2012

Use of ontologies for validating MAS analysis models

Antonio Alejandro Lopez Lorca

University of Wollongong

Recommended Citation
UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.
USE OF ONTOLOGIES FOR VALIDATING MAS ANALYSIS MODELS

A thesis submitted in fulfilment of the requirements for the award of the degree of

Doctor of Philosophy

from

University of Wollongong

by

Antonio Alejandro López Lorca

School of Information Systems & Technology

August 2012
DECLARATION

I, Antonio Alejandro López Lorca, declare that this thesis, submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the School of Information Systems & Technology, Faculty of Informatics, University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualifications at any other academic institution.

Wollongong, August 2012

___________________________
Antonio Alejandro López Lorca
ABSTRACT

Ontologies, as a knowledge representation tool, rely on formal descriptions of semantics. They have often been used in software development to support various activities and generally improve the value of the systems produced. However, their use during requirements engineering activities to validate that the product being developed complies with the client’s conceptualisation is largely unexplored. In the field of Agent Oriented Software Engineering (AOSE), a few of the extant methodologies contemplate the use of ontologies but only for modelling the domain problem to support agent communication. Due to the complexity of Multi Agent Systems (MAS), errors in modelling activities during their development can be costly. Early validation of MAS models can prevent rework or the building of a system that is non-compliant with the client’s specification. This thesis produces an automatic validation and verification MAS models process that can be applied to the model development process defined by any of the extant AOSE methodologies. An ontology-based validation and verification MAS models process add-on and an associated automatic support tool are developed. The development of the process and the tool is interleaved with three case studies to evaluate and refine them. The process is conducted iteratively to accommodate the lifecycle defined by most AOSE methodologies, and it concurrently incrementally validates the MAS models produced.

A key contribution of this research is automating the bulk of the complex tasks in the ontology-based validation process, harnessing the automatic reasoning opportunities offered by the use of formal ontologies. The process utilises an AOSE metamodel that describes the most common features of MAS identified in the literature. The process validates the MAS models for completeness against the client’s conceptualisation and verifies their structure for consistency. When problems are detected, the process supports their rectification by suggesting changes to the requirements that may have remained
undetected to the developers. The process effectiveness is validated using different case studies. Validity threats are mitigated by ensuring that different ontology engineers and AOSE modellers are used, that various problem domains are selected for the applications being developed, and that different AOSE methodologies are applied; this is achieved by varying the case studies.
ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my supervisor, Ghassan Beydoun, without whom, this thesis would have not been possible. For three years you have mentored me. Your advice has improved my researching, writing and teaching skills. Thank you for everything.

I would like to thank to all the people that discussed with me about my research. You listened to me patiently, shared your thoughts with me and gave me your feedback or pointed to new promising directions. Thanks.

It is of course not possible to keep the sanity during three years of intense work without the support of good friends. You have been an essential part of my life all this time. You have become my family and I cannot imagine how hard everything would have been without you. You are the best.

Finally, I would like to thank my family, especially my parents. You have always supported me in everything I have done. Your constant encouragement has pushed me to work hard to achieve my dreams. This thesis is for you.
A number of publications have been produced as part of the work conducted for this thesis in peer-reviewed international journals and conferences and in a book chapter.

CONFERENCES


BOOK CHAPTER


# CONTENTS

1 INTRODUCTION 1
1.1 Background and Motivation 1
1.2 Research Objective 3
1.3 Significance of the Research 6
1.4 Organisation of the Thesis 7
1.5 Chapter Summary 9

2 BACKGROUND AND LITERATURE REVIEW 11
2.1 Agents and MAS 11
   2.1.1 Definition of MAS 11
   2.1.2 Benefits of MAS Technology 12
2.2 Overview of AOSE Methodologies 14
   2.2.1 Limitations of AOSE Methodologies 17
   2.2.2 V&V Support in AOSE 19
2.3 Ontologies Background 21
   2.3.1 Ontology Definition and Taxonomies 21
   2.3.2 Benefits of Ontologies 23
   2.3.3 Ontology Tools and Languages 26
2.4 Ontology-driven Software Development 29
   2.4.1 Roles of Ontologies in Software Development 31
   2.4.2 Ontologies for Requirements Engineering 32
   2.4.3 Ontologies beyond Requirements Engineering 39
2.5 Ontology-driven AOSE 42
   2.5.1 Ontologies Applied to AOSE 43
   2.5.2 Ontologies as a Validation Vehicle in AOSE 46
2.6 Conclusion 48
2.7 Chapter Summary 50

3 RESEARCH DESIGN 53
3.1 Overview of Design Science Research Method 53
3.2 Phase 1: Problem Identification 55
3.3 Phase 2: Initial V&V Process Definition 58
3.4 Phase 3: V&V Process Evaluation 59
3.5 Phase 4: Automating the V&V Process 61
3.6 Phase 5: Evaluation of the Automatic V&V Process 61
3.7 Chapter Summary 63

