COMMUNICATING AGENTS OVER RESTRICTED ONTOLOGIES: A KNOWLEDGE SHARING APPROACH

A DISSERTATION SUBMITTED TO THE UNIVERSITY OF MANCHESTER FOR THE DEGREE OF MASTER OF SCIENCE IN THE FACULTY OF SCIENCE AND ENGINEERING

2016

Marco Aguilar Castro
School of Computer Science
Contents

Abstract .............................................................................................................................. 6
Declaration .......................................................................................................................... 7
Copyright ............................................................................................................................ 8
Acknowledgements........................................................................................................... 9
Chapter 1. Introduction ..................................................................................................10
Chapter 2. Ontologies and the tools for understanding, navigating and extracting knowledge. .........................................................................................................................15
  2.1 Ontologies ..............................................................................................................15
  2.2 OWL 2 and the OWL API ....................................................................................15
  2.3 Description Logic ..................................................................................................16
  2.4 LETHE ..................................................................................................................20
Chapter 3. Multi-agent systems and agent communication languages ..........22
  3.1 Software agents .....................................................................................................22
  3.2 AgentSpeak(L) and Jason Interpreter ................................................................22
  3.3 Agent’s Communication Language (ACL) ..........................................................24
  3.4 Alternative approaches to the communicating agents .........................................25
Chapter 4. The communication design and the knowledge services provision ......27
  4.1 The communicating agents approach ................................................................27
  4.2 Initializing the agent’s communication system ......................................................29
  4.3 Communication’s design: Ontology Query Service ..............................................31
  4.4 Communication’s design: Restricted view generation service ............................35
Chapter 5. The communicating agent’s design ..........................................................42
  5.1 An overview of the agent’s design ......................................................................42
  5.2 Agent Structure .....................................................................................................44
  5.3 Visualizing the Communicating Agent’s ............................................................49
  5.4 Prototype Updates ...............................................................................................51
Chapter 6. Evaluation and Results Analysis .................................................................53
  6.1 Selection of OWL Ontologies ..............................................................................53
6.2 Prototype functionalities and use cases evaluation............................55
6.3 Knowledge sharing experiments..................................................58
6.4 Ontology Extractions using LETHE...........................................67
Chapter 7. Conclusions.....................................................................73
References......................................................................................77
Appendix .........................................................................................79
   A. Project Gantt ............................................................................79
   B. Types of formulas used by AgentSpeak ....................................80
   C. List of symbols accepted by the Ground Term parser ..............81
   D. Signature size comparison for the extracted ontologies using LETHE...82

Word count: 18640
List of Tables

Table 6.1.1. Metrics of the ontology used for the Functionality Evaluation ...........53
Table 6.1.2. Metrics of the SNOMED modules .............................................54
Table 6.2.1. Functionality Test design .............................................................57
Table 6.3.1. Scenario 1: Experiment tests .......................................................62
Table 6.3.2. Signature comparison (Test 1.2) .................................................62
Table 6.3.3. Signature comparison (Test 1.4) .................................................63
Table 6.3.4. Scenario 2: Experiment tests .......................................................64
Table 6.3.5. Signature comparison (Tests 2.2, 2.4, 2.6 and 2.8) .................65
Table 6.3.6. Signature comparison (Tests 2.10, 2.12, 2.14, 2.16) ..............65
Table 6.3.7. Scenario 3: Experiment tests .......................................................66
Table 6.3.8. Signature comparison (Test 3.2) .................................................67
Table 6.4.1. Comparison between LETHE’s uniform interpolant methods ........68
Table D.1. Signature comparison between input symbols and LETHE interpolants..82
List of Figures

Figure 4.1.1. Ontologies interaction and their relation with agents. .........................28
Figure 4.1.2. Communication premise: assignment of users per agent. ...................29
Figure 4.2.1. Legend for the prototype services diagrams.....................................30
Figure 4.2.2. Initialization of the agents.................................................................31
Figure 4.3.1. Results shown to the user by the agent.............................................32
Figure 4.3.2. Use case 1 diagram: Query service – Single-agent............................33
Figure 4.3.3. Use case 2 diagram: Query service – Multi-agent..............................35
Figure 4.4.1. Use case 3 diagram: Restricted ontology service – Single-agent .......37
Figure 4.4.2. Use case 4 diagram part 1: Restricted ontology service – Multi-agent .................................................................40
Figure 4.4.3. Use case 4 diagram part 2: Restricted ontology service – Multi-agent. .................................................................41
Figure 5.2.1. Conceptual representation of the Knowledge Agent structure. .................44
Figure 5.3.1. Conceptual representation of communication between two agents. ..49
Figure 5.3.2. Conceptual representation of the multi-agent communication. ............50
Figure 5.3.3. General interaction between users and agents.................................50
Figure 6.2.1. Functionality evaluation cases..........................................................55
Figure 6.2.2. Additional functionality evaluation cases ...........................................56
Figure 6.3.1. Ontologies allocation for knowledge sharing experiment ......................58
Figure 6.3.2. Knowledge Sharing experiment scenarios........................................60
Figure 6.4.1. Signature size variability comparison...............................................71
Figure A.1. Project Gantt ..................................................................................79
Figure B.1. Types of formulas of AgentSpeak......................................................80
Abstract

The access to knowledge, its understanding and navigation, have increasingly become a necessity in both, research and industry; yet, it is usually pervasive and this represents a challenge. This document presents a standalone prototype of a multi-agent system that aims to solve the access to distributed knowledge, by assigning agents to each part of the knowledge and giving them the ability to share it with each other when required. These agents and the communication functionalities are modelled using Jason language and its multi-agent platform. The knowledge is represented and manipulated in OWL 2. The communication is aimed to be carried out by using a set of terms that are common to all the ontologies of the agents in the same multi-agent platform. LETHE allows these agents to create restricted views of their knowledge with respect to these common symbols. Therefore, based on this mutual knowledge, the agents are able to understand the semantics of the information received from other agents.

The presented Communicating Agents prototype, provides the user with two services: one for querying knowledge and another for creating restricted views of their knowledge. Also, these services are provided to the user in single-agent and multi-agent modes. The latter involves the communication and exchange of knowledge among agents with respect to the user’s demands.

There are presented a series of evaluations and experiments using real world ontologies. The results obtained guarantee a successful knowledge sharing when there is a set of symbols known by all agents. Added to this, and the success of all the prototype functionalities’ evaluations, a robust prototype is provided in this document. These results support a potential use of the Communicating Agents prototype to solve knowledge sharing problems.
Declaration

This dissertation is the candidate’s original work, and no portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.
Copyright

i. The author of this dissertation (including any appendices and/or schedules to this dissertation) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this dissertation, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has entered into. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the dissertation, for example graphs and tables (“Reproductions”), which may be described in this dissertation, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this dissertation, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy, in any relevant Dissertation restriction declarations deposited in the University Library, and The University Library’s regulations.
Acknowledgements

I would like to thank in first place to the Costa Rican Ministry of Science, Technology and Telecommunications, MICITT for the scholarship awarded to me from the “Fondo de Incentivos” to accomplish this MSc. degree.

Also, I would like to thank my supervisor Dr.-Ing. Renate Schmidt of the School of Computer Science. Her guidance and feedback was always useful and on time. She was permanently available for any question about my research. Also, her enthusiasm and constant motivation towards a development of an innovative and challenging project gave me positivism and perseverance to accomplish this project successfully.

My gratitude also, to Dr. Patrick Koopman for providing me with the access to the LETHE library and solving any questions related; to Dr. Rafael Heitor Bordini and Dr. Jomi Fred Hübner for their availability and opportune answers to my question about AgentSpeak and Jason Interpreter.

Thanks also to Nicolas Matentzoglu who always helped me with answers to my questions about the OWL API; and to Yongsheng Gao for providing the access to and guidance through the use of the SNOMED ontology and the modularity plug-in.

Finally, I would like to express my gratitude to my partner, Susana, and my family for giving me their support, valuable insights and encouragement along this project.

Eng. Marco Aguilar Castro
Chapter 1. Introduction

The present document provides an application of ontology manipulation tools, description logic and multi-agent systems to create a standalone prototype where agents, by means of specific tools such as LETHE, the OWL API and the Jason Interpreter, are able to process and share knowledge with each other using the common terms in their ontologies.

The developed artefact hinges on a framework where knowledge is segmented into partially overlapping portions, i.e. knowledge representations with intersections among them. For example, the processes of a company, even when each of them has starting and ending points, specific inputs, a sequence of activities and expected outputs; the knowledge required to perform each process, as narrow or broad as it can be, is not independent from the knowledge required to execute previous, subsequent or even parallel processes. Therefore, this intersection or common knowledge, can be used as the language for sharing the remaining portions of “know-how” in a way that is understood by all the agents involved.

The role of ontologies and agents in knowledge sharing, with respect to the scope of the developed artefact, aims to simplify the laborious task of searching for the pieces of facts, instructions, learnings and other wisdoms required to assemble the representation of the targeted knowledge domain. Ontologies provide an approach where facts, learnings and—generally speaking—data, can be compiled, linked and represented as a knowledge conceptualization for further use. This use can be either simple queries, or complex inferences to make explicit the implicit relations and notions contained in the ontology.

Agents, by their side, provide a simple yet efficient interaction with the user, while executing almost autonomously queries and generating representations of requested knowledge. Furthermore, agents in this same self-directed fashion, can collect the related knowledge from different domains in a collaborative way and provide a comprehensive or multi-agent depiction of it, enabling the user to access it in a way that otherwise would have been inaccessible. The Communicating Agents prototype presented in this
document, makes use of the Jason agent language, and its platform to construct the multi-agent system.

A small number of approaches have combined OWL ontologies and agents, but it has been for different purposes than the one presented in this document. The use of description logic’s expressive languages (such as \( \mathcal{ALC} \)), that can represent ontologies (e.g. OWL 2\(^1\)), and the tools developed for manipulating them (the OWL API and LETHE), open a range of possibilities for knowledge sharing in an efficient way and also provide a powerful framework for managing knowledge across several domains.

This synergy between knowledge tools and agents, enables meaningful uses of ontologies, that not only allows the user and the agents to navigate ontologies, but also permits them to visualize and understand the knowledge domain in a more detailed and customized manner, by making use of collaborative tools like the developed in this document.

The stages of this project are conceived in the following objectives, which provide a coherent sequence of activities (see Appendix A for the project’s Gantt), where the prototype is not only modelled and built, but is also evaluated under different scenarios. The following bullet points describe each objective:

- Develop a Java based standalone prototype for communicating agents using OWL ontologies. This development includes the integration of the functionalities provided by the OWL API libraries for understanding, navigating and extracting information from ontologies, as well as the development of the communication capabilities with other agents.

- Integrate into the agents’ functionalities the LETHE tool package for uniform interpolation, to provide the required abilities for extracting specific symbols in the ontologies with respect to a common language, so it can be understood by all the agents involved in the communication.

- Develop the standalone prototype where the communicating agents will carry out the exchange. It will provide the platform where the agents can

\(^1\) Web Ontology Language
find each other, establish a communication and transfer the required knowledge.

- Select the experiment’s OWL ontologies, according to the complexity settings of each case and the compatibility restrictions of the used tools (the OWL API and LETHE). These ontologies will be selected from renowned repositories such as those from The University of Manchester, the University of Oxford and SNOMED-CT.

- Run the complete prototype over all the different cases of the evaluation, where the agents’ functionalities and the implemented packages (interfacing with the OWL API and LETHE) are fully tested with regard to the communication output accuracy and the selected ontologies for each testing scenario.

- Analyse the obtained results from each case, to identify how and under which parameters the use of the LETHE tool helped to achieve successful communications, as well as to detect required improvements to the prototype that will lead to the optimization of the prototype’s functionalities.

- Design and execute a human-agent interaction case as an additional experiment, based on the obtained results from the experiment stage, in order to maintain a coherent analysis by applying it only to those cases relevant to the project’s scope.

- Update the prototype with the most relevant improvements according to the analysis of the results from the evaluations and experiments; with the goal of guaranteeing the delivery of not only a tested prototype, but also one with optimized functionalities.

The document presents all the stages required to develop, test and improve the Communication Agents prototype. First, the technical background is covered in Chapters 2 and 3, while the methodology followed to develop the
 prototype is described in the next two chapters, Chapters 4 and 5. Next, Chapter 6 present the evaluation design, results and discussion of them. Finally, the conclusions, limitations and further research are presented in Chapter 7.

The definition of an ontology as a knowledge representation is presented in the first section of Chapter 2. Next, OWL 2 is defined along with its relation with the OWL API and the ontology management capabilities. Section 2.3 present the notions of Description Logic used by the prototype. The reasoning and benchmark tools are explained after this. The expressiveness of the ontologies used in this project and their technical definitions and properties are defined as well. A modularity review is also included given its use in the experimentation stages. Finally, LETHE and its properties is addressed in the final section of this chapter.

Chapter 3 makes use of agent theory to describe the concept of autonomous software agents, its capabilities and characteristics, from a work planning approach, as entities that have beliefs, desires and intentions. On the Section 3.2, is described the agent programming language, its architecture and functionalities. Next section covers the agent’s communication language and the main methods used for knowledge sharing. Finally, Section 3.4 enumerates alternative approaches to the use of agents and ontologies and how they are similar, or not, to the proposed approach.

The methodology Chapters 4 and 5 present the communication’s design and the agent’s design, respectively. The former is addressed in Chapter 4, it covers the communication among agents and how it enables them to share their knowledge by using sets of common symbols. Section 4.1 gives a general description of the knowledge sharing approach, while Sections 4.2, 4.3 and 4.4 respectively, describe the initialization of the communication, and the two services available to the user for accessing and sharing knowledge. These services are built into the agents’ capabilities.

The design of the prototype is provided in Chapter 5, where a general description is given first to understand the overall architecture of the agents. Then, Section 5.2 presents all the technical details of the prototype itself, why and how each module is constructed and how they interact in order to provide the agent with the necessary functionalities to share knowledge using the
services described in Chapter 4. Section 5.3 describes in detail how the agent sends and receives information, in order to enable the agents’ communication capabilities. Finally, and in attention to the last objective of the present project, Section 5.4 presents all the improvements done to the prototype that were discovered on the evaluation stage.

