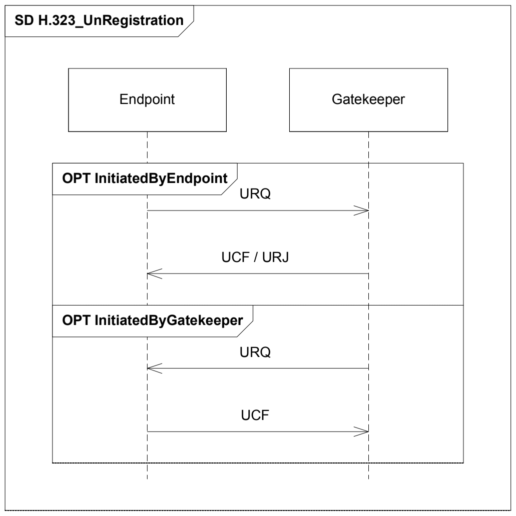


Figure A-8 – H.323 Endpoint unregistration

The unregistration procedure may either be initiated by the endpoint itself, or by the gatekeeper. This is specifically done by sending a unregistration request (URQ) to the other entity, which in turn can answer with a UCF or a URJ.

A.3.2.4 Direct Endpoint Call Signalling

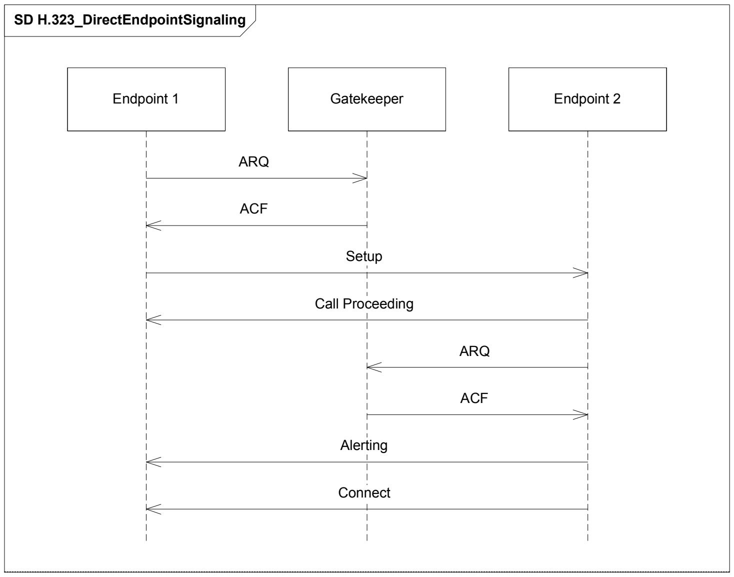


Figure A-9 - Direct Endpoint Call signalling

When an endpoint wants to place a call to another endpoint, this is can be done in several different ways. Depicted in Figure A-9 is the most common of these. Such procedures may vary in how the call signalling (Setup, Call proceeding, Alerting, Connect etc) is routed. In the shown procedure, all except the necessary RAS messages are exchanged directly between the endpoints.

A.3.2.5 Location Request

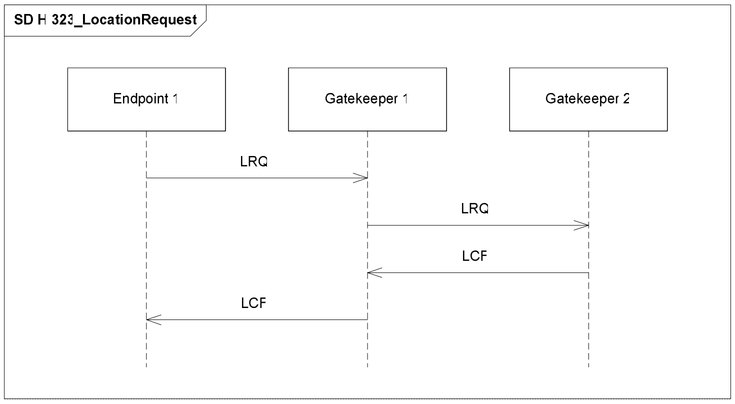


Figure A-10 – Location Request

If an endpoint or gatekeeper does not know the address of another endpoint, it may send out a location request (LRQ) to other gatekeepers which may know the endpoint. This request can be answered with either a location confirm (LCF) or location reject (LRJ) depending on gatekeeper knowing the endpoint or not.

A.3.3 Codecs

Audio codec	**Description**
G.711	8-bit compounded PCM (A-law or μ-law), 64kbp/s, 8kHz, required for H.323.
G.722	ADPCM audio encode/decode, 64kbit/s, 7kHz, optional for H.323
G.722.1	ADPCM audio encode/decode, 32kbit/s, 7kHz, optional for H.323
G.723	ADPCM, 6.3 and 5.3kbit/s, 8kHz, optional for H.323
G.728	LD-CELP, 16kbp/s, 8kHz, optional for H.323.
G.729	LD-CELP, 8kbp/s, 8kHz, optional for H.323
AAC	16, 32, 64, 96 or 128kbit/s, 8-96kHz
Security codec	**Description**
H.235	Security and Encryption for H.323 multimedia terminals
Video codec	**Description**
H.261	Supports 352x288 (CIF or FCIF) and 176x144 (QCIF). DCT-based algorithm tuned for 2B to 6B ISDN communication. Required for H.320, H.323, and H.324.
H.263	Much-improved derivative of H.261, tuned for POTS data rates. Mostly aimed at QCIF and Sub-QCIF (128x96 -- SQCIF), while providing better video than H.261 on QCIF and CIF. Optional for both H.320 and H.323.
H.264	Joint collaboration between the ITU and ISO. Improved video over H.263 providing similar quality at half the bandwidth.

Table A-3 – Overview of important audio, security and video codecs [15]