4 AN ONTOLOGY-BASED MAS MODELS V&V PROCESS 67
4.1 Definition of the Ontology-based V&V Process 67
  4.1.1 Task 1: Ontology Acquisition 69
  4.1.2 Task 2: Ontology Augmentation 71
  4.1.3 Task 3: Ontology Validation 74
  4.1.4 Task 4: MAS Software Models Validation 76
  4.1.5 Task 5: MAS Software Models Improvement 77
4.2 MAS Models Ontology-based Validation Operators 79
  4.2.1 Goal Model Operators 80
  4.2.2 Role Model Operators 82
  4.2.3 Organisation Model Operators 83
  4.2.4 Environment Model Operators 84
  4.2.5 Agent Model Operators 86
  4.2.6 Interaction Model Operators 88
  4.2.7 Scenario (Model) Operators 89
4.3 Preliminary Validation of Operators 91
  4.3.1 The Aircraft Turnaround Problem 92
  4.3.2 Ontology Acquisition 94
  4.3.3 Ontology Augmentation 96
  4.3.4 Ontology Validation 98
  4.3.5 Models Validation – Iteration 1 98
  4.3.6 Models Improvement – Iteration 1 100
  4.3.7 Models Validation – Iteration 2 100
  4.3.8 Models Improvement – Iteration 2 102
  4.3.9 Models Validation – Iteration 3 105
  4.3.10 Models Improvement – Iteration 3 106
  4.3.11 Models Validation – Iteration 4 106
4.4 Conclusions of the Case Study 107
4.5 Chapter Summary 111

5 AUTOMATING THE ONTOLOGY-BASED MAS MODELS V&V
  PROCESS 113
5.1 Co-Evolving the Ontology and the MAS Models 113
5.2 Refining the Comparison Operators 118
5.3 Automating the V&V Process 122
5.3.1 The AOSE Metamodel 124
5.3.2 Revisiting the Ontology Augmentation 138
5.3.3 Operators Application 142
5.4 A Support tool for Automating the V&V Process 143
5.4.1 Domain Ontology Augmentation Module 144
5.4.2 Ontology-Models Comparison Module 150
5.5 Chapter Summary 154

6 Evaluation of the Automatic V&V Process 157
6.1 Validating the Automatic V&V Process 158
6.1.1 The Wine Broker Domain 158
6.1.2 Ontology Acquisition – Iteration 1 160
6.1.3 Ontology Augmentation – Iteration 1 161
6.1.4 Ontology Validation 164
6.1.5 Operators Application – Iteration 1 164
6.1.6 Discrepancy Removal – Iteration 1 165
6.1.7 Ontology Acquisition – Iteration 2 167
6.1.8 Ontology Augmentation – Iteration 2 167
6.1.9 Operators Application – Iteration 2 168
6.1.10 Discrepancy Removal – Iteration 2 169
6.1.11 Ontology Acquisition – Iteration 3 171
6.1.12 Ontology Augmentation – Iteration 3 171
6.1.13 Operators Application – Iteration 3 172
6.1.14 Discrepancy Removal – Iteration 3 173
6.1.15 Ontology Acquisition – Iteration 4 174
6.1.16 Ontology Augmentation – Iteration 4 174
6.1.17 Operators Application – Iteration 4 175
6.2 Conclusions of the Wine Broker Case Study with ROADMAP 175
6.3 Ontology-based Operators Adjustment for MOBMAS 179
6.3.1 Task Model Operators 179
6.3.2 Role Model Operators 180
6.3.3 Organisational Model Operators 181
6.3.4 Resource Model Operators 182
6.3.5 Goal Model Operators 183
6.3.6 Agent Model Operators 184
6.3.7 Operators Application Mechanism Adjustment 185
6.4 V&V Process Methodology Independence Evaluation 188
6.4.1 Ontology Acquisition – Iteration 1 189
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4.2</td>
<td>Ontology Augmentation – Iteration 1</td>
<td>189</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Operators Application – Iteration 1</td>
<td>191</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Discrepancy Removal – Iteration 1</td>
<td>194</td>
</tr>
<tr>
<td>6.4.5</td>
<td>Ontology Acquisition – Iteration 2</td>
<td>196</td>
</tr>
<tr>
<td>6.4.6</td>
<td>Ontology Augmentation – Iteration 2</td>
<td>196</td>
</tr>
<tr>
<td>6.4.7</td>
<td>Operators Application – Iteration 2</td>
<td>196</td>
</tr>
<tr>
<td>6.4.8</td>
<td>Discrepancy Removal – Iteration 2</td>
<td>198</td>
</tr>
<tr>
<td>6.4.9</td>
<td>Ontology Acquisition – Iteration 3</td>
<td>199</td>
</tr>
<tr>
<td>6.4.10</td>
<td>Ontology Augmentation – Iteration 3</td>
<td>199</td>
</tr>
<tr>
<td>6.4.11</td>
<td>Operators Application – Iteration 3</td>
<td>200</td>
</tr>
<tr>
<td>6.5</td>
<td>Conclusions of the V&amp;V Process Evaluation</td>
<td>200</td>
</tr>
<tr>
<td>6.6</td>
<td>Chapter Summary</td>
<td>203</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion</td>
<td>205</td>
</tr>
<tr>
<td>7.1</td>
<td>Research Summary</td>
<td>205</td>
</tr>
<tr>
<td>7.2</td>
<td>Research Contribution</td>
<td>208</td>
</tr>
<tr>
<td>7.3</td>
<td>Research Limitations and Future Research Directions</td>
<td>210</td>
</tr>
<tr>
<td>7.4</td>
<td>Concluding Remarks</td>
<td>212</td>
</tr>
</tbody>
</table>