Chapter 6 contains in Section 6.1 an explanation of how the ontologies were selected and prepared in order to be used sensibly in the evaluation and experimentation of the prototype. The evaluation of the prototype presented in Section 6.2 and covers the evaluation of the functionalities and use cases described in Chapter 4. In the next section, 6.3, the knowledge sharing experiment is developed in three scenarios in order to test the knowledge sharing capabilities of the Communicating Agents prototype and identify the cases where it performs better. Section 6.4, analyses and discusses the performance of LETHE and the related findings from all the evaluations and experiments.

Finally, the conclusions and future work are presented in Chapter 7, the objectives and findings from the evaluation and experimentation stage are contrasted here. It also revisits the relevant highlights regarding the prototype’s development and usage. This chapter concludes with a discussion of the new applications of the prototype, future work and improvements to it, as well as additional experiments and extensions those presented in Chapter 6.
Chapter 2. Ontologies and the tools for understanding, navigating and extracting knowledge.

2.1 Ontologies

According to Gruber (1993), to represent some knowledge it is required to construct a conceptualization of it, that is to build a specific abstract notion of the objects in the world that are required to be represented. Gruber (1993) states that an ontology is a specific and abstract view of the world described in an explicit manner. To this definition Uschold and Gruninger (1996) add that it is a shared understanding of this specific view, which enables the usefulness of an ontology for this project, as it provides a standardized instrument to understand the real world concepts to be represented. Finally, the OWL Working Group defines an ontology as follows:

"An ontology is a set of precise descriptive statements about some part of the world." (OWL Working Group, 2012a, p. 3)

Therefore, throughout this document there will also be used the term knowledge representation to refer to an ontology. Once defined what is an ontology, the importance for knowledge sharing is that it provides a meaning for each of the objects (concrete or abstract) represented, in an unambiguous fashion (Uschold & Gruninger, 1996). So, no matter who or what interprets them, the meaning is always the same and cannot be confused with other term’s definition. This characteristic is also addressed by the OWL Working Group (2012a).

2.2 OWL 2 and the OWL API

Now, ontologies need a language or standard to be represented, in this prototype, ontologies use OWL 2. The OWL Working Group (2012a) refers to OWL 2 as a declarative language used to express ontologies. It is formally defined as follows:

"OWL is a computational logic-based language such that knowledge expressed in OWL can be reasoned with by computer
Finally, a specific syntax should be selected for the user’s inputs. This project uses for the user’s interface the OWL Manchester Syntax. It is a frame-based syntax for OWL 2 descriptions (OWL Working Group, 2012b) that uses natural language and an infix notation to define a vocabulary for the DL symbols, restrictions and even a complete OWL ontology (Horridge M., et al., 2006). Applications like Protégé uses it for user’s inputs, queries and manipulation of symbols and axioms.

Horridge and Bechhofer (2011) provide a comprehensive Java-based API to manage ontologies. An ontology in the OWL API is represented by a set of annotations and axioms. The OWL API is compliant with the W3C OWL 2 specifications. Among its most relevant functionalities for this project, there can be mentioned the following from Horridge and Bechhofer (2011). First, entities can be asserted by different ontologies. Second, the reasoner is given the possibility to be initialised, therefore, changes in the ontology can be assigned to a buffer for later update or processed immediately, which have an impact on the resources used to load and process an ontology.

2.3 Description Logic
For this project, the OWL ontologies use direct semantics, which signifies that it uses a description logic to assign a meaning (i.e. semantics) to OWL 2 ontologies (OWL Working Group, 2012a). Therefore, it is important to introduce what a description logic (DL) is, given that from this part onwards, the references to this type of ontologies will be intertwined with DL notions. According to Baader et al., (2003) define description logic is a decidable fragment of First Order Logic that can be defined as a knowledge representation formalism. The DL languages are defined by two types of predicates: unary and binary. Description Logics consider both, the knowledge structure and the reasoning services (Baader, et al., 2003).

OWL 2 makes use of the term entities to refer to classes, properties (roles) and instances (OWL Working Group, 2012a). For this project there are not used role symbols. Concept symbols (classes) are those atomic concepts or
Unary predicates that represent individuals or classes (Baader, et al., 2003). There are several types of axioms present in an OWL 2 ontology, for this project, the two most important are: subclass or concept inclusion and equivalent class or concept equivalence. Then, according to Baader et al. (2003), for two concepts $C$ and $D$, these two axioms can be represented respectively by equations 1 and 2 below:

$$C \sqsubseteq D$$  \hspace{1cm} (1)  

$$C \equiv D$$ \hspace{1cm} (2) 

Like others symbols, concepts in an ontology can have constant terms or instances assigned. Then, a concept $C$ that belongs to an ontology $O$, and a constant term or instance $b$ of the concept $C$, can be expressed as:

$$O \models b : C$$ \hspace{1cm} (3)

and then according to Baader et al. (2003),

$$C(b)$$ \hspace{1cm} (4)

In the context of this project, there are considered as useful ontologies, those who are consistent and whose concepts are satisfiable. Based on the definition given for both by Ben-Ari (2012), a satisfiable formula or axiom is true in some interpretation. On the other hand, an ontology is considered to be consistent when there cannot be proved that the ontology’s axioms and their negations exist in the same ontology, or in other words, there cannot be proved that the ontology is self-contradictory. Therefore, given an ontology $O$ and any axiom in the ontology $A$, the next two equations must be true:

$$O \not\models A \sqsubseteq \bot$$ \hspace{1cm} (5)

$$O \not\models T \sqsubseteq \bot$$ \hspace{1cm} (6)

**Reasoning, HermiT and DL Queries**

However, an ontology on its own is only a representation, a document that has no autonomy or cannot extract or infer meaning over its content, that is why is required a reasoner in order to obtain inferences from the knowledge contained in an ontology. Dentler et al. (2011) define a reasoner as a program that from the assertions made by the ontology, infers logical consequences.
So, consistently with the OWL 2 definition given above, these programs allow to explicitly obtain implicit knowledge embedded in the ontology. Among the tasks performed by a reasoner can be mentioned: classification, debugging and querying (Dentler, et al., 2011).

The prototype uses HermiT (Glimm, et al., 2014), a reasoner compliant with OWL 2 Direct Semantics, which for general understanding is defined by its authors as a hypertableau based reasoner. The HermiT libraries are compatible with the OWL API, builds over it to perform its functionalities as a reasoner.

To benchmark the results from the evaluations and experiments there is used the DL Query embedded in Protégé 4.3 which perform searches over a classified ontology. This is a functionality provided to avoid creating named classes in order to check if specific axioms belong or not to the ontology and evaluate if the correct or expected subclasses are subsumed by a specific class definition (Horridge & Drummond, 2016). These DL Query uses also HermiT as the reasoner providing the functionalities required.

**Expressiveness Language**

Description logic have different languages. They are used by OWL 2 ontologies to enrich the semantics of the symbols used (expressiveness), yet, it has a cost, measured in the amount of computational resources required by the reasoner when making inferences from the knowledge represented. In this project the focus is set on some members of the $\mathcal{AL}$-languages.

First, the basic $\mathcal{AL}$ includes atomic concepts, negated atomic concepts, the universal concept or Top, the bottom concept, the intersection symbol, value restrictions, and limited existential quantification; respectively denoted by $\{C, \neg C, T, \bot, \cap, \forall R.C, \exists R.T\}$, where $C$ is a concept symbol and $R$ is a role symbol (Baader, et al., 2003).

Now, in order to understand the extensions to the $\mathcal{AL}$-languages some notions are presented. The first is the domain of discourse, denoted by $\Delta^I$, and contains the elements to be represented (Baader, et al., 2003). The next notion is the interpretation, it maps atomic formulas to truth values (Ben-Ari, 2012).
This project deals with the following extensions: $\mathcal{C}$, $\mathcal{E}$, $\mathcal{R}$ and $\mathcal{H}$. In description logic, $\mathcal{C}$ represents the use of negation of arbitrary concepts, given a class $\mathcal{C}$, then this means using $\neg \mathcal{C}$, therefore, $\mathcal{ALC}$ restricts to two variables the formulas of the first order logic fragment represented (Baader, et al., 2003). Now, the same authors define $\mathcal{C}$ as follows:

$$(\neg \mathcal{C})^I = (\Delta)^I \backslash (\mathcal{C})^I$$  \hspace{1cm} (7)

The full existential quantification is denoted by $\mathcal{E}$ and Baader et al. (2003) define it as $\exists R. \mathcal{C}$, where $R$ is a role symbol and $\mathcal{C}$ is a concept symbol, this is defined by the same authors as follows:

$$(\exists R. \mathcal{C})^I = \{ a \in \Delta^I | \exists b. (a, b) \in R^I \land b \in \mathcal{C}^I \}$$  \hspace{1cm} (8)

The extension $\mathcal{R}$ defines a role inclusion. Baader et al. (2003) define it as below for to roles $R$ and $S$:

$$R \sqsubseteq S$$  \hspace{1cm} (9)

Finally, the role hierarchy extension $\mathcal{H}$, according to Baader et al. (2003), represents a set of role inclusions such as the one shown in equation 9.

**Syntactic-Locality-Based Modularity**

For the experiments presented in this document, a pre-processing of certain ontologies is required, the approach followed to accomplish this, is the extraction of modules from an ontology.

Horridge et al. (2014) define that for an ontology $\mathcal{O}$, and a signature $\Sigma$ where $\Sigma \subseteq \text{sig}(\mathcal{O})$, is possible to extract a syntactic-locality-based module $\mathcal{M}$ from $\mathcal{O}$ using $\Sigma$. It is extracted by inspecting the syntax axioms in $\mathcal{O}$ and is define by these authors as:

$$\mathcal{M} = \text{Module}(\mathcal{O}, \Sigma) \subseteq \mathcal{O}$$  \hspace{1cm} (10)

The usefulness of this type of modules for this project is that a module $\mathcal{M}$ behaves like $\mathcal{O}$ for an axiom $\alpha$ that uses $\Sigma$ (Horridge et al., 2014). Therefore, as define by these same authors:

$$\mathcal{M} \models \alpha \iff \mathcal{O} \models \alpha$$  \hspace{1cm} (11)
Among the types of syntactic-locality-based modules this project only considers three: TOP (T-module), BOT (⊥-module) and STAR (T⊥*-module) (Horridge, et al., 2014). These require and ontology \( \mathcal{O} \) and a seed signature \( \Sigma \) (Del Vescovo, et al., 2010). The TOP and BOT modules can be approximated by modules based on empty set (\( \emptyset \)-locality) or full domain (\( \Delta \)-locality), while the STAR can be obtained by nesting the previous two (Sattler, et al., 2009). Finally, Horridge et al. (2014) summarizes TOP as containing the axioms of the relations between \( \Sigma \) and more specific terms of \( \mathcal{O} \) and BOT as containing the axioms for relations between \( \Sigma \) and more general terms of \( \mathcal{O} \), while STAR is composed by the axioms of the relationships among terms in \( \Sigma \).

2.4 LETHE

This tool performs non-standard reasoning tasks (Koopmann & Schmidt, 2015a). The saturation-based reasoning used by LETHE have properties such as logical completeness that other approaches lack (Koopmann & Schmidt, 2015a). From the functionalities provided by LETHE, the most relevant for the prototype are those for performing forgetting and uniform interpolations. These provide the means by which the restricted ontologies are obtained in the prototype. Thus, enabling the sharing of the knowledge among agents with the guarantee of common understanding by both, the agent that sends the information, and the agent who receives the information, while avoiding ambiguities as well.

According to Koopmann and Schmidt (2015b), when using forgetting on an ontology \( \mathcal{O} \), LETHE permits the removal of unnecessary, private or irrelevant concept and role symbols from it. However, it conserves the entailments or logical consequences from the complete ontology. Then, Koopmann and Schmidt (2015b) define that for a set of symbols \( \Sigma \), an ontology \( \mathcal{O} \) and a logic \( \mathcal{L} \), the result of forgetting \( \Sigma \) is \( \mathcal{O}^{-\Sigma} \), where

\[
sig(\mathcal{O}^{-\Sigma}) = \sig(\mathcal{O}) \setminus \Sigma
\]

and for all axioms \( \alpha \) in \( \mathcal{L} \),

\[
\sig(\alpha) \cap \Sigma = \emptyset
\]

\( \mathcal{O}^{-\Sigma} \models \alpha \) iff \( \mathcal{O} \models \alpha \)
The prototype makes use of LETHE under the $\mathcal{ALC}$ language of Description Logics. For this project only the T-Box is considered in the LETHE tasks. It is important to highlight that $\mathcal{ALC}$ reasoning involving T-Boxes is $\text{EXPTIME}$–complete (Baader, et al., 2003) and the decidability of the uniform interpolation is $2\text{-EXPTIME}$ (Koopman & Schmidt, 2014). In addition, the use of LETHE with $\mathcal{ALC}$ is interpolation complete, therefore, it is guaranteed that the reasoning algorithm goes through all the inferences required to forget the requested symbols (Koopmann & Schmidt, 2015a). LETHE can be used with more expressive languages such as $\mathcal{ALCH}$ (Koopmann & Schmidt, 2013) and $\mathcal{SHQ}$ (Koopman & Schmidt, 2014).

As mentioned by Koopmann and Schmidt (2015b), the uniform interpolants generated by LETHE have several applications, among them these same authors enumerate the computation of logical differences, segmentation of an ontology to perform a specific analysis, the concealment of information accessed by several user and the reuse of specific sets of symbols for a specific purpose. These applications are important in the context of the present prototype, since they enable the agents to extract sections of an ontology and then apply them as described above.

Koopmann and Schmidt (2015c) have developed in Java a series of libraries with LETHE functionalities in compliance with OWL 2 and OWL API 3.5.2. Thus, the prototype uses the two main methods provided for uniform interpolation and forgetting.
Chapter 3. Multi-agent systems and agent communication languages

3.1 Software agents
According to Shoham (1993), the meaning of the word agent has changed from its initial context to a more specific one given its use in computer science. First, an agent was some person acting in representation of another; now, an agent, physical or software, is an entity that operates with autonomy and in a continuous fashion, normally other agents and processes take place at the same time (Shoham, 1993). Among the most relevant characteristics of an agent, Wooldridge and Jennings (1995) and Shoham (1993) enumerate knowledge, belief, capabilities and choice. Also, Wooldridge and Jennings (1995) mention that agents can be proactive or reactive in the way they execute their plans and achieve their goals.