**REFERENCES** | 213
LIST OF FIGURES

Figure 2.1  Semantic Web layer cake  28
Figure 3.1  Research phases  55
Figure 3.2  Activities of Phase 1 of research process  56
Figure 3.3  High level V&V process description  58
Figure 3.4  Research process summary  64
Figure 4.1  Iterative and incremental process  69
Figure 4.2  Ontology-based V&V process overview  70
Figure 4.3  A goal model example  81
Figure 4.4  A role model example  83
Figure 4.5  An organisation model example  84
Figure 4.6  An environment model example  86
Figure 4.7  An agent model example  87
Figure 4.8  An interaction model example  89
Figure 4.9  Aircraft turnaround problem specification  92
Figure 4.10  Case study 1 ontology excerpt  96
Figure 4.11  Case study 1 augmented ontology excerpt  97
Figure 5.1  Overview of the refined V&V process  115
Figure 5.2  Task 1: Ontology Preparation  116
Figure 5.3  Task 1.2: Domain Ontology Augmentation  117
Figure 5.4  Task 2: MAS Models Validation  117
Figure 5.5  Task 2.1: Ontology-Models Comparison  118
Figure 5.6  Ontology augmentation complexity  123
Figure 5.7  Model inter-dependencies example  123
Figure 5.8  AOSE metamodel  127
Figure 5.9  AOSE metamodel excerpt  128
Figure 5.10  Example of property chain  133
Figure 5.11  Pursues property chain definition  137
Figure 5.12  Pursues property chain example  138
Figure 5.13  Example of domain ontology  139
Figure 5.14  Minimal number of connections  142
Figure 5.15  Selection ontology excerpt  143
Figure 5.16  Ontology augmentation steps  145
**Figure 5.17** Role hierarchy inconsistency 146
**Figure 5.18** Operators application steps 151
**Figure 5.19** Discrepancy report 153
**Figure 6.1** Graphic description of the wine broker 159
**Figure 6.2** Case study 2 ontology excerpt 161
**Figure 6.3** Classification of case study 2 concepts 162
**Figure 6.4** Augmented case study 2 ontology excerpt 1 163
**Figure 6.5** Augmented case study 2 ontology excerpt 2 163
**Figure 6.6** Augmented case study 2 ontology excerpt 3 164
**Figure 6.7** A task model example 180
**Figure 6.8** A role model example 181
**Figure 6.9** An organisational model example 182
**Figure 6.10** A resource model example 182
**Figure 6.11** A goal model example 183
**Figure 6.12** An agent model example 184
**Figure 6.13** Extended selection ontology excerpt 186
**Figure 6.14** Case study 3 ontology excerpt 190
**Figure 6.15** Classification of case study 3 concepts 191
**Figure 6.16** Augmented case study 3 ontology excerpt 1 192
**Figure 6.17** Augmented case study 3 ontology excerpt 2 192
**Figure 6.18** Augmented case study 3 ontology excerpt 3 193
**Figure 6.19** Augmented case study 3 ontology excerpt 4 194

**LIST OF TABLES**

Table 2.1 Ontologies for requirements engineering 33
Table 2.2 Ontologies for AOSE 45
Table 4.1 AOSE relations for ontology augmentation 72
Table 4.2 A scenario model example 90
Table 4.3 Case study 1 application times 108
Table 5.1 Example of MAS ontology augmentation 122
Table 5.2 AOSE metamodel inverse properties 131
Table 5.3 Metamodel property chains 134
Table 5.4 Sub-property mapping 139
Table 5.5 Domain and range of metamodel properties 140
Table 5.6 Inferred domains and ranges 141
Table 5.7 Domain concepts classification into AOSE classes 141
Table 5.8 Modelling elements used in MAS models 144
Table 5.9 Types of ontology-based operators 154
Table 6.1 Estimated times for the second case study 177
Table 6.2 Modelling elements for MOBMAS 187
Table 6.3 V&V process overhead comparison 201
Table 6.4 Second and third case study time comparison 202
Table 6.5 Ontology and models improvement times 203

LISTINGS

Listing 5.1 OWL 2 code for zero or more cardinality 132
Listing 5.2 OWL 2 code for one or more cardinality 132
Listing 5.3 OWL 2 code for exactly one cardinality 133
Listing 6.1 Cardinality violations for concept Bank 167
Listing 6.2 Cardinality violations for iteration 2 172
INTRODUCTION

This chapter introduces the main aspects of this research that will be discussed in later chapters of the thesis. Section 1.1 sets the background and motivation of this research. Section 1.2 outlines the goals of the research in this thesis. Section 1.3 describes the significance of the research for the scientific community in general. Section 1.4 provides an overview of the thesis structure. Section 1.5 concludes the chapter.

1.1 BACKGROUND AND MOTIVATION

Software development practices have changed greatly in the last half century. Development paradigms and associated programming languages have evolved to achieve higher levels of expressivity and abstraction. Object oriented technologies and agile developments have made structured programming styles obsolete. Emerging trends, such as model driven engineering and agent oriented technologies, seek to facilitate the development of even more complex systems by defining high-level abstraction constructs that mimic domain elements in the application environment (Schmidt 2006; Shoham 1990). This increasing complexity leads to ever growing emphasis on validation and verification. For instance, complex software systems are typically first released to users as beta versions. As expected, users discover bugs in these beta versions. Patches are developed to address these identified bugs and more trusted software releases are made.

Modelling is a central activity in software development processes. Modelling is also a general human activity conducted to make sense of the world. In the context of a software development process, the outcome of modelling is typically a model of a system, which is a description or specification of the system and its environment for a specific purpose (Alhir 2003). A typical software development pro-
cess includes organisation of team members and project stakeholders, and uses a comprehensive structure so that everyone involved can understand the current state of the project. The process involves common activities such as the investigation of user requirements (analysis phase), the setting up of necessary features of system (specification phase), the creation (or adaption) of a suitable solution (design phase), the development of the proposed solution (implementation phase), certification that the solution solves the original problem (testing phase), certification that the solution works in context (integration phase) and modification of the working solution as new requirements are identified (maintenance phase). The outcomes of these various activities are models that represent various perspectives on the system that is sought. These models represent complementary aspects of software systems.

Across various phases and activities of the software development process, models commonly overlap. This occurs to varying extents. For example, using the Rational Unified Process (RUP) (Kruchten 2001), the Unified Modelling Language (UML) class diagrams (in early analysis) and sequence diagrams (in late analysis and early design) overlap in the definition and uses of classes and objects. Removing a class from the class diagram forces the removal of the lifelines that are associated with that class from the sequence diagram. Models from different phases may also be developed by different software developers. For instance, the first type of model (e.g. the UML use case) seeks to represent elements of the problem space to facilitate understanding of the client’s requirements. This model may be developed by a senior analyst who meets with the client. The second type of model (e.g. the UML state machine diagram) focuses on the solution that the developer creates to address the problems identified by the analyst. This model may be developed by a senior programmer who communicates only with the senior analyst (not the client). While modelling activities generally facilitate the development of complex systems, it is important to note that, as these activities are undertaken by various people, at different times, using different representations and capturing varying facets of the system, the process of modelling can be prone to errors.
Cognitive limitations of modellers, together with the interdependencies of models, can lead to errors. These errors can then propagate to later phases of the development cycle and become more costly to fix (Westland 2002). This thesis is concerned with capturing such errors in the requirement analysis models before they propagate to later phases. The focus of the thesis is agent oriented analysis as used in agent oriented software engineering (AOSE). The unique characteristic of AOSE that differentiates it from other development paradigms is that the central modelling units of the paradigm are agents (Wooldridge 2002). These model agent software entities are goal driven and situated in an environment. These entities form blocks to build a Multi-Agent System (MAS). The system goals are achieved through the collective goals of the agents. Within a system, these agents continuously sense the environment and react accordingly, however it is also possible for the agents to be proactive to achieve their goals. Agents can affect the environment in which they are situated and can be affected by the actions of other agents. The outcome of an agent’s action may be dependent on the execution state of the environment. One important and common feature of agents is that they communicate with other agents, and can negotiate or compete with other agents to achieve their goals (Bordini et al. 2007).

1.2 RESEARCH OBJECTIVE

Despite their potential in terms of distribution, robustness and resiliency, agent oriented technologies have not been adopted by industry to date (Pavón et al. 2008). This is due, in large part, to their inherent complexity and lack of a standard development methodology (Pechoucek and Marik 2008). Many authors have proposed Multi-Agent Systems development methodologies targeting various types of applications (Henderson-Sellers and Giorgini 2005). Each methodology requires a different set of models to define the system (Tran and Low 2005). Dependencies between agent models are even greater than the dependencies in existing paradigms. Development phases also overlap; for instance, requirement models can also interact with
low level design models in some cases (Miller et al. 2010). This thesis focuses on ensuring that these dependencies are properly followed and provides a process to detect actual and potential errors in the requirement analysis phase. This process is underpinned by the use of a reference model – a domain ontology – to ensure that the analysts do not deviate or introduce inconsistencies during modelling.

The focus on validating requirement analysis models is rooted in their status as the basis for the rest of the development cycle. Errors in the requirement analysis activities can lead to disastrous consequences if they are not quickly detected. For example, such errors can lead to products that do not fulfil the expected functionality, exceeding budget or great delays (Ellis 2009). Indeed, as articulated by the 1999 Turing Award winner Fred Brooks in his widely cited paper (Brooks 1987), “The hardest single part of building a software system is deciding precisely what to build. . . No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later”. The most usual mechanism used to conduct the requirement validation is the formal technical review (Pressman 2005). The review team includes developers and users who examine the specification looking for errors in content, missing information, inconsistencies, conflicting or unrealistic requirements. Although this technique can point to many errors, it has to be performed manually and is highly time-consuming. In this thesis, domain ontologies are instead used as a ‘sounding board’ to check the semantics of the models produced during analysis. They can formally represent any given domain by defining its concepts, relations between concepts and associated axioms or rules (Guarino 1998). Ontologies are typically expressed using semantic-capable languages that can be understood by humans and computers, such as OWL 2 (W3C 2009a). The thesis capitalises on this and uses ontologies to automatically check models’ consistencies and simultaneously formulate queries to the developers to ensure that potential errors are resolved before they propagate. Towards this, the goals pursued are as follows:

- To first study how ontologies are currently used along the software development lifecycle, in particular in the validation of requirements models in AOSE.
• To design a process to use ontologies for the validation of AOSE software models, that can be adapted for any AOSE development methodology.

• To automate the AOSE validation approach as a decision support system tool, capitalising on the formalism of ontologies.

• To validate the validation process and the tool in several case studies.

To further highlight the scope of this thesis, it should be noted that the following goals were not pursued as they were outside the scope of the research:

• The thesis will not define a new AOSE methodology. Rather, a validation add-on process that can be used together with any of the many existing development methodologies will be created.