Communication in multi-agent systems can be carried out directly or indirectly (Genesereth & Ketchpel, 1994), that is to say, with another program intervening in the delivery or processing of the message (direct) or without it (indirect). This project makes use of AgentSpeak (Bordini & Hübner, 2006) as the language for programming the multi-agent system.

3.2 AgentSpeak(L) and Jason Interpreter
The programming language used for the agents is defined as follows:

“AgentSpeak(L) is a programming language based on a restricted first order language with events and actions.” (Rao, 1996)

The agent-oriented programming approach followed by AgentSpeak(L) is based on a reactive planning system, with a Belief-Desire-Intention (BDI) architecture (Bordini, et al., 2007).

Beliefs are the facts of the world representation that are known by the agent (Bordini, et al., 2007). According to Bordini and Hübner (2006) agents require a belief base where each initial belief is a ground atomic formula.

beliefName(constantValue1, ...)
The intentions are the goals the agent have, and this will drive it to select a specific plan (Bordini, et al., 2007). There are two types of goals: achievement (\(!\text{goal}\)) to reach a state in which the atomic formula is true; and test (\(?\text{goal}\)) to assert if an atomic formula is an agent’s belief.

\[
!\text{goal}(\text{constantValue1}, ...) 
\]

The desires are the set of plans the agent is able to execute (Bordini, et al., 2007). These plans can be added or subtracted (also for dealing with plan errors). Bordini et al., (2007) define three types: belief addition/deletion, achievement-goal addition/deletion and test-goal addition/deletion. Plans are composed of three parts:

\[
\text{triggering\_event} : \text{context} \leftarrow \text{body} 
\]

The body is an inseparable set of actions (atomic formulas). The triggering event define the goal required to initiate the actions in the body, while the context define the beliefs required to perform these actions (Bordini & Hübner, 2006). The body is executed only if the triggering event and the context restrictions are met.

Now, Jason is a Java-based interpreter that implements an extension of AgentSpeak(L), but now, atomic formulas, lists, integers and strings are considered terms. Terms can be variables and bound and unbound variables are treated as literals. Also plan contexts can use relational operators in its declarations (Bordini & Hübner, 2006). Annotations are included and can be used to trace the source of the terms. Appendix B illustrates how these terms relate with each other.

The agents are created in a .asl file where using the Jason Interpreter syntax, all the beliefs, desires and intentions are explicitly written. Each file represents a type of agent, but the multi-agent system contains instances of these types of agents (Bordini, et al., 2007).

Jason already have a set of internal actions for standard purposes, that can be used in the context or in the body of a plan (Bordini, et al., 2007). Also, customized internal actions can be developed as Java classes to add specific functionalities to the agents, however, contrary to the programming convention, the class name should be in lowercase (Bordini, et al., 2007). The
Jason API provide the functionalities of Jason Interpreter to develop the customized actions. This API can be used to develop a multi-agent project.

It is important to comply with the syntax for terms. Any constant value assigned to a belief, must comply with the conventions for constants: first char must be lowercase and a letter, the rest of the characters must be alphanumerical, with no spaces and use camelCase to distinguish words. Variables follow the same restrictions except that the first char must be uppercase.

According to Bordini and Hübner (2006), Jason provides a platform for setting the desired environment where the agents interact. The Java class containing the characteristics of the environment is referred in this platform. The Jason API can also be used to customize the platform and the environment. Among the relevant multi-agent infrastructures, there is offered a centralized option, for one machine; and the SACI option, for networks where agents are not in the same machine. This platform also allows to set the type of agents in the multi-agent system and how many instances of each are created. The source path for getting the agents’ .asl file and the class paths to link the platform with external libraries, as well as the execution control of the agents are set here also. This last setting allows the agents to execute their tasks synchronous or asynchronous (Bordini, et al., 2007).

3.3 Agent’s Communication Language (ACL)

Wooldridge and Jennings (1995) describe an agent language as a system that provides the means to program the functionalities and characteristics of an agent (knowledge, belief, capabilities and choice), no matter if it is a software or a physical agent. However, Gruber (1993) warns about the independency required between the agent communication language and the shared symbols from the knowledge representation. Also, Pazienza and Vindigni (2005) mention communication complexities such as not including the words that the recipient agent uses, ignoring the meaning of words used in the communication, or assigning several meanings to the same word. A careful and accurate use of the set of common symbols and the uniform interpolants can help deal with these potential issues. According to Genesereth and
Ketchpel (1994), an agent communication language (ACL) consists of 3 parts: vocabulary, an inner language and an outer language, this is covered below.

The approach followed by AgentSpeak and Jason for the communication among agents is based on speech-act theory, which make it a knowledge-level communication rather than a method invocation such as those used by programming languages, e.g. Java (Bordini, et al., 2007). These authors enumerate two types of statements, the illocutionary force (type of statement) e.g. inform, suggest, etc.; and the perlocutionary force (intention of the statement). Bordini et al., (2007) addressed that in the agent communication scenario, these classifications of speech-act are called performatives and they expand in more detail the two previously defined.

Jason uses two internal actions to perform the communications among agents, one sends a specific message to a specific agent and the other sends the same message to all agents (Bordini, et al., 2007), below is defined the code:

```
  . send(receiver_agent, performative, propositional_content)
  . broadcast(performative, propositional_content)
```

The performatives are divided in four areas: information exchange (beliefs sharing), goal delegation (asks to execute a specific plan), information seeking (asks about other agent’s beliefs) and know-how related (exchange of plans). These performatives are based on KQML² concept and semantics (Bordini, et al., 2007). The FIPA standard is similar, but is not used by AgentSpeak. The Jason .send and .broadcast internal actions can be used and changed from within another customized internal action, but then the default plan +!kqml_received must be added to the desires of the agents required to receive and understand these customized messages.

3.4 Alternative approaches to the communicating agents

As far as this research is concerned, there are no current publications for approaches to communicating agents using uniform interpolation or LETHE to exchange knowledge with each other. However, there are alternative

---
² Knowledge Query and Manipulation Language.
approaches for the use of communicating agents and ontologies, although for different purposes.

Gruber (1993) mentions the use of ontologies to standardize the agents’ commitments from a knowledge-level approach, i.e. there is a unique ontology that provides the terms required by the agents to communicate with and understand each other, even when these terms are technical.

Racer is a reasoning agent which in a reactive fashion executes the query provided and infers the knowledge from an OWL ontology. Applications for this reasoner agent are mentioned by Haarslev and Möller (2003) with an approach to information retrieval applications in the semantic web, working in a multi-agent environment and using A-boxes to represent the document’s information.

Research from Tamma et al. (2005) uses a multi-agent architecture with agents arranged in a network that resembles an ontology, this allows the efficient retrieval of knowledge annotated according to its semantics.

In an approach presented by Soh (2005), agents query ontologies to extract associated URLs, but no OWL 2 language or reasoners are used.

Laera et al. (2006) use an argumentation framework between agents to define the symbols and their meaning in order to establish a coherent communication with each other.

Finally, Doran et al. (2010) describe an approach of communication between agents making use of ontology mappings and argumentation based negotiation to define the meaning of the communication symbols used.
Chapter 4. The communication design and the knowledge services provision

4.1 The communicating agents approach

The developed artefact is a standalone prototype of a set of agents programmed in Java. These agents are capable of communicating with each other by creating a communication system of agents in Jason. This system is able to provide the following two services to the users: the ability to query the knowledge embedded in an ontology, and the possibility of requesting a restricted view of an ontology. Also, according to the user’s demand, the agents can communicate with each other and request these services from each other. For example, if the user requests a restricted view of an ontology, then the user’s agent will provide one based on its ontology, but the agent can also ask another agent to provide a restricted view based on their ontologies. Then, the user’s agent will make available to the user all the consistent views and whose concept axioms are satisfiable. The query and the restricted view services with its single and multi-agent variants, are described in detail in Sections 4.3 and 4.4.

The prototype uses only one type of agent, a knowledge agent. All the instances of this type of agent are allowed to communicate with each other when sharing the same network, this is achieved by using a Jason platform. This agent type contains all the functionalities to provide the services mentioned previously.

In order for the knowledge agents to be capable of working with the represented knowledge, the ontologies used the OWL 2 language and the agents are capable of manipulating them by making use of the OWL API and the LETHE library which provide the core methods to design the functionalities that enable them to provide the services requested by the users.

The knowledge agents carry out the information exchange in their domain with users or other agents. This knowledge is expressed in terms of concept and role symbols and exchanged based on the common symbols with each other. An example of a manufacturing industry is given to explain how the artefact works:
A company with 4 departments: Manufacturing, Sales, Marketing and R&D. The relation among them can be explained in general terms as follows: Marketing identifies the need of a new product with specific characteristics. Then, R&D is in charge of developing this new product. Next, Manufacturing is responsible of producing the new goods with the specified characteristics. Finally, Sales will deal with setting up the prices and the contracts with the retailers.

In this example, the purpose of the prototype is that each department has an agent that manages an ontology which encodes the knowledge owned by that department or process. Figure 4.1.1(a) shows the interaction between the four departments of the previous example, and how the knowledge agents are allocated. Each grey circle corresponds to an ontology associated to an agent, i.e. its knowledge domain. Figure 4.1.1(b) represents the shared symbols among ontologies, these are going to be the terms over which the agents will exchange their knowledge. Finally, Figure 4.1.1(c) shows the universal set of symbols over which all agents can represent and exchange knowledge, it serves as a standardized core of terms over which any existing and new ontology must be based on.

![Ontologies interaction and their relation with agents.](image)

---

3 Assume the engineering department is split in R&D and Manufacturing; once the product has been developed it goes into production. This example is not meant to be taken as an exhaustive example representing the complexity and variety of processes in a company.
According to the first objective of this project, the development of the standalone prototype must provide the agents with the capabilities for understanding and navigating their ontologies and also for extracting from them the information requested by the users. However, there is a communication premise required to develop the prototype, it defines that one user can only have one agent assigned, but one agent can be allocated to several users, Figure 4.1.2 illustrates this premise. The assignment is done according processes in which the user is involved, then the agent must have an ontology that represents this knowledge.  

![Figure 4.1.2. Communication premise: assignment of users per agent.](image)

In the next sections are described the initialization of the agents’ system and the two services provided to the user. The following use cases consists of a description of the activities performed by the user and the agent(s), a characterization of the use case and, where necessary, examples to help construct a better understanding.

### 4.2 Initializing the agent’s communication system

When the user runs the standalone tool, he/she is starting the prototype’s environment through a Jason platform, thus, all the knowledge agents

---

4 The benefits of using one ontology for all the processes of a company or any other variant falls beyond the defined scope of this project.
registered in it. In this moment the agents acquire their initial beliefs from the XML initialization file. This is a single file with the initial information for all the agents; this includes: the agent’s name, the agent’s credentials, the process(es) covered by the agent and its ontology, the company name (this can be changed to the division name, subsidiary or similar categorization), the ontology file path and finally, the identification numbers of those users allowed to access the agent. All these information is unique for each agent, even though, nothing prevents the configuration to repeat certain values for each agent’s initial beliefs, with the exception of the user-agent allocation, which must comply with communication premise described earlier in this section and illustrated in Figure 4.1.2.

Once the initial beliefs are grounded, the agents are ready to provide all the services to the user, upon demand. Figure 4.2.2 illustrates this process and Figure 4.2.1 shows the legend for the symbols of this and subsequent diagrams used to describe the services provided by the agents.

The initialization process can be characterized as follows. The trigger of this process is the booting of the multi-agent system by the user. The actors involved in this use case are the user and the knowledge agents in the current Jason platform. The expected goal is to provide each agent with the basic initial knowledge (initial beliefs) to be able to provide the services to the user. Finally, the failed conclusions are either when the ontology loaded by the agent is inconsistent or there is an unsatisfiable concept, given that this means that there is an error in the representation of the knowledge. Also, a failed conclusion is when a ground term contained in the XML initialization file has an illegal syntax according to the AgentSpeak and Jason rules.

![Legend](image)

**Figure 4.2.1.** Legend for the prototype services diagrams.
4.3 Communication’s design: Ontology Query Service

The query service is performed by a knowledge agent using HermiT. The agent retrieves all those axioms that represent sub-classes, equivalent classes or instances of the axiom (i.e. query) submitted by the user in OWL Manchester Syntax. This service is inspired by the functionalities of the DL Query tab of Protégé 4.3. With respect to the ontologies, the more expressive they are, the more flexible and meaningful queries can be performed by the user. This section describes the two variants of how the ontology query service can be provided by the multi-agent system.

Service variant: Single-agent

The single agent service is offered when the user requests an ontology query only of its agent’s knowledge. If no classes or instances are obtained from the ontology, a message is sent to the user acknowledging this. Otherwise, the agent shows the answer to the user in the tabular format shown in Figure 4.3.1.

---

5 For symbols meaning refer to legend on Figure 4.2.1
Here the symbols are identified by type (class or instance) and by its inference classification with respect to the query. Also, the agent will save this table in a text document in the default folder and after that it shows to the user the file path where this document was saved.

This service can be represented by a use case. The request of the query service by the user in the single-agent mode triggers the use case, therefore, the actors will be the user and its allocated knowledge agent only. The following element of this characterization is its goal, which is the delivery of an answer from the knowledge represented by the agent's ontology. Finally, a failed conclusion occurs when the query has syntax problems, i.e. is not written accordingly to the OWL Manchester Syntax rules; or when the query includes symbols that are not contained in the agent's ontology. A case that should not be confused with a failed conclusion, is when the axiom provided by the user, i.e. the query; has no subclasses, equivalent classes or instances. This will not return any symbols which in some cases can be a sensible and expected answer.

![kAgent1: Query Results](image)

**Figure 4.3.1.** Results shown to the user by the agent.
Figure 4.3.2 shows the tasks described above. This process can be started by the user once the initialization has been done.

**Figure 4.3.2.** Use case 1 diagram: Query service – Single-agent.

---

6 For symbols meaning refer to legend on Figure 4.2.1
Service variant: Multi-agent

Like the previous service, the agent provides the user with the answer to the query. However, in this case the user is requesting a comprehensive answer by selecting the multi-agent option. This means that the agent will receive the user’s query and then asks the query also to the rest of the agents in the current Jason platform and wait for the results from the other agents. Each agent will save in the default folder a table with the answers like the one shown in Figure 4.3.1 and will send the file path of this document to the requester agent. Also, each agent will show this table to the user via the requester agent. If some agent does not obtain an answer, it will not generate a document and will then send a message to the requester agent acknowledging this, instead of a file path. Finally, the user’s knowledge agent will show all the file paths to the user along with its results. Figure 4.3.3 shows this service which can be also started by the user once the agents are initialized.