• The thesis will not propose new methods to build ontologies. Rather, the thesis assumes that the validation process will start with a suitable existing ontology. The thesis will provide mechanisms to augment this initial ontology and to make it suitable for the application of the validation process.

• The thesis will not provide a mechanism to automatically modify the agent models. Rather, it will provide a tool that analyses the models, generates a report listing aspects susceptible for review and proposes improvements. The developers (in consultation with the client at times) will decide whether to accept or to reject any automatically proposed improvement. The tool will guide the developer through the ramifications of any changes in the models.

• The thesis does not implement Multi-Agent Systems (MAS). The validation process concludes when a set of analysis MAS models that are internally consistent and compliant with the client’s requirements is produced. The evolution of the analysis models to design and detailed design models and their codification into a software product and deployment of the system is outside the scope of the thesis.
1.3 SIGNIFICANCE OF THE RESEARCH

Ontologies have been applied, as reusable components, to software systems at run time and development time (Dillon et al. 2008). Their use, however, for the validation of requirements models is scarce. Some authors have used ontologies in the verification of the structure of the developed models (Benevides et al. 2009; Benevides and Guizzardi 2009), but none have used them to validate that the models comply with the client requirements. This thesis focuses on the validation tasks of the requirements engineering phase, exploiting and reusing ontologies as a knowledge resource. As one component of this, the thesis pursues an understanding of how ontologies can be used to improve the development of Multi-Agent Systems (MAS). This knowledge will advance the field of Agent-Oriented Software Engineering (AOSE) leveraging the benefits of ontology-based approaches, focusing on the role that ontologies play on the validation of MAS models in early phases of the development process.

A key contribution of this research is the definition of a process for the validation of MAS models. This process validates the models in early stages of development, avoiding compound errors carried throughout the whole product lifecycle. The later the detection of an error, the more expensive it becomes to fix it, as more workproducts are affected. The provision of a process that identifies errors in the early stages of development removes one of the obstacles that discourage industry from adopting the agent technology. In this sense, the thesis contributes to harnessing the commercial potential of MAS technology. It will contribute to the acceptance of AOSE in industrial projects by providing a process to achieve high quality and error free models. Moreover, newcomers to the AOSE paradigm will find the validation process and support of this thesis useful, as AOSE has particularities that are non-existent in other paradigms. The validation process also generally assists inexperienced modellers on modelling tasks. When tackling a project dealing with a complex or unknown domain, experienced modellers will also benefit from its use.

The validation framework is methodology agnostic. It can be applied to any AOSE methodology. The augmentation of the domain
ontology is the only method-dependent step and this is driven by the methodology metamodel. Therefore, the metamodel driven formulation of the validation process can conceivably be applied to various paradigms, provided that appropriate metamodels are available. In other words, once the ontology adaptation is made, the validation process is seamless regardless of the paradigm, methodology or models used. In AOSE, most methodologies have an explicit metamodel (Henderson-Sellers and Giorgini 2005). Indeed, a generic metamodel for MAS has recently been developed and has been shown to describe at least 22 methodologies (Beydoun et al. 2006a, 2009; Low et al. 2009). This thesis allows for reuse of this metamodel and other metamodels. This is a significant knowledge reuse considering the efforts that are invested in developing the methodologies metamodels. Indeed, this knowledge reuse is a notable contribution of this thesis. Knowledge reuse is also prominent in reusing existing domain ontologies. With the advent of the Semantic Web, more ontologies are developed and made available, e.g. Swoogle (2007), the OBO Foundry (2012), the TONES Ontology Repository (2010) or the Pronto Ontology Repository (2008). For many domains, it is likely that a suitable ontology can be found in an open repository or that one can be engineered using one or more related ontologies. As the validation tool only requires a lightweight ontology to trigger automatic suggestions and error detection, this development can be relatively fast. This in turn can become a reusable ontology for future validations in the same or similar domains.

1.4 ORGANISATION OF THE THESIS

This thesis is organised into seven chapters, following the structure of the research that has been conducted:

- **Chapter 1 – Introduction.** This chapter introduces the background, motivation, objectives and significance of the research. It also outlines the structure and content of the thesis.

- **Chapter 2 – Background and Literature Review.** This chapter reviews the concepts that constitute the core of the research:
ontologies and agent technologies. It then examines how these concepts have been used in existing approaches. It focuses on usage of ontologies to improve software development, also known as ontology-driven software development, and uses of ontologies in AOSE. The chapter concludes with a summary of the shortcomings of the existing approaches and an identification of the research gap in the field.

- **Chapter 3 – Research Design.** This chapter introduces the research methodology adopted for this project. The goals of the research are revisited. The required research activities are then described.

- **Chapter 4 – An Ontology-based MAS Models V&V Process.** This chapter outlines a general process to validate AOSE software models using ontologies. This process is conceived as an add-on to be used with any existent AOSE development methodology. Then, the validation process is applied in a real world case study. The scenario of the experiment, its development and outcomes are presented, and conclusions are drawn. These will be used to refine and automate the process.

- **Chapter 5 – Automating the Ontology-based MAS Models V&V Process.** The case study conclusions are used to improve and extend the validation process. A reasoning mechanism amenable for automation is synthesized to support the validation. A concomitant tool is developed.