The characterization of the use case for this variant of the querying service is defined as follows, the trigger, again, is the request of the user, with the multi-agent option selected. The actors include not only the user and its knowledge agent, but all the other knowledge agents in the current Jason platform. The goal is to provide the user with each of the results obtained by all the agents with respect to the submitted query. The failed conclusions as in the previous case, are when either the syntax of the query is not valid, regarding the OWL Manchester Syntax rules, or if the symbols of the query are not included in the ontology’s signature. However, in contrast with the single-agent case, it is not a failure if one or more symbols from the query are not present in some ontology signature, this can be a desired result.
4.4 Communication’s design: Restricted view generation service

In contrast with the previous service and its variants, the restricted view is a request from the user to create a new ontology, typically containing less information, from the agent’s ontology. The knowledge agent will ask the user to select the concept symbols he/she wants to forget, i.e. remove, from the original ontology. Only those obtained uniform interpolants which are consistent and which all its concepts are satisfiable will be saved and considered as useful for the user. In the following descriptions of this service, when an agent checks for the usefulness of an ontology or uniform interpolant, is checking the satisfiability and consistency conditions just described. The

---

*For symbols meaning refer to legend on Figure 4.2.1*
present section defines the single-agent and multi agent variants of the restricted view generation service.

**Service variant: Single-agent**

Once initialized the multi-agent system and the user has set the service to the “Single Agent” option, and submitted the concept symbols to forget, the agent generates a uniform interpolant, making use of the LETHE library through the agent’s internal actions. Then, the agent will check if the ontology is useful for the user, if so, the agent saves it to the default folder. Next, it shows the file path where the new ontology was saved to the user. In case the generated view is not useful to the user, the agent will display a message explaining that based on the symbols chosen to forget, it cannot build a useful uniform interpolant. Figure 4.4.1 shows a diagram for this service.

This service can be characterized as a use case too. Its trigger is the request done by the user of a restricted view of the agent’s ontology, where the single-agent option is selected. This leads to define the actors as the user and its knowledge agent only. The goal is the successful construction, by the agent, of a restricted useful to the user. Finally, this use case fails when the given set of symbols is not contained partially or completely in the signature of the agent’s ontology. It is necessary to separate this case from the ones where the restricted view is not useful for the user because the interpolant was computed, but given the semantics of the generated view, it does not satisfy the conditions to make it useful (satisfiability and consistency). Also, is not a failure since this can be a desired output for testing scenarios where a set of symbols is meant to be unable to generate a useful restricted view, this can be used in security applications or for checking knowledge integrity.
Figure 4.4.1. Use case 3 diagram: Restricted ontology service – Single-agent

For symbols meaning refer to legend on Figure 4.2.1
It is important to mention that once the restricted view of the original ontology has been successfully generated, the user can manually change the ontology path in the XML initialization file to the new ontology’s path, initialize the agent again and request it to perform any of the services described in this chapter, now using this new ontology. This possibility makes the prototype a very flexible tool for the user, where he/she can request any of the services with respect to the main ontology or to any restricted view of it. This circumstance can be illustrated by extending the example provided at the beginning of this chapter, where, for instance, a user working in the manufacturing department, instead of querying in general about the relations of the products with the marketing knowledge, he/she can generate a restricted view with respect to a specific set of products and query over it.

Another case where the use of LETHE to provide a restricted view of an ontology can be useful is when a supervisor requires to conceal sensitive or confidential information from the team in charge without restricting the access to the knowledge or creating from scratch a new ontology without these concepts and their relations.

**Service variant: Multi-agent**

Similarly, as in the multi-agent query service, the user must select the multi-agent option after requesting the agent to generate a restricted view of its current ontology. Once the concept symbols are provided to the agent, this will generate its restricted view as in the previous case, but now, it will send the concept symbols of its restricted view to all the agents in the network. Then, each of the agents will receive and compare the symbols with the set of classes contained in their ontology’s signature. The symbols present in both lists are the ones over which this agent will construct its restricted view of the ontology. After that, the agent checks if the new ontology is useful, if it is, then the agent saves it into the default folder and sends the file path of it to the requester agent. In the case where the generated view is not useful, the agent will not save the ontology and will send the requester agent a message acknowledging this. Once the requester agent has all the answers, it will display the file paths and error messages to the user accordingly with the messages received from each agent. Figure 4.4.2 and Figure 4.4.3 show the tasks described.
For this service, the characterization of it as a use case is given by a trigger defined by the user’s request to the assigned knowledge agent for a restricted view of the agent’s ontology with the multi-agent option. Consequently, the actors are the user and all the knowledge agents in the Jason platform. The goal of this use case is to obtain individual and useful restricted views of each agent’s ontologies with respect to the common symbols from those provided by the user’s agent. As in the single-agent variant, the failure for this use case is when the set of symbols contains concepts that are not present in any of the ontologies of the agents in the network. It is important to differentiate this event from the ones when a restricted view from any or all of the agents is not useful, as mentioned in the single-agent variant, it will not be a failed conclusion.

An extension to the supervisor’s information concealment example given for the previous case can be done here, where LETHE is used to generate a restricted view of the ontology to conceal sensitive or confidential information from other departments or processes, so the agents can request any information that is included in the new restricted view, rather than in the original ontology, without damaging the integrity of the public information and providing confidentiality to the sensitive one. Now the agent, can continue sharing knowledge and communicating with other agents fluently and without the risk of disclosing sensitive or confidential information.

As mentioned in the single-agent variant of this service, the user can also make the agents use the obtained restricted views to perform separate queries or even create restricted views of them, however this iterative process is not covered in the present research.
Figure 4.4.2. Use case 4 diagram part 1: Restricted ontology service – Multi-agent

9 For symbols meaning refer to legend on Figure 4.2.1
Once the communication is designed, the communicating agents’ structure needs to be set accordingly to provide all the stated services and variants mentioned in this chapter. This design is required to be flexible in order to allow upgrades and additions of tools, modules and functionalities in the future. Chapter 5 provides a complete explanation of how the knowledge agents are structured as well as how they use their functionalities to communicate with each other and with the user for exchanging knowledge.

---

10 This diagram is the continuation of that in Figure 4.4.2.
11 For symbols meaning refer to legend on Figure 4.2.1
Chapter 5. The communicating agent’s design

5.1 An overview of the agent’s design

The knowledge agents are responsible for providing the users with sensible insight from their stored knowledge as answers to the concrete requests made by the same users. For this, the agents have been designed as a Java standalone prototype, implementing several libraries with specific functionality goals. These agents need to have a series of capabilities to accomplish three main tasks. The first is understanding the symbols provided by the user and those embedded in the knowledge representations. The second is navigating these symbols and structures which contain them in order to extract a coherent set of entities. The last one is to communicate and share the requested information either with the user or with other agents in a syntactically and semantically coherent manner.

In order to provide a communication and information sharing platform, the agents are built using a Java based agent language, AgentSpeak, specifically an interpreter of this language, Jason. Jason provides the architecture for the agents to set beliefs, goals and plans which allows them to, autonomously, decide which plan to execute given a specific context, i.e. a set of beliefs and goals. This interpreter sets a platform, which allows the agents to find each other and exchange beliefs, goals and even ask for the execution of specific plans. Given that only one type of agent is required and there are several different knowledge representations, several instances of a knowledge agent are required in the multi-agent system of the prototype. In the experiments shown in Chapter 6, up to four instances were created.

Some of the standard internal actions provide the required functionalities to achieve the communication among the agents. However, it is necessary to create customized internal actions in order to provide the exact functionalities required by the agents to accomplish the prototype goals and provide the agents with the tools for interfacing with the OWL API and the LETHE library for handling the ontologies. It is also required to modify the implementation of the default communication internal actions (.send and .broadcast) in order to make the agents able to recognize the different scenarios in which they could be working. For this, it is required to access the KQML default variables in Jason’s
internal architecture and then be able to customize the listening functionalities of the agents to identify, receive and understand the messages from other agents. This is crucial to achieve a meaningful communication, according to the first and third goals set for this prototype.

Once the agents have the functionalities for communicating with each other, and the structure of beliefs, goals and plans, in order to guarantee the first and second goals of the prototype, there is required to make them able to understand and navigate the knowledge representations and sets of symbols provided. For these, the OWL API is the main tool, it provides the ability to the agents to make sense out of the received symbols and the ontologies associated. Next, in order to extract the requested knowledge and give back an answer to the user, there were required two more tools, one for each specific service. For the case of the ontology querying, there is used a reasoner, HermiT. In the next section can be found the description of the role of the reasoner in more detail. Finally, for the key service that provides a restricted view of the agent’s ontology according to the set of symbols given by the user there is used LETHE java library which provided the capability for generating uniform interpolants from an ontology. LETHE provides two options, the first uses a set of symbols to be forgotten from the ontology and the second uses a signature over which the interpolant is generated. Both functionalities are used by the knowledge agents of this prototype.

However, even when customized internal actions were programmed in Java and the OWL API and LETHE libraries were in this language too, the two ontology libraries were required to handle their packages’ dependencies using the software for project management Apache Maven. Then, for simplicity and flexibility of the prototype’s environment, an adaptor is built to connect the Jason platform via the internal actions with the OWL ontology libraries, instead of building them all together in the Jason platform. This give the benefit of having two separate fully functional environments working together, with a further advantage, changes in either of them would not affect the other.
5.2 Agent Structure

In order to provide a deeper understanding of the structure of the knowledge agents, Figure 5.2.1 presents an illustration of their full design, a visualization of the interactions between the environments, the different modules and the different data required or generated by each module.

![Diagram of Knowledge Agent structure](image)

**Figure 5.2.1.** Conceptual representation of the Knowledge Agent structure.

In leftmost side of the figure it is shown how the user, as described in the use cases in the previous chapter, first initializes the agent and then demands the services from it. This is done via the customized internal actions which transform the initialization data from the XML file and the user inputs into beliefs and goals for the agent. Once the new goals are set, the agent can decide which plans to execute. All these occur in the Jason platform.

The adaptor is the key module of the prototype because it enables access to every module. First, it receives the requirements from the agent’s internal actions and then transforms them into requests for the ontology modules. Next, the ontology modules process the data and return the answers. Finally, the adaptor takes these answers and makes them compatible with the internal actions of the agent. In summary, the agent can get answers from an ontology only by executing the appropriate plans that call the internal actions.
The knowledge agent's BDI architecture
Every instance of the knowledge agents has its set of initial beliefs. Even when the literal that describes them has the same name for all of the agents, the ground terms filling these literals are different and depend on the values set in the initialization file as mentioned in Section 4.2.

The intentions are defined by the goals that each agent have, after executing a plan, the goals are updated so the agent is guaranteed to provide the complete service, this also sets the commitment of an agent towards the completion of a service. The only common goal to all agents is the .start goal, which enables the Jason platform to create an instance of the agent.

The set of desires, or plans, is the same for all the instances of knowledge agents. The difference is in the specific beliefs that each agent has because it determines whether an agent chooses to send an answer or to wait for all the agent’s answers to be received. Each plan is directly linked to a customized internal action specifically coded to comply with the users request in terms of knowledge handling and sharing, including the compatibility with the adaptor methods. Some standard internal actions are used within the plan’s body, and within the customized internal actions code.

Customized internal actions and agent’s communication
For the communication among the agents, the main standard internal actions used are .send and .broadcast, with the performatives tell and achieve, in order to transfer beliefs and ask for a specific service or plan execution respectively. However, the flexibility for sending this information in the Jason language is limited, for which is required to access the KQML default variables used implicitly by AgentSpeak and incorporate it explicitly to the agent’s desires as if it were a plan to be executed in Jason. This gave the agents more flexibility since a message can be sent from a customized internal action without losing the default functionalities of the standard action.

The adaptor: a bridge between the agents and the ontologies
The adaptor is designed as a bridge to enable the agents access to the ontologies content in order to navigate and extract the knowledge in them for a posterior sharing. The adaptor’s methods are called by the agent’s internal
actions, then, these methods call those in the ontology management environment for querying the ontology or for generating a restricted view of the current ontology. Therefore, the adaptor needs to manage the Jason API and the OWL API dependencies in order to understand the information received and be able to convert it into an understandable format depending on which module is to receive it.

This adaptor is designed to allow only one way communications, this means that the agents can access the methods in the ontology environment, but the ontology environment cannot access the methods in the multi-agent environment. It is designed as such, due to programming restrictions by the referencing of the packages and also, to keep independent the modules. So, the agent’s module works autonomously once the user has requested a service, while the modules for ontology handling are a reactive piece of the prototype’s environment linked by the adaptor.

**The role of the OWL API**

Even though the agent has the belief of where the ontology is located, the ontology is not loaded to the Jason platform and the agents cannot navigate it and retrieve answers until the OWL API module is reached by the agent’s internal action through the adaptor. Likewise, each service provided by the agent is carried out in these modules outside the multi-agent environment, which provide high flexibility and ease of maintenance.

The OWL API is used differently for each service. For the querying service, it is used jointly with the reasoner to compute inferences from the query axiom submitted by the user. In the case of the restricted view generation, the OWL API is used jointly with the LETHE library to generate the uniform interpolant. In both cases, the reasoner and the LETHE library are extensions to the OWL API, therefore, their methods cannot be executed alone, they rely on those of the OWL API.

It is also employed, along with the reasoner, to evaluate the satisfiability of the ontology’s concepts and to test the consistency of the ontologies at the initialization of each agent and after a restricted view is generated, to ensure that the new ontology is adequate for the usage of the user and the agents.
The role of the reasoner
As mentioned in the OWL API description, the reasoner plays an important part in the initialization, the query service and the interpolant generation. The reasoner used throughout the project is HermiT. In the first and last cases enumerated, HermiT is used to proof the satisfiability of the concept symbols contained in the ontology or a restricted view and the consistency of it as well. For the second case, ontology querying, the use of this library is more extensive, in order to infer the direct and indirect subclasses, equivalent classes and instances of the submitted query axiom.

The HermiT reasoner is managed in the project as a dependency as is done with the OWL API by using the project management software Apache Maven.