- **Chapter 6 – Evaluation of the Automatic V&V Process.** The automated process and the tool developed in the previous chapter are applied in a second and third case study to test and verify their developer, domain and methodology independence.

- **Chapter 7 – Conclusion.** This chapter closes the thesis by examining the conclusions of the research, its limitations and future paths to be explored.
1.5 CHAPTER SUMMARY

This chapter has introduced the research that is presented in this thesis and its significance. The background of the research, including a justification of its motivation and goals, was provided. An overview of the thesis structure and content has also been presented. In the next chapter, a more detailed look at the existing literature will be presented. This will further highlight the significance and the contributions of this thesis to the agent oriented and ontologies communities in general, and to requirements validation and verification specifically.
BACKGROUND AND LITERATURE REVIEW

This chapter reviews the essential concepts that underpin the research presented in this thesis and relevant related works. It examines works related to Multi-Agent Systems (MAS) and ontologies. The extent of ontology usage in requirements engineering, in the various phases of the software development lifecycle in general, and for MAS development in particular, are also considered. The chapter concludes with a summary statement on the emerging trends.

The chapter is organised as follows: Sections 2.1, 2.2 and 2.3 provide background on MAS, AOSE methodologies and ontologies, respectively; Section 2.4 examines existing applications of ontologies throughout the software development lifecycle; Section 2.5 focuses on how ontologies have been used during the development of MAS; Section 2.6 presents the conclusions extracted from the literature review; and finally Section 2.7 concludes the chapter.

2.1 AGENTS AND MAS

This section introduces the concept of MAS and related ideas that are relevant to this thesis. Important characteristics of MAS are first examined. The benefits of MAS technology adoption and acceptance of MAS by industry are briefly overviewed.

2.1.1 Definition of MAS

Lesser (1999) defines a MAS as a computational system where two or more agents interact or work together to perform some tasks or achieve some goals. An agent is defined by Wooldridge (2002) as a software entity that is situated in some environment, and that is capable of autonomous action in this environment in order to meet
its design objectives. Features attributed to agents are (Bordini et al. 2007):

- Agents are situated. Agents can sense their environment and modify it through a set of possible actions.

- Agents are non-deterministic. Agents do not have complete control of their environment. Effects of actions cannot be assumed to be the expected until a posteriori checks are performed. Many agents can affect the environment concurrently. Different executions of the same action in the same situation may not produce the same effect.

- Agents are autonomous. Agents can operate independently in order to achieve their goals. Decisions made to achieve goals are under an agent’s control and are not driven by others.

- Agents are proactive. Agents are not passive entities waiting for orders. They try to achieve the goal they have been assigned, showing goal-directed behaviour.

- Agents are reactive. Agents are responsive to changes in the environment. They follow plans to achieve their goals. If plans do not produce the expected results, agents choose an alternative course of action.

- Agents are cooperative. Agents can cooperate with other agents to coordinate efforts to accomplish their goals.

Typically, the problems that MAS tackle are complex and exceed the capabilities or knowledge of individual agents. Applying MAS analysis, the problem is split into sub-problems that can be solved using the most appropriate paradigm or technique. Interdependent problems require collaborations between agents. In contrast to traditional non-distributed information systems, in MAS there is no global control authority and data are decentralised (Sycara 1998).

2.1.2 Benefits of MAS Technology

The characteristics outlined above make MAS especially suitable for scenarios where distribution and autonomy are important (Pechoucek 2007).
and Marik 2008). MAS lend themselves to scenarios where the required knowledge or data are not available centrally or the system control needs to be distributed. Autonomy enables agents to develop automated decision making applications where the user delegates responsibility to the system, which in turn may have to deal with unexpected situations. In particular, three types of domains comply with this description: geographical distribution of knowledge and control or environments with partial or temporary communication accessibility, for example logistics, collaborative exploration, mobile and collective robotics or pervasive systems; competitive domains where knowledge sharing is restricted, for example, e-commerce, e-business or supply-chain management applications; and finally, domains that require time critical response and robustness, for example, industrial control systems with replanning or fast local reconfiguration.

MAS also provide intuitive modelling concepts suitable for simulation and modelling scenarios where an easy migration from simulation to deployment in a real environment is desirable. MAS versatility can be harnessed in open systems scenarios that require integration and interoperability of software systems not known a priori, such as ubiquitous computing. Developing complex systems can be facilitated by agent technology, as individual agents reason about and solve different parts of the problem, negotiating to achieve the final solution. From the existing literature, Bergenti and Vargiu (2010) identified the business sectors that have adopted agent technologies to some extent. Amongst these identified sectors, they highlight simulation and training applications for the defence domain (Sierhuis et al. 2009), logistics and supply-chain management (Jacobi et al. 2008) and industrial control systems (Pechoucek et al. 2008). They present three case studies of applications of MAS technologies in Italian industry. Pechoucek and Marik (2008) also analyse MAS applications in industry. They describe projects in the domains of on-board ship equipment management, engine assembly planning, air traffic control and RFID enabled material handling control. The authors also collect the reports of MAS applications in industry published elsewhere, categorising these works depending on their focus area, namely, manufacturing control, logistics, production planning,
simulation, agent based unmanned aerial vehicle control, space exploration applications, training, distributed diagnostics, networking and supply chain integration.