The role of LETHE Library
The LETHE library is used to generate restricted views of an ontology with respect to a provided set of concept symbols or a specific signature. The prototype uses both features, the computation of the uniform interpolant by forgetting a set of symbols from the original ontology and the generation of a uniform interpolant from the original ontology based on a provided signature. Both cases can be used by the user, as he/she can send to the agent a set of symbols or a signature and the knowledge agent makes use of the LETHE library accordingly. Initially, the user was allowed to select only the symbols to be forgotten from the list presented by the agent, but after the experiments run in Chapter 6, the rest of the functionality described above is added. Section 5.4 discusses this improvement.

For the variant where the multi-agent option is enabled by the user, his/her assigned agent sends to the other agents the symbols provided. Then, all the other knowledge agents compare them against the symbols of their ontology’s signature and the symbols in common are used to generate a uniform interpolant from their ontology.

Since the LETHE library makes extensive use of the OWL API, it is also handled as a dependency of the project using the Apache Maven software.
User Interface and the interaction with the agents.

The user interaction with the agents is conceived and designed to be as easy and clear as possible, therefore a series of windows and GUIs were designed to ease the input of the services settings as well as the presentation of the results obtained by the agent or agents.

As mentioned in Section 3.2 Jason allows only alphanumerical characters for the ground terms, which are used to represent the agent’s beliefs. Thus, the prototype was provided with a parser for the user inputs and the initialization file, so the user does not need to be aware of these restrictions. The parser converts the ground terms into Jason compliant syntax and converts them back to the original string when are required to be shown in the user interface. A list of the symbols currently supported by the parser is shown in Appendix C.

When the user initializes the multi-agent system, he/she is informed if all the ontologies are useful, if any of them does not comply with the conditions of a useful ontology, then the user is informed of which ontology has the error and the multi-agent system will shut down. The user needs to correct the ontology and update the XML initialization file with a path to the correct ontology or to a compliant one.

Once the agents have been successfully initialized, the user’s agent displays a window to him/her so the type of service to be provided can be selected, also the GUI requests the user to choose if the service will be a single-agent or multi-agent one. Then, in the case of a query, a window is displayed to the user for him/her to submit the query axiom in OWL Manchester Syntax. Similarly, for the restricted view service, the user will either select the classes to forget from the list of concept symbols provided by the agent or he/she will upload a text document with the signature over which the uniform interpolant must be created.

When all the information required by the agent is submitted, it executes the service and shows the results to the user. For the ontology querying service, the agent saves a text file with the results as described in Section 4.3 and also displays a table as the one showed in Figure 4.3.1, where the user can also swap the columns to visualize the data in a customized way.
Finally, for both services, the agent displays a window to the user with the paths where the text documents, for ontology queries, or the restricted ontologies are saved, so the user can access them any time later without depending on the multi-agent system or any specific agent.

5.3 Visualizing the Communicating Agent’s
This section provides a detailed explanation on how the agents communicate with each other using customized internal action, the KQML default plans and the standard internal actions .send and .broadcast.

First, the communication is carried out within the Jason platform and makes use of the standard internal actions mentioned above which will be coded into Jason or executed from a customized internal with the respective KQML plan in the agent’s set of plans. In Figure 5.3.1, the communication is represented by the light blue circle (standard or customized internal actions) and dark blue circle (Jason Interpreter). The red arrow indicates the orientation of the communication, which can be bidirectional.

**Figure 5.3.1.** Conceptual representation of communication between two agents.

Henceforth, the communication between the two agents is represented only by these concentric blue circles from the prototype’s structure diagram. Figure 5.3.2 represents communication in a multi-agent environment. As it can be inferred from the figure, the orientation of the knowledge sharing is allowed to happen bidirectional and between any pair of agents.
Figure 5.3.2. Conceptual representation of the multi-agent communication.

Figure 5.3.3 represents a general interaction including the users. Here, it can be noticed again the communication premise mentioned in Section 4.1, where one agent can be assigned to multiple users, but one user can only have allocated one knowledge agent. Nevertheless, as this figure (below) illustrates it, the mentioned premise does not restrict the communication among agents nor the access to the knowledge from the user.

Figure 5.3.3. General interaction between users and agents.
5.4 Prototype Updates

Once the prototype was finished, the evaluation and experimentation stages were carried out, the design and results of these evaluations are described in Chapter 6. However, as stated in the last goal of the present research, any further improvement found at these stages should be implemented. Thus, this section provides a description of the improvements made to the final prototype after the execution of the evaluations and experiments. Some of them have already being mentioned given the importance and impact to the prototype and the delivery of the services to the user.

Provided the variety of ontologies available, as well as the potential requirements from the users when conceptualizing the knowledge into an ontology, the prototype was upgraded to be able to show to the users not only the labels from the concept symbols, but the IRI in cases where these symbols did not contain a label. As simple as it might appear, the visualization of the symbols was found to be affected, since before this improvement, the results obtained when using ontologies such as SNOMED, without labels, was a list of blank rows, due to the missing labels.

For large ontologies, the user might not be able or may not have enough time to select all the symbols required to be forgotten from an ontology in order to generate the desired restricted view of it. Therefore, it was enabled to load a text document with the IRI of the concept symbols over which the uniform interpolant was desired to be computed. During the experiments, this improvement proved to be a valuable time saving feature.

Finally, the agents saved the text documents and the new ontologies in a default folder, but when executing the evaluations and taking into account a possible application of the prototype where the user has an organized set of folders to aid him/her with the daily work, there was implemented the possibility for the user to choose the specific path where he/she wanted to save the new file. This helps the prototype provide a better user experience. Also, the files are saved with a standardized name and keeps track of the agent who provided that information. Also, for the restricted views, when the file name has an “F” it means that it was generated by the user’s agent using the forgetting functionality of LETHE, if it has an “I1”, it means that was the user’s agent who generated it using LETHE’s signature feature for uniform interpolants; finally, if
the name has an “I”, it means that was generated by another agent in the platform using the LETHE’s signature feature for uniform interpolants.

Other features were found to be potential improvements, but given the scope of the prototype and the complexity of the improvement were not implemented and are mentioned in detail in Chapter 7.
Chapter 6. Evaluation and Results Analysis

This chapter presents the description, results and discussion of the evaluations and experiments done to the Communicating Agents prototype. It also presents the findings about the use of LETHE, obtained during the prototype’s evaluation and experimentation stages.

6.1 Selection of OWL Ontologies

According to the fourth objective stated for the present research, the selection of the OWL ontologies used in the experiments is a sensible aspect for the performance of the prototype and, in specific, for the performance of LETHE and the reasoner. Therefore, the knowledge representations should be selected accordingly to the performance and compatibility requirements of these tools.

There were selected two ontologies, the first one was the Transportation.owl, from the Information System Group’s Ontology Repository of the University of Oxford.\textsuperscript{12} It is used for the evaluation of the prototype’s functionality and the use cases. The metrics for this ontology are shown in Table 6.1.1.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Transportation.owl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axiom</td>
<td>3340</td>
</tr>
<tr>
<td>Logical axiom count</td>
<td>794</td>
</tr>
<tr>
<td>Class count</td>
<td>445</td>
</tr>
<tr>
<td>Object property count</td>
<td>92</td>
</tr>
<tr>
<td>Data property count</td>
<td>4</td>
</tr>
<tr>
<td>Individual count</td>
<td>155</td>
</tr>
<tr>
<td>DL expressivity</td>
<td>$\mathcal{ALC}$</td>
</tr>
</tbody>
</table>

The second ontology is required for testing the prototype’s performance under different scenarios of knowledge sharing. Therefore, it was decided to use an

\textsuperscript{12} The Transportation.owl ontology can be obtained from the following website: http://www.cs.ox.ac.uk/sgq/ontologies/lib/GardinerCorpus/http___reliant.teknowledge.com_DAML_Transportation.owl/2009-02-13/sources/http___reliant.teknowledge.com_DAML_Transportation.owl.owl

\textsuperscript{13} These metrics were obtained using Protégé 4.3.
ontology currently employed in the academia or industry to evaluate not only the knowledge sharing scenarios, but also the response of the prototype to a real world ontology. The ontology chosen was SNOMED, obtained with the collaboration of Mr. Yongsheng Gao from the International Health Terminology Standards Development Organisation, IHTSDO. The metrics of this ontology are shown in Table 6.1.2, where the first important fact to point out is its size, in terms of axioms and classes, even though the expressiveness is low. Also, is important to highlight the lack of individuals in the ontology, therefore, when the ontology querying service is run on SNOMED or any extraction of it, the agent will not return any instances.

Provided the dimensions of the ontology and the importance of testing the prototype with a real world knowledge representation, there was decided to extract a module using a module extraction plug-in\textsuperscript{14} for Protégé. This plug-in was used in Protégé version 4.3.

<table>
<thead>
<tr>
<th>Axiom</th>
<th>SNOMED</th>
<th>TOP</th>
<th>BOT</th>
<th>STAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axiom</td>
<td>1,448,255</td>
<td>1,385,463</td>
<td>23,269</td>
<td>23,045</td>
</tr>
<tr>
<td>Logical axiom count</td>
<td>319,380</td>
<td>304,773</td>
<td>4,923</td>
<td>4,849</td>
</tr>
<tr>
<td>Declaration axioms count</td>
<td>319,446</td>
<td>305,130</td>
<td>4,928</td>
<td>4,890</td>
</tr>
<tr>
<td>Class count</td>
<td>319,369</td>
<td>305,070</td>
<td>4,924</td>
<td>4,886</td>
</tr>
<tr>
<td>Object property count</td>
<td>77</td>
<td>57</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Data property count</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Individual count</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DL expressivity</td>
<td>ALE</td>
<td>ALE</td>
<td>ALE</td>
<td>ALE</td>
</tr>
</tbody>
</table>

The results shown above are similar to those obtained by (Sattler, Schneider, & Zakharyaschev, 2009). From the type of modules offered by this tool (TOP, BOT, STAR) was selected STAR due to the size of the module. Then, since this module is required to be split into different portions, one for each agent, the performance of LETHE for extracting the agent’s ontologies was another factor to decide to use the module STAR. The restricted views obtained from the module are going to be the ontologies used for the knowledge sharing experiments, covered in the Section 6.3.

\textsuperscript{14} https://sites.google.com/site/ontologymodularity/

\textsuperscript{15} These metrics were obtained using Protégé 4.3.
The following three sections present the different cases of evaluation of the prototype, where according to the fifth goal of this project the prototype functionalities, use cases, communication, and knowledge sharing capabilities are tested. Also, the seventh goal, which was an additional evaluation using a human interaction, is accomplished in the following two sections, given that they were performed by impersonating a real user instead simulating one with a piece of code. Additionally, Section 6.4 present a further analysis on the use of LETHE in applications for uniform interpolation such as ontology extraction.

6.2 Prototype functionalities and use cases evaluation

This section focuses on the evaluation of the prototype’s functionalities. This set of tests verify the 3 main aspects of the prototype: the multi-agent communication and knowledge sharing capabilities, the user experience and the use cases compliance according to the definitions given in Chapter 4. This experiment is based on the example of an industry application given on Section 4.1 because it provides a simple yet exhaustive scenario to test the prototype functionalities.

The multi-agent communication testing was divided in three scenarios shown in Figure 6.2.1. First, as illustrated in Figure 6.2.1(a), the prototype was tested only with one instance of the knowledge agent in the Jason platform, and progressively more instances were added (see Figure 6.2.1(b) and Figure 6.2.1(c)). The number of instances tried were 1, 2 and 4. This is represented also in the “Number of Instances” column of the Table 6.2.1. This table is shown later in this section.

![Figure 6.2.1. Functionality evaluation cases.](image)
Next, for evaluating the user experience, all the tests were executed taking the role of a user. Therefore, the code was not altered to simulate or substitute any user-agent interaction. This practice provided a useful insight of the ease of use of the prototype while using a real world ontology, as well as the usefulness of the outputs delivered by the agents. Also, in order to validate the access and correct interaction between the user and any of the agents in the platform, there were added six additional tests (from Test #11 to #16 in Table 6.2.1), since the user can only interact directly with one agent according to the communication premise mentioned in Section 4.1, and shown in Figure 6.2.1(c). Figure 6.2.2 illustrates these additional tests where the user’s agent will change each time. In the “User’s Agent” column of Table 6.2.1 is shown this approach of the design.

![Diagram](image)

**Figure 6.2.2. Additional functionality evaluation cases**

Finally, the evaluation required to test the compliance with the use cases requires that for each multi-agent communication scenario, all possible services were tested, i.e. when there was only 1 agent in the environment the multi-agent variants of the services were not tested. Table 6.2.1 shows the
“Service Type” and the “Communication (Service Variant)” columns to describe this part of the design.

Table 6.2.1. Functionality Test design

<table>
<thead>
<tr>
<th>Test #</th>
<th>User’s Agent</th>
<th>Number of Instances</th>
<th>Service type</th>
<th>Communication (Service Variant)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>KAgent1</td>
<td>1</td>
<td>Query</td>
<td>Single-Agent</td>
</tr>
<tr>
<td>2</td>
<td>KAgent1</td>
<td>1</td>
<td>Interpolant</td>
<td>Single-Agent</td>
</tr>
<tr>
<td>3</td>
<td>KAgent1</td>
<td>2</td>
<td>Query</td>
<td>Single-Agent</td>
</tr>
<tr>
<td>4</td>
<td>KAgent1</td>
<td>2</td>
<td>Query</td>
<td>Multi-Agent</td>
</tr>
<tr>
<td>5</td>
<td>KAgent1</td>
<td>2</td>
<td>Interpolant</td>
<td>Single-Agent</td>
</tr>
<tr>
<td>6</td>
<td>KAgent1</td>
<td>2</td>
<td>Interpolant</td>
<td>Multi-Agent</td>
</tr>
<tr>
<td>7</td>
<td>KAgent1</td>
<td>4</td>
<td>Query</td>
<td>Single-Agent</td>
</tr>
<tr>
<td>8</td>
<td>KAgent1</td>
<td>4</td>
<td>Query</td>
<td>Multi-Agent</td>
</tr>
<tr>
<td>9</td>
<td>KAgent1</td>
<td>4</td>
<td>Interpolant</td>
<td>Single-Agent</td>
</tr>
<tr>
<td>10</td>
<td>KAgent1</td>
<td>4</td>
<td>Interpolant</td>
<td>Multi-Agent</td>
</tr>
<tr>
<td>11</td>
<td>KAgent2</td>
<td>4</td>
<td>Query</td>
<td>Multi-Agent</td>
</tr>
<tr>
<td>12</td>
<td>KAgent2</td>
<td>4</td>
<td>Interpolant</td>
<td>Multi-Agent</td>
</tr>
<tr>
<td>13</td>
<td>KAgent3</td>
<td>4</td>
<td>Query</td>
<td>Multi-Agent</td>
</tr>
<tr>
<td>14</td>
<td>KAgent3</td>
<td>4</td>
<td>Interpolant</td>
<td>Multi-Agent</td>
</tr>
<tr>
<td>15</td>
<td>KAgent4</td>
<td>4</td>
<td>Query</td>
<td>Multi-Agent</td>
</tr>
<tr>
<td>16</td>
<td>KAgent4</td>
<td>4</td>
<td>Interpolant</td>
<td>Multi-Agent</td>
</tr>
</tbody>
</table>

These 16 tests shown in Table 6.2.1 were run using the Transportation ontology. In order to avoid introducing unwanted noise in terms of the agent’s communication capabilities, all 4 knowledge agents were initialized with the same ontology. This guarantees that if a specific result is expected, all the agents are expected to answer and all the answers are expected to be the same.

In order to benchmark the results obtained, Protégé 4.3 was used to check the expected query results as well as the correctly use of the obtained restricted views from each agent. For the testing the query service, the axiom queried was a disjunction of two concepts: BodyMotion or ElectricDevice. For the restricted view, the set used had 15 concept symbols.

All tests were successfully executed by the Communicating Agent’s prototype and all the results were satisfactory. The knowledge was shared in all the cases without any error from the prototype. The queries’ outputs were
the same for all the agents in all the cases. For the ontology restricted view, all the agents were able to compute one with the symbols given, either by the user or by other agent, all the ontologies obtained complied with the usefulness conditions defined in Section 4.4. However, there are some findings worth mentioning when using LETHE, these are discussed in Section 6.4.

6.3 Knowledge sharing experiments

In this section is covered the main experiment of this project. It looks forward to evaluate not only the knowledge sharing capabilities of the prototype, but the performance under different realistic scenarios which help to understand when is sensible to use the knowledge sharing skills of the agents and the tools used by them and when might be better to only work with the user’s agent in single-agent mode.

The experiment consists of running a fully functional Communicating Agents prototype based on the example given in Section 4.1, where there are four instances of the knowledge agent in the Jason platform and each agent has assigned a different knowledge representation as illustrated in Figure 6.3.1. Each of these ontologies correspond to a different ontology extraction from the STAR module taken from the SNOMED ontology, as explained in detail previously in Section 6.1.

![Figure 6.3.1. Ontologies allocation for knowledge sharing experiment](image-url)
Each experimentation test will be run for both of the prototype’s services. In the case of the ontology query service, the idea is to show sensible queries which allow the prototype give meaningful answers rather than looking for a very expressive axiom. This is done by finding specific concepts with subclasses and/or equivalent classes and then, write a query that can extract them. In all the cases these queries were prepared and verified using the DL Query tab of Protégé 4.3.

For the restricted view generation service, there were selected specific sets of symbols which complied with each scenario’s requirements of universal sets and overlap sets. Each restricted view generated successfully by the prototype was also opened using Protégé 4.3 to verify the file integrity and ensure that it did not depend on the Communicating Agent’s prototype in order to be used.

The experiment consists of 3 scenarios for knowledge sharing, where each scenario will use a different set of 4 ontology extractions from the STAR module, one for each agent. Additionally, each of these sets guarantees that all concept symbols from the module are allocated in at least one ontology, according to the scenario under evaluation. All scenarios have the following three experiment variables: the existence of a universal set of concept symbols, the existence of an overlap set of concept symbols and the size of each overlap set. The first two variables can take only true or false values to indicate if that scenario deals or not with a universal set and/or an overlap set. The last variable can represent two configurations, an equal overlap size for all agents or a different overlap size for all agents. The size is measured in terms of the number of concept symbols in the set. Also, the symbols in the universal set cannot be in any overlap set.

There is a restriction enforced in this experiment in order to analyse the communication and knowledge sharing skills, the restriction is that each agent will have only two neighbours with which will share a subset of its ontology’s concept symbols (overlap set).

The Figure 6.3.2 gives an illustration of the concept symbols contained in the ontology of each of the four agents according to the experimentation scenarios. First, in Figure 6.3.2(a) each colour box (red, grey, blue and yellow) represent the set of concept symbols in each agent’s ontology, this is the
starting point to generate all the three scenarios. The black squares attached represent the same subset of universal symbols for all the agents. In (b) and (c) this same unique subset is represented by a black square with letter “U”. Now, the scenario 1 is represented by (b) and (c) and will contain a universal set of symbols and 4 overlap sets, as depicted by the darker squares or blocks with a letter “O”. Since the overlap sets have the size as a variable, then, Figure 6.3.2(b) shows the equal size configuration and Figure 6.3.2(c) shows the different size configuration. In summary, scenario 1 evaluates two configurations and for example, the ontology for Agent2 in Figure 6.3.2(b) is represented by the red box, the darker red and purple squares with an “O” and the black square with an “U”.

Scenario 2 evaluates the absence of a universal set and the two overlap size configurations. Thus, Figure 6.3.2(d) and Figure 6.3.2(e) does not contain a black square in the centre, like in scenario 1, and both configurations of the overlap size variable are represented.

Finally, scenario 3 evaluates the complete absence of shared symbols, no universal set and no overlap set, therefore, there is no black square or darker blocks in Figure 6.3.2(f). Then, the concept symbols of the Agent4’s ontology in Figure 6.3.2(f) is represented only by the yellow box. For this scenario, the communication is expected to be unsuccessful.

Figure 6.3.2. Knowledge Sharing experiment scenarios.
The overlap set representations in Figure 6.3.2 help understand the agent’s neighbourhood. For example, in Figure 6.3.2(b), Agent2 only have two neighbours, Agent1 and Agent3, therefore, Agent4 will be considered ignorant with respect to Agent2, this concept is more interesting when there is no universal set of symbols. Notice that the Ignorant Agent is relative to the knowledge agent under analysis, normally depends on who is the user’s agent.

Formally, equations below define the previously described rules that both, the overlapped sets and the universal set, must comply. To understand better these formulas, take $k_{Agents}$ as the set of all knowledge agents instances in the Jason platform, $k_{Agents} = \{k_{Agent1}, k_{Agent2}, k_{Agent3}, k_{Agent4}\}$. Then, $Universal$ represents the set of concept symbols present in all the knowledge agents that belong to $k_{Agents}$. The variables $x, y$ and $z$ represent a different instance from $k_{Agents}$. Finally, $Overlap_{xy}$ is the set of concept symbols shared between $x$ and $y$.

\[
Overlap_{xy} \not\equiv Universal\tag{15}
\]
\[
Overlap_{xy} \equiv Overlap_{yx}\tag{16}
\]
\[
Overlap_{xy} \not\equiv Overlap_{yz}\tag{17}
\]

Therefore, the common language for the communication between agent $x$ and agent $y$, where $x \neq y$, will be expressed by the following equation:

\[
Common\ Language = Overlap_{xy} \cup Universal\tag{18}
\]

**Scenario 1: Common language with Universal and Overlap sets.**

As explained above, this first scenario will have a universal set of concept symbols, an overlap set of concept symbols and the two configurations for the overlap set size. Table 6.3.1 shows the set of tests for this scenario. In this set of tests, no agent is expected to be completely ignorant to the symbols being shared by the user’s agent, $k_{Agent1}$.
Table 6.3.1. Scenario 1: Experiment tests.

<table>
<thead>
<tr>
<th>Test #</th>
<th>User’s Agent</th>
<th>Universal set</th>
<th>Overlap set</th>
<th>Overlap size</th>
<th>Ignorant Agent</th>
<th>Service type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>kAgent1</td>
<td>True</td>
<td>True</td>
<td>Equal</td>
<td>None</td>
<td>Query</td>
</tr>
<tr>
<td>1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Restricted view</td>
</tr>
<tr>
<td>1.3</td>
<td>kAgent1</td>
<td>True</td>
<td>True</td>
<td>Different</td>
<td>None</td>
<td>Query</td>
</tr>
<tr>
<td>1.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Restricted view</td>
</tr>
</tbody>
</table>

For the tests 1.1 and 1.3 the respective queries were 118622000 or 24827003, and 24827003 or 37782003. The results obtained from both queries were satisfactory, which means that each agent provided the expected answers and all matched with those obtained from Protégé’s DL Query. All the results contained the same direct subclasses, and as expected the multi-agent answers enriched the user’s knowledge with the new indirect subclasses inferred. Those indirect subclasses which were similar between agents, were part of the universal set of symbols. Hence, the knowledge sharing for all the agents and the ontology querying service is considered successful. For the restricted views generated in tests 1.2 and 1.4, the submitted symbols were 107 and 106 respectively, taken from the universal set, which allows to measure the impact of the overlap in the results.

As shown in Table 6.3.2 and Table 6.3.3 all the symbols were understood by the rest of the agents, as expected, and without interference of the overlap set. Analysing tests 1.2 and 1.4 separately, even though each agent’s ontology signatures have different sizes, the sizes of the restricted views remained the same for all the cases. For each test, the four restricted views generated had the same concepts. The only difference is that for the equal overlap test, the restricted views’ sizes were smaller than for the different overlap test’s views. Section 6.4 discusses in detail these findings.

Table 6.3.2. Signature comparison (Test 1.2).

<table>
<thead>
<tr>
<th></th>
<th>Size of agent’s ontologies’ signature</th>
<th>Submitted Symbols</th>
<th>Size of restricted view’s signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>kAgent1</td>
<td>1723</td>
<td>By User: 107</td>
<td>16</td>
</tr>
<tr>
<td>kAgent2</td>
<td>1656</td>
<td>By Agent1: 107</td>
<td>16</td>
</tr>
<tr>
<td>kAgent3</td>
<td>1751</td>
<td>By Agent1: 107</td>
<td>16</td>
</tr>
<tr>
<td>kAgent4</td>
<td>1722</td>
<td>By Agent1: 107</td>
<td>16</td>
</tr>
</tbody>
</table>
Table 6.3.3. Signature comparison (Test 1.4).

<table>
<thead>
<tr>
<th></th>
<th>Size of agent’s ontologies’ signature</th>
<th>Submitted Symbols</th>
<th>Size of restricted view’s signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>kAgent1</td>
<td>1636</td>
<td>By User: 106</td>
<td>28</td>
</tr>
<tr>
<td>kAgent2</td>
<td>1641</td>
<td>By Agent1: 106</td>
<td>28</td>
</tr>
<tr>
<td>kAgent3</td>
<td>1593</td>
<td>By Agent1: 106</td>
<td>28</td>
</tr>
<tr>
<td>kAgent4</td>
<td>1628</td>
<td>By Agent1: 106</td>
<td>28</td>
</tr>
</tbody>
</table>

For future experiments, a mixed set containing symbols from the universal and the overlap set can be used. It will be a combination between Scenarios 1 and 2 of the present experiment.

**Scenario 2: Common language only with Overlap sets.**

This scenario deals with the cases when there is no universal set, but there is overlap between every pair of agents. For this case, the tests are built using the universal set as false, the overlap set as true and the overlap sets’ size was tested in both configurations.

Given the design of the overlap, and the removal of the universal set, the Ignorant Agent depends on which is user’s agent. Therefore, even when the Ignorant Agent has the capabilities to communicate, it will only inform that none of its symbols match those used in the query axiom or in the set of symbols for generating a restricted view of its ontology. Hence, the tests needed to include the user’s agent as a variable. The user’s agent was changed in order to test all the agents for each service and each overlap configuration. This is shown in Table 6.3.4.

The results from the queries on each of the tests matched the benchmark results from DL Query tab in Protégé. Each user’s agent was asked two specific queries aimed to be answered only be itself and one of the overlapping agents. For example, in test 2.1 the user’s agent was kAgent1. First, a query was aimed to the overlap with kAgent2, the agents kAgent3 and kAgent4 did not produce any answer and communicated so to the requester agent, kAgent1. Then, another query was submitted to kAgent1 aimed to the overlap with kAgent4 according to the neighbours defined in Figure 6.3.2(d) and (e). In summary, for test 2.1 kAgent1 was asked two queries (one answered with
kAgent4 and another with kAgent2) and kAgent3 was the Ignorant Agent because it does not have an overlap with kAgent1 according with the experiment's design.