Significant effort has been dedicated to promote MAS in industry. However, the agent based paradigm has not yet been widely adopted. Since its initiation in 2005, a well-established industry track has been held annually as part of the main discussion forum for the MAS community, the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS). In 2010 the track changed its name from "Industry Track" to "Industry and Applications Track" and from 2011 on to "Innovative Applications Track" to allow the participation of academic contributors (perhaps because of a limited involvement of industry participants). Pechoucek and Marik (2008) identify bottlenecks that make agent-based applications more popular among research groups than large industrial organisations: There is limited awareness of the advantages of agency, which prevent the adoption by many industries that could benefit from their application. Even when industry uses them, successful MAS projects are not sufficiently publicised. The technology capabilities are often misunderstood. This leads to over expectations of early industry adopters and frustration when project outcomes do not fulfil expectations. There is a general risk associated with the adoption of a relatively new technology that has not been proven in a large scale project before. Finally, the lack of mature development tools suitable for industrial development also creates hurdles for its adoption.

2.2 OVERVIEW OF AOSE METHODOLOGIES

A noted in Chapter 1, AOSE promotes a software development paradigm where the central modelling units are agents (Wooldridge 2002). A number of AOSE methodologies can be found in the literature. Some of the most notable examples are Gaia (Wooldridge et al. 2000), Adelfe (Bernon et al. 2003), Prometheus (Padgham and Winikoff 2004), PASSI (Cossentino and Potts 2002) or INGENIAS (Pavón et al. 2005). The methodologies define various modelling languages, steps, techniques and models to produce MAS (Argente
They often use different agent-based constructs and target different development settings or phases. Gaia (Zambonelli et al. 2005), for instance, supports the full development cycle of MAS from analysis to implementation. Prometheus (Padgham and Winikoff 2005; Sun et al. 2010) defines an agent-based development process of three phases – system specification, architectural design and detailed design – to develop MAS based on a specific agent architecture (BDI architecture (i.e. Belief – the agent’s knowledge of the world; Desire – the agent’s goals; Intentions – the goals that the agent is committed to achieve at certain moment)). Adelfe (Bernon et al. 2011, 2003) is oriented to the development of adaptive MAS, i.e. systems that can adapt themselves to unpredictable, evolutionary and open environments. PASSI (Cossentino 2005) and its evolution ASPECS (Cossentino et al. 2010) focus on agent societies to describe a complete development process from requirements specification to implementation. TROPOS (Ali et al. 2008; Castro et al. 2002) covers the analysis and design phases of MAS development and is based on the $i^*$ requirements elicitation approach (Yu 1997).

Low et al. (2009) identify important common features across 10 extant AOSE methodologies: Gaia, TROPOS, MAS-CommonKADS (Iglesias et al. 1996), Prometheus, PASSI, Adelfe, MaSE (Wood and DeLoach 2000), RAP (Taveter and Wagner 2005), MESSAGE (Garijo et al. 2005) and INGENIAS. Most methodologies have been applied to a variety of problems and are considered domain independent. Only Adelfe and RAP are restricted to adaptive systems or distributed organisational systems respectively. Some methodologies limit the size of the system that they can describe. For example, Gaia and MaSE limit the size of MAS to less than 100 and 10 agents respectively. In terms of development lifecycle, most methodologies support the analysis and design phases while a few also detail the implementation activities (e.g. PASSI, Adelfe or INGENIAS). The development phases are typically conducted iteratively (Low et al. 2009). Another survey (Tran 2006; Tran et al. 2006) of 16 widely used AOSE methodologies (MaSE, MASSIVE (Lind 2001), SODA (Molesini et al. 2008, 2010, 2006; Omicini 2000), Gaia, MESSAGE, INGENIAS, BDMI (Kinny et al. 1996), HLIM (Elammari and Lalonde 1999), MEI (Kendall et al. 1995), Prometheus, PASSI, Adelfe, COMOMAS (Glaser
1997), MAS-CommonKADS, CASSIOPEIA (Collinot and Drogoul 1998) and TROPOS) shows that eleven of them require their models to be developed iteratively. The remaining five (SODA, BDIM, MEI, COMOMAS and CASSIOPEIA) do not specify the process model describing the development lifecycle. However, all those methodologies assume progressive refinement of models gradually, and in stages, culminating with the MAS implementation. That is, their underlying process model is iterative. This is consistent with AOSE experts’ opinions about the most appropriate software development lifecycle that AOSE methodologies should adopt, with the vast majority of experts suggesting that it should be iterative and incremental (Tran 2006). Various reasons were given to support this opinion:

- Complex systems cannot be built at once; functionalities have to be gradually incorporated.
- It is the best development model to prevent risks and facilitate maintenance.
- Iterations allow refinements to be made in an organised and predictable way.
- Increments allow for short delivery cycles and enhance project visibility.
- New agents may appear and others become obsolete as the system continuously evolves.
- MAS are generally more evolvable and dynamic than most other systems, so it is not desirable to freeze requirements.

Most extant AOSE methodologies detail the process to produce an implemented system or a set of design models from scratch. However, some authors have proposed mechanisms to accelerate the development process. For example, Cheah et al. (2012) propose the reuse of task knowledge during the analysis phase of MAS. Certain tasks, such as the retrieval of information from a repository, often reoccur in many MAS development projects. These authors propose the use of design patterns to describe how a typical problem can be solved using concepts specific to MAS, such as roles, goals or
resources. These patterns can then be applied as complements to a number of AOSE development methodologies. Miller et al. (2011) advocate involving the client in the MAS model development as much as possible to avoid errors and therefore reduce production times. In a more technical approach and as part of the outcome of this thesis, Lopez-Lorca et al. (2011) propose a MAS model validation and verification (V&V) add-on process aiming at preventing the propagation of modelling errors from system analysis to later phases of the development process. This, in turn, reduces development times as it avoids rework.