**Table 6.3.4. Scenario 2: Experiment tests.**

<table>
<thead>
<tr>
<th>Test #</th>
<th>User’s Agent</th>
<th>Universal set</th>
<th>Overlap set</th>
<th>Overlap size</th>
<th>Ignorant Agent</th>
<th>Service type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>kAgent1</td>
<td>False</td>
<td>True</td>
<td>Equal</td>
<td>kAgent3</td>
<td>Query</td>
</tr>
<tr>
<td>2.2</td>
<td>kAgent2</td>
<td>False</td>
<td>True</td>
<td>Equal</td>
<td>kAgent4</td>
<td>Restricted view</td>
</tr>
<tr>
<td>2.3</td>
<td>kAgent3</td>
<td>False</td>
<td>True</td>
<td>Equal</td>
<td>kAgent1</td>
<td>Restricted view</td>
</tr>
<tr>
<td>2.4</td>
<td>kAgent3</td>
<td>False</td>
<td>True</td>
<td>Equal</td>
<td>kAgent1</td>
<td>Query</td>
</tr>
<tr>
<td>2.5</td>
<td>kAgent4</td>
<td>False</td>
<td>True</td>
<td>Equal</td>
<td>kAgent2</td>
<td>Restricted view</td>
</tr>
<tr>
<td>2.6</td>
<td>kAgent4</td>
<td>False</td>
<td>True</td>
<td>Equal</td>
<td>kAgent2</td>
<td>Query</td>
</tr>
<tr>
<td>2.7</td>
<td>kAgent4</td>
<td>False</td>
<td>True</td>
<td>Equal</td>
<td>kAgent2</td>
<td>Restricted view</td>
</tr>
<tr>
<td>2.8</td>
<td>kAgent4</td>
<td>False</td>
<td>True</td>
<td>Equal</td>
<td>kAgent2</td>
<td>Query</td>
</tr>
<tr>
<td>2.9</td>
<td>kAgent1</td>
<td>False</td>
<td>True</td>
<td>Different</td>
<td>kAgent3</td>
<td>Restricted view</td>
</tr>
<tr>
<td>2.10</td>
<td>kAgent2</td>
<td>False</td>
<td>True</td>
<td>Different</td>
<td>kAgent4</td>
<td>Query</td>
</tr>
<tr>
<td>2.11</td>
<td>kAgent2</td>
<td>False</td>
<td>True</td>
<td>Different</td>
<td>kAgent4</td>
<td>Restricted view</td>
</tr>
<tr>
<td>2.12</td>
<td>kAgent3</td>
<td>False</td>
<td>True</td>
<td>Different</td>
<td>kAgent1</td>
<td>Query</td>
</tr>
<tr>
<td>2.13</td>
<td>kAgent3</td>
<td>False</td>
<td>True</td>
<td>Different</td>
<td>kAgent1</td>
<td>Restricted view</td>
</tr>
<tr>
<td>2.14</td>
<td>kAgent4</td>
<td>False</td>
<td>True</td>
<td>Different</td>
<td>kAgent2</td>
<td>Query</td>
</tr>
<tr>
<td>2.15</td>
<td>kAgent4</td>
<td>False</td>
<td>True</td>
<td>Different</td>
<td>kAgent2</td>
<td>Restricted view</td>
</tr>
<tr>
<td>2.16</td>
<td>kAgent4</td>
<td>False</td>
<td>True</td>
<td>Different</td>
<td>kAgent2</td>
<td>Query</td>
</tr>
</tbody>
</table>

In some cases, the neighbour agent provided an answer to the query with inferred symbols that were not known by the user’s agent. This finding provides the ground for saying that the user and its agent were enriched by these unknown symbols, yet, their ontology was not updated with them.

For the interpolant tests there is no common set of symbols to use for communicating with all the agents. However, as shown in Table 6.3.5 and Table 6.3.6, a meaningful restricted view was generated only when the set of overlapped symbols was different in size and the user’s agents were kAgent1 or kAgent4 (tests 2.10 and 2.16 from Table 6.3.4). In all the other cases the generated ontologies contained only the concept Thing, which is satisfiable and provides a consistent ontology, but is meaningless for the user. Empirically, and based on the obtained results, it can be said that without a common set of symbols for communicating among the agents, the size is not the only variable to control in order to enable the communication.
### Table 6.3.5. Signature comparison (Tests 2.2, 2.4, 2.6 and 2.8).

<table>
<thead>
<tr>
<th>User's Agent</th>
<th>Size of agent’s ontologies’ signature</th>
<th>Submitted Symbols</th>
<th>Size of restricted view’s signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent1</td>
<td>kAgent1</td>
<td>1351</td>
<td>By User: 26</td>
</tr>
<tr>
<td>Agent2</td>
<td></td>
<td>1339</td>
<td>By Agent1: 21</td>
</tr>
<tr>
<td>Agent3</td>
<td></td>
<td>1399</td>
<td>By Agent1: 0</td>
</tr>
<tr>
<td>Agent4</td>
<td></td>
<td>1184</td>
<td>By Agent1: 5</td>
</tr>
<tr>
<td>Agent1</td>
<td>kAgent2</td>
<td>1351</td>
<td>By Agent2: 21</td>
</tr>
<tr>
<td>Agent2</td>
<td></td>
<td>1339</td>
<td>By User: 41</td>
</tr>
<tr>
<td>Agent3</td>
<td></td>
<td>1399</td>
<td>By Agent2: 20</td>
</tr>
<tr>
<td>Agent4</td>
<td></td>
<td>1184</td>
<td>By Agent2: 0</td>
</tr>
<tr>
<td>Agent1</td>
<td>kAgent3</td>
<td>1351</td>
<td>By Agent3: 0</td>
</tr>
<tr>
<td>Agent2</td>
<td></td>
<td>1339</td>
<td>By Agent3: 20</td>
</tr>
<tr>
<td>Agent3</td>
<td></td>
<td>1399</td>
<td>By User: 30</td>
</tr>
<tr>
<td>Agent4</td>
<td></td>
<td>1184</td>
<td>By Agent3: 9</td>
</tr>
<tr>
<td>Agent1</td>
<td>kAgent4</td>
<td>1351</td>
<td>By Agent4: 5</td>
</tr>
<tr>
<td>Agent2</td>
<td></td>
<td>1339</td>
<td>By Agent4: 0</td>
</tr>
<tr>
<td>Agent3</td>
<td></td>
<td>1399</td>
<td>By Agent4: 10</td>
</tr>
<tr>
<td>Agent4</td>
<td></td>
<td>1184</td>
<td>By User: 15</td>
</tr>
</tbody>
</table>

### Table 6.3.6. Signature comparison (Tests 2.10, 2.12, 2.14, 2.16)

<table>
<thead>
<tr>
<th>User's Agent</th>
<th>Size of agent’s ontologies’ signature</th>
<th>Submitted Symbols</th>
<th>Size of restricted view’s signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent1</td>
<td>kAgent1</td>
<td>1315</td>
<td>By User: 68</td>
</tr>
<tr>
<td>Agent2</td>
<td></td>
<td>1401</td>
<td>By Agent1: 38</td>
</tr>
<tr>
<td>Agent3</td>
<td></td>
<td>1327</td>
<td>By Agent1: 0</td>
</tr>
<tr>
<td>Agent4</td>
<td></td>
<td>1267</td>
<td>By Agent1: 30</td>
</tr>
<tr>
<td>Agent1</td>
<td>kAgent2</td>
<td>1315</td>
<td>By Agent2: 38</td>
</tr>
<tr>
<td>Agent2</td>
<td></td>
<td>1401</td>
<td>By User: 58</td>
</tr>
<tr>
<td>Agent3</td>
<td></td>
<td>1327</td>
<td>By Agent2: 20</td>
</tr>
<tr>
<td>Agent4</td>
<td></td>
<td>1267</td>
<td>By Agent2: 0</td>
</tr>
<tr>
<td>Agent1</td>
<td>kAgent3</td>
<td>1315</td>
<td>By Agent3: 0</td>
</tr>
<tr>
<td>Agent2</td>
<td></td>
<td>1401</td>
<td>By Agent3: 20</td>
</tr>
<tr>
<td>Agent3</td>
<td></td>
<td>1327</td>
<td>By User: 38</td>
</tr>
<tr>
<td>Agent4</td>
<td></td>
<td>1267</td>
<td>By Agent3: 18</td>
</tr>
<tr>
<td>Agent1</td>
<td>kAgent4</td>
<td>1315</td>
<td>By Agent4: 30</td>
</tr>
<tr>
<td>Agent2</td>
<td></td>
<td>1401</td>
<td>By Agent4: 0</td>
</tr>
<tr>
<td>Agent3</td>
<td></td>
<td>1327</td>
<td>By Agent4: 18</td>
</tr>
<tr>
<td>Agent4</td>
<td></td>
<td>1267</td>
<td>By User: 48</td>
</tr>
</tbody>
</table>
Variables such as the selection criteria for the overlapped symbols and how this set represents the ontology entailments, are suggested to be considered in future experimentation. As far as the scope of this project is concerned, even though only in few cases the ontologies generated were meaningful, it was not due to a failure in the prototype’s functionalities, but to the intrinsic entailments of the agents’ ontologies and LETHE’s algorithms. As an additional evaluation of LETHE, these results are explored in more detail in Section 6.4

Scenario 3: Common language without Universal and Overlap sets.
This scenario aims to proof that the prototype responds sensibly to cases where the knowledge present drastic limitations to be understood in a distributed disposition. The universal set and the overlap set variables are set as false, therefore, the overlap size configuration is not relevant. In this set of tests, all the agents are expected to be completely ignorant to the user’s agent information due to there is no common language. Table 6.3.7 shows the set of tests for this scenario.

<table>
<thead>
<tr>
<th>Test #</th>
<th>User’s Agent</th>
<th>Universal set</th>
<th>Overlap set</th>
<th>Overlap size</th>
<th>Ignorant Agent</th>
<th>Service type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>kAgent1</td>
<td>False</td>
<td>False</td>
<td>-</td>
<td>All</td>
<td>Query</td>
</tr>
<tr>
<td>3.2</td>
<td>kAgent1</td>
<td>False</td>
<td>False</td>
<td>-</td>
<td>All</td>
<td>Restricted view</td>
</tr>
</tbody>
</table>

For test 3.1 only the user’s agent was able to provide an answer, all the rest of agents in the network knew nothing about the user’s query. Likewise, in test 3.2 the user’s agent was the only one with knowledge about the symbols provided by the user to produce a restricted view of its ontology. In this case, the set of symbols provided to the agent was of 280 concepts and the restricted ontology generated was of 116 as shown in Table 6.3.8. This difference between the submitted and the obtained concepts present throughout the three scenarios is covered in detail in Section 6.4 from LETHE’s perspective. For the rest of the agents, the result obtained was an ontology containing only the Thing concept, which is satisfiable and allows a consistent ontology, but is useless from the user’s perspective. Therefore, it can be said that both tests
were successful and the prototype behaves sensibly even in scenarios where the communication is not possible.

Table 6.3.8. Signature comparison (Test 3.2)

<table>
<thead>
<tr>
<th></th>
<th>Size of agent’s ontologies’ signature</th>
<th>Submitted Symbols</th>
<th>Size of restricted view’s signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent1</td>
<td>1118</td>
<td>By User: 280</td>
<td>116</td>
</tr>
<tr>
<td>Agent2</td>
<td>1085</td>
<td>By Agent1: 0</td>
<td>0</td>
</tr>
<tr>
<td>Agent3</td>
<td>1154</td>
<td>By Agent1: 0</td>
<td>0</td>
</tr>
<tr>
<td>Agent4</td>
<td>912</td>
<td>By Agent1: 0</td>
<td>0</td>
</tr>
</tbody>
</table>

In future experiments, this scenario can be used to measure the degree of ignorance of the agents with respect to the allocated ontologies. The results can be helpful for the ontology engineer in order to characterize or just verify where are the common symbols or where are the gaps in the knowledge representation, and then, enable the communication and knowledge sharing among all the agents in the multi-agent environment.

6.4 Ontology Extractions using LETHE

Given the recent development of LETHE, is considered pertinent to provide a section where the use of this tool, specifically its uniform interpolation capabilities, is evaluated. In this section, the prototype is not the focus of evaluation, it was already fully evaluated previously. First, there are presented the findings addressed in Section 6.2. These findings are shown as a comparison between the use of the forgetting and signature interpolation methods for computing a uniform interpolant. In the second part is provided an analysis of the results obtained when LETHE was used for extracting the ontologies assigned to the agents for the experiment explained in Section 6.3.

Given that investigating why these findings happen is beyond the scope of the present project, the value added by this investigation is precisely to present these findings, derived from applying LETHE in different scenarios. The results presented in this section provide testing scenarios for future work on LETHE performance analysis as well as future work for defining the optimal set of
symbols in the context of different ontologies, entailments and logics expressivity.

**Findings with respect to LETHE from the functionality evaluation**

The uniform interpolants obtained by using the forgetting method are not the same as the ones obtained by the signature interpolation method. Refer to the multi-agent service variant in Section 4.4, for the details of how each method is used by the agents. While for the forgetting method the user input only 15 concepts, for the signature interpolation method, the submitted signature only had 430 concepts from the ontology of the user’s agent (those not asked to be forgotten). The results using the forgetting method were consistent among them. Also, the results using the signature method were consistent among them. A numerical comparison is provided in Table 6.4.1.

**Table 6.4.1.** Comparison between LETHE’s uniform interpolant methods.\(^\text{16}\)

<table>
<thead>
<tr>
<th></th>
<th>Axiom</th>
<th>Transportation.owl</th>
<th>Forgetting Method</th>
<th>Signature Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axiom</td>
<td>3340</td>
<td>2609</td>
<td>2172</td>
<td></td>
</tr>
<tr>
<td>Logical axiom count</td>
<td>794</td>
<td>690</td>
<td>616</td>
<td></td>
</tr>
<tr>
<td>Class count</td>
<td>445</td>
<td>429</td>
<td>416</td>
<td></td>
</tr>
<tr>
<td>Object property count</td>
<td>92</td>
<td>81</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Data property count</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Individual count</td>
<td>155</td>
<td>155</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>DL expressivity</td>
<td>𝐀𝐋𝐇</td>
<td>𝐀𝐋</td>
<td>𝐀𝐋</td>
<td></td>
</tr>
</tbody>
</table>

In the case of the forgetting method, even when the metrics show that only 16 symbols were forgotten, when analysing it there was discovered that the algorithm forgot the 15 symbols submitted, as expected, but also forgot two unexpected symbols more. One of these symbols do not have any subclasses or instances and is a direct subclass of Thing. The other is a subclass of one of the forgotten symbols, in spite that other symbols have subclasses too and were not eliminated, also this symbol shared the same direct superclass with another concept that remained in the ontology. With respect to the forgotten object properties, those object properties that did not have any range or domain assigned were removed by the algorithm. This is what happened to

\(^{16}\) These metrics were obtained using Protégé 4.3.
the data properties as well. Also, the remaining properties hierarchy was flattened.

For the interpolation method, the 15 forgotten symbols were removed and also 14 more classes. These additional classes did not have any direct or indirect subclass, nor instances, and were direct subclasses of Thing. The forgotten individual is an instance of one of the forgotten concept. Both object and data properties were expected to be removed, since the signature submitted to LETHE in this case only included concept symbols.

Now, as mentioned in Section 2.4, LETHE’s algorithms removes unnecessary, private or irrelevant concept and role symbols from the ontology over which it is producing a restricted view. The findings explained are consistent with this property of LETHE. Therefore, the use of it by the prototype is adequate and provides the expected outputs. Nevertheless, is important to consider these conditions when defining the symbols to be forgotten or the symbols to be preserved from the original ontology.

Finally, some changes were discovered on 22 concept labels after using LETHE. These labels were swapped for the lexicon annotations of the original ontology, therefore another precaution is important to note here because the name of the concept might be changed in the restricted view, and an unaware user might get confused and be misled to think that the restricted view is wrong or does not contain some of the expected symbols.

**Findings with respect to LETHE from the knowledge sharing experiment**

As mentioned in Section 2.4, LETHE can be used for several applications, one of them is the ontology reuse for extracting sets of symbols for a specific use. Hence, given the design and requirements of the prototype’s experiments, LETHE presented an interesting application besides the one already being used by the agents themselves as a service.

There was created a java class that used the OWL API for reading the module from SNOMED, generating and saving each of the extracted ontologies. These ontologies were extracted using the signature interpolation method from the LETHE library and, according to the set of symbols required for each scenario of the knowledge sharing experiment. Also, the java class
was programmed with the same three experiment variables mentioned in Section 6.3. Hence, all the extractions were produced in sets of 4, one per agent per combination of the experiment variables. This was done also to simplify the processing of the module given requirements described in Section 6.3. for each agent’s ontology and scenario. There were extracted 20 different ontologies, in sets of 4 (due to the 4 agents), from the same SNOMED STAR module using LETHE signature interpolation method.

Given the size of the STAR module (see Table 6.1.2) the performance was a sensible variable. The java class was run from the Eclipse IDE in a Linux platform. However, the initial heap memory and maximum heap memory were set to 2.5GB and 5GB respectively. An important finding is that towards the end of each extraction, the interpolator method slows down, a possible explanation to this can be obtained from Koopmann and Schmidt (2015a), it is that the first symbols processed by the LETHE’s interpolant method are those with less occurrences in the ontology. Therefore, it can be assumed that the last symbols are going to have more occurrences, consequently more entailments and finally, will be more expensive to process, thus slower than the first ones.

All the 20 ontologies have an \( \mathcal{AL} \) expressivity (see Section 2.3), a simplification from \( \mathcal{ALE} \), obtained by the use of LETHE interpolation processes. This same table shows that the STAR module have 4886 concepts, however, given that some scenarios have overlapped symbols and universally common symbols, dividing 4886 by 4 will not give the expected number of symbols for each extracted ontology from the STAR module. Figure 6.4.1 shows the size difference among the sizes of the signatures submitted to LETHE for computing a uniform interpolant and the actual size of the signature of each extracted ontology.
There can be observed in the previous figure, that all the signatures’ sizes present variations. While a 35% of the signatures contained more symbols, 65% contained less, none were found to have the same number of concept symbols. In the cases where there are less symbols, again, Section 2.4 can give a similar answer to the one just mentioned in for the previous analysis of LETHE for the functionality evaluation. On the other hand, the 35% cases where there were found more symbols, need a further analysis of the entailments given that the universal and overlap sets are not necessarily the reason of this behaviour, also it can be due the arbitrary split of the STAR module signature not based on the concept entailments. Nevertheless, it is important to highlight that all the 20 ontologies were tested consistent and all its concepts were satisfiable.

Also, Figure 6.4.1 shows that the variability of the signatures’ size is smaller when there are universal and overlapped sets and on the contrary it is greater when no universal or overlap sets are defined. In the cases where there are universal symbols, having smaller overlap decreases the range of

\[\text{Table of Size Differences per Extraction} \]

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Uniform and Equal Overlap</th>
<th>Uniform and Different Overlap</th>
<th>Equal Overlap</th>
<th>Different Overlap</th>
<th>No Universal and No Overlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25.4</td>
<td>-17.6</td>
<td>7</td>
<td>-11</td>
<td>-104</td>
</tr>
<tr>
<td>2</td>
<td>-41.6</td>
<td>-12.6</td>
<td>-5</td>
<td>75</td>
<td>-137</td>
</tr>
<tr>
<td>3</td>
<td>53.4</td>
<td>-60.6</td>
<td>55</td>
<td>1</td>
<td>-68</td>
</tr>
<tr>
<td>4</td>
<td>24.4</td>
<td>-25.6</td>
<td>-160</td>
<td>-59</td>
<td>-310</td>
</tr>
</tbody>
</table>

\[\text{Figure 6.4.1. Signature size variability comparison.}^{17}\]

\[\text{Appendix D contains the data of the signature’s size analysis.}^{17}\]
variability and leads to less symbols in the signature. Additionally, without universal symbols, having slightly more overlap decreases the range of variability and centres it almost in zero. Finally, without universal and overlapped symbols, the extracted ontologies tend to have significantly less symbols, however it presents the higher variability.

From these findings is important to point out that the variability of the signatures’ size needs to be investigated with a bigger sample, so far these inferences can only be empirical and subject to more extensive analysis. Also, these findings set the ground and highlight some behaviours for computing ontologies for knowledge sharing purposes that are worth a more specific research.

Another aspect to take into account is the method used to split the original ontology, in this case, an arbitrary position was defined, however, it would be interesting to explore different methods, based on the entailments of the symbols, rather than the concepts alone. This exploration should be done taking into account the technical knowledge required to apply these methods, so, it can be put in perspective if whether a normal user can perform them.

Finally, LETHÉ is a useful tool with many applications, but the obtaining of sensible views requires a knowledge from the ontology and the logic behind LETHÉ’s algorithm.
Chapter 7. Conclusions

The present document has described the successful construction and testing of the Communicating Agents prototype. The tools used (the OWL API, LETHE and Jason language and platform) and the way they are articulated allow the prototype to provide knowledge sharing services to the user. These services were tested and provided successful results, confirming a satisfactory design and implementation of the presented artefact.

The knowledge management versatility of the prototype allows the user to work only within his/her knowledge domain with the single-agent option or to appeal to the knowledge available in the network using the multi-agent option. Also, the user can continue working with the prototype in single-agent mode if there cannot be retrieved any inference or restricted view from the network agents for the user’s request.

The use of OWL 2 ontologies, as the knowledge representation language, has provided a helpful and robust input for developing an application for exchanging knowledge using a multi-agent system. Additionally, the OWL API, allows the users and agents to navigate and understand the knowledge represented.

Performing ontology extractions by using LETHE, allows the agents to construct successfully a restricted view from a more general ontology. The possibility of constructing these views allows a novel application of LETHE for sharing unknown knowledge in terms of common and known symbols.

The use of Jason (language, platform and API) provides the means for creating a reactive, yet autonomous, multi-agent system capable of sending and receiving messages that can be filled with a sensible knowledge representation.

The implementation of the adaptor is crucial in the agent’s structure because it articulates the prototype to make the agent and the ontology modules work together, even though they are built in different environments. The adaptor also allows any module to be upgraded or changed without having to modify the other ones, the only requirement is to update the adaptor’s methods accordingly.
Furthermore, the adaptor allows the inclusion of additional and different modules without affecting the existing ones and providing the appropriate access to each of them, even if they are developed in different environments too. A future scenario with a more complex mix of modules might require a specialized agent to work as the adaptor. However, the impact over the flexibility of the new approach will need to be tested.

An advantage of the prototype’s design is that there is no need for additional human-agent interaction experiments, since it already provides an interface to access and experiment with all the services provided. Nevertheless, further evaluations related to the user's computational skills and improvements to the user experience should be part of the next stage of the prototype’s development.

In this sense, the experiments executed highlighted some improvements to the agent’s design and the user experience as well. These upgrades were implemented and enabled the prototype to provide the knowledge sharing services in a simpler manner and with an improvement in the robustness of the services offered.

The Communicating Agents prototype allows the knowledge sharing to go beyond the scope of the agents. The users can also share the files generated after using a service of a prototype. These files are completely independent from the prototype, thus this condition adds flexibility to use the knowledge, share it or even load it to the prototype as new ontologies for executing the knowledge services on it.

The selection of real world ontologies for the evaluation and the experimentation permitted the prototype to deliver more realistic results. These results support the prototype as a robust solution for knowledge sharing in real world applications.

More experiments are required to study the impact of the overlap sets size and overlap type’s configuration in order to understand under which scenarios these sets are suitable for knowledge sharing.

The results obtained from the experiments provided additional information about the LETHE toolkit performance. Thus, as a preliminary conclusion there can be said that the symbols of the produced interpolant may differ to those expected due to the simplifications allowed by the LETHE’s algorithms and
interpolation methods used. Hence, it is required to determine the best practices for selecting the set of symbols for producing a uniform interpolant with respect to the ontology’s expressivity and a specific interpolation method.

The symbols found in the restricted views that do not belong to the recipient agent’s knowledge base can provide a way to enrich the recipient agent’s current knowledge, leading to a formal method of learning using ontologies and LETHE. A similar approach can be taken with the new symbols received when using the ontology query service, where the query itself can be used to map the new symbols to the current ontology and thus learn them.

A prototype like the one presented in this project has some limitations inherent to the tools it makes use of. One of its limitations lies on the computational resources required to upload, navigate and reason over a specified ontology. Parameters such as the allocated heap memory and the efficiency of the garbage collector needs to be addressed carefully in order to make use of the prototype’s services and obtain results in a period of time suitable to the user’s need. This limitation is found in the processes where the OWL API, LETHE and the reasoner are used.

With respect to the knowledge sharing scenarios, a limitation discovered for the use of the prototype is that there is required a universal set of symbols to enable a meaningful communication. Overlapped knowledge between certain agents in the network requires a further analysis to understand the best parameters under which the prototype can share meaningful conceptualizations of specific knowledge.

The intrinsic construction of the knowledge representation, may also present limitations when some entailments and symbols may not be configured as the user expect, misleading him/her to request the intended knowledge inadequately. Although this is not a limitation of the prototype itself, it certainly represents a limitation for the use of the prototype in an arbitrary scenario.

Revisiting the company processes example, people needs access information in order to make good decisions, but when the access to the information is fragmented and unstandardized, gathering and using information becomes an arduous, inaccurate and endless task. The Communicating Agents prototype helps solving these situations by requiring the implementation of OWL 2 ontologies and using a multi-agent’s system that
can understand, navigate, extract and share the knowledge. Therefore, this prototype can be used in applications of corporate knowledge sharing such as distribution of best practices, lessons learned and problem solving techniques.

Given the capabilities provided by LETHE to compute uniform interpolants, a potential application of this prototype can be to generate restricted views to conceal specific knowledge without pruning the entailments and losing the meaning of the remaining symbols. However, it is required a sensible allocation of ontologies, agents and users with respect to the intended use to achieve successful implementations of this application.

Future applications of the prototype might require to allow role symbols to be added to the universal and overlap sets in order to enrich the communications and knowledge sharing. This upgrade will require a complete new set of ontologies for evaluating the prototype as well as modifications to the knowledge sharing experiments to test and understand how LETHE and the reasoner will perform, and if the demand of computational resources changes.

In particular, future work for the ontology query service includes the capability to find the direct and indirect super classes of a given axiom, in order to improve the quantity and comprehensiveness of the knowledge inferred and shown to the user. Also, for the multi-agent option, the user’s agent can highlight the new symbols in the answers received from other agents in order to help the user differentiate between the own symbols and the external ones.

Currently, the standalone prototype runs in the multi-agent platform provided by Jason. A future upgrade will be to consider a full deployment of the agents over a LAN or even the internet, where multi-thread programming, concurrent access to resources and the Jason network capabilities, such as the SACI infrastructure, will become more relevant, even the creation of new agents with different roles is a variable that will be required to evaluate.

The LETHE library provides a powerful toolkit for manipulating ontologies. The prototype has only used the uniform interpolant capabilities to solve issues within the knowledge sharing services. However, the incorporation of new modules which make use of other applications of LETHE such as the computation of logical differences, is still unexplored. The discovery of hidden entailments or even the calculation of the universal set of symbols or the overlap sets might be possible by using the complete toolkit provided by LETHE.
References


Appendix

A. Project Gantt

MSc. Project Plan

Marco Aguilar Castro - 9578805
Supervisor: Dr.-Ing. Renate A. Schmidt

ACTIVITIES
1. Research over technical context (OWL Ontologies, LETHE, Agents)
2. Define aim and objectives
3. Define prototype structure and prototype requirements
4. Define experiment and evaluation methods
5. Write the Initial Report
6. Design and develop the prototype of the agents (with OWL and LETHE packages)
7. Design and start developing the standalone platform for the information exchange
8. Write Progress Report
9. Select the required OWL ontologies for the experiments
10. Run the experiments
11. Analyse experiment’s results
12. Design and execute a human-agent interaction (if time available)
13. Update the prototype with relevant identified improvements (if time available)
14. Write Dissertation

MILESTONES (at the end of the marked activity in the Gantt)
1. Prototype Requirements
2. Initial Report
3. Standalone prototype of communicating agents
4. Progress Report
5. Evaluation results
6. Dissertation

Figure A.1. Project Gantt
B. Types of formulas used by AgentSpeak

![Diagram of AgentSpeak formulas]

**Figure B.1.** Types of formulas of AgentSpeak

Based on the Figure 3.2 from Bordini, Hübner and Wooldridge (2007, p. 45)
C. List of symbols accepted by the Ground Term parser

- First Capital Letter
- First char is number
- Space
- Dot
- Comma
- Colon
- Semi-Colon
- Exclamation mark
- Question mark
- Dash
- Slash
- Backslash
- Opening Parentheses
- Closing Parentheses
- Opening <
- Closing >
- Hash symbol #
- Line Tab
### D. Signature size comparison for the extracted ontologies using LETHE

**Table D.1.** Signature comparison between input symbols and LETHE interpolants.

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Extraction Number</th>
<th>Submission Signature Size</th>
<th>Extraction Signature Size</th>
<th>Difference between signatures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sets:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Universal</td>
<td>1</td>
<td>1697.6</td>
<td>1723</td>
<td>25.4</td>
</tr>
<tr>
<td>- Overlap</td>
<td>2</td>
<td>1656</td>
<td>1691.6</td>
<td>-17.6</td>
</tr>
<tr>
<td>Size: Equal</td>
<td>3</td>
<td>1751</td>
<td>1653.6</td>
<td>-12.6</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1722</td>
<td>1620.6</td>
<td>-25.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1351</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1344</td>
<td>1308</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1375</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1387</td>
<td>-59</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1222</td>
<td>-104</td>
</tr>
<tr>
<td>No Sets</td>
<td>1</td>
<td>1118</td>
<td>1222</td>
<td>-137</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1085</td>
<td>1154</td>
<td>-68</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>912</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Sets:** Universal - Overlap, Size: Equal

**Sets:** Universal - Overlap, Size: Different

**Sets:** Overlap, Size: Equal

**Sets:** Overlap, Size: Different

**No Sets**