2.2.1 Limitations of AOSE Methodologies

There is a lack of consensus in the AOSE community on standard elements in a methodology amongst the extant proposals, as different methodologies emphasize different aspects of MAS. Some authors have attempted to design a unified AOSE methodology universally applicable based on extant ones (Dam and Winikoff 2012). Other authors, however, argue that such a unified methodology is not possible because of the very different philosophies underpinning the various AOSE methodologies (Low et al. 2009). For example, as earlier noted Adelfe is well suited to modelling adaptive systems. However, it may be inappropriate or even inapplicable for modelling other kind of agents, such as purely reflexive (these agents do not need to define representations or aptitudes) or purely planning agents (Adelfe does not support the definition of plans in the agent internal structure) (Bernon et al. 2011). Another example involves the definition of an agent’s goals. While this concept is central to many methodologies (INGENIAS, BDIM, HLIM, MEI, Prometheus, PASSI, Adelfe, COMOMAS, MAS-CommonMAS or TROPOS) because it models the objective that agents pursue through the execution of their plans and actions, other methodologies (MaSE, MASSIVE, SODA, Gaia, MESSAGE or CASSIOPEIA) do not even model goals (Collinot and Drogoul 1998; Garijo et al. 2005; Lind 2001; Molesini et al. 2010; Wood and DeLoach 2000; Zambonelli et al. 2005). To tackle the lack of uniformity in AOSE methodologies, some research (Beydoun et al.
Background and Literature Review

A literature review (Low et al. 2009; Tran et al. 2006; Tran and Low 2005) has been conducted to identify the concepts and relationships used by the various methodologies to unify notations when possible. Some of the most widely accepted elements used by AOSE methodologies are: Agents and interaction or acquaintance relations between agents (used by all reviewed methodologies); role models (identified in 10 out of 16 methodologies); and organisational structures between agents and their environments (included in 9 out of 16 methodologies). Even when these concepts are common across methodologies, there is no consensus in the definition of models that use them.

AOSE methodologies also vary in their limitations and omissions. Some methodologies, such as Prometheus, Adelfe or PASSI (and ASPECS), restrict the type of agents that can be defined to certain styles, i.e. BDI, adaptive or object-oriented styles, respectively (Bernon et al. 2011; Cossentino et al. 2010; Sun et al. 2010). Others, such as MaSE, MASSIVE, SODA, Gaia or CASSIOPEIA, do not provide enough detail on internal design of agents (Collinot and Drogoul 1998; Lind 2001; Molesini et al. 2010; Wood and DeLoach 2000; Zambonelli et al. 2005). Many methodologies present limitations in the scope of their models. For example, MaSE, BDIM, MEI and Prometheus do not support the definition of agent organisations (Kendall et al. 1995; Kinny et al. 1996; Sun et al. 2010; Wood and DeLoach 2000), i.e. authority relationships amongst roles in the system. However, the evolution of MaSE in O-MaSE (DeLoach 2005; García-Ojeda et al. 2008) supports the description of the MAS organisation. Similarly, Gaia, BDIM, Adelfe, ASPECS and COMOMAS do not define the design of agent interactions in sufficient detail (Bernon et al. 2011, 2003; Glaser 1997; Kinny et al. 1996; Zambonelli et al. 2005). At the process design level, MESSAGE, INGENIAS, MAS-CommonKADS and TROPOS do not clearly define the development phases or the transition between them (Ali et al. 2008; Garijo et al. 2005; Iglesias et al. 1996; Pavón et al. 2005). SODA, HLIM, MEI and CASSIOPEIA have issues with the definition of the models to be created or their notation (Collinot and Drogoul 1998; Elammari and Lalone 1999; Kendall et al. 1995; Molesini et al. 2010).
2.2.2 Validation and Verification Support in AOSE Methodologies

A limitation in the current state of AOSE methods is that there are no mechanisms to assure that a MAS will behave appropriately in a range of situations (Winikoff 2008). Although formal methods that target well-known properties (such as liveness or safety) have been extensively explored, techniques that focus on domain-specific correctness are still lacking (Winikoff 2008). This is evident in the above analysis which highlights that the model representation formalisms used in all the reviewed methodologies, except for CASSIOPEIA (Collinot and Drogoul 1998), enable inter-model consistency checking and half of them explicitly define intra- and inter-model consistency checking rules or guidelines. However, none of them supports the validation of the models against the domain knowledge and user conceptualisation. In fact, although MaSE, MASSIVE, INGENIAS, Prometheus, PASSI, Adelfe and TROPOS claim to provide mechanisms to support the validation and verification of the models, a closer inspection of these AOSE methodologies shows that these mechanisms are limited to structural and domain-independent checking:

- MaSE uses the knowledge modelled in the ontology to verify the type of the parameters exchanged by agents (DiLeo et al. 2002).

- One of the activities defined in MASSIVE involves verifying that the coded artefact complies with the models built and reengineering the implementation in detected case violations (Lind 1999).

- INGENIAS supports verification activities at two levels. It ensures that the structure of the models is correct and that the implementation complies with the models (Pavón et al. 2005).

- Prometheus requires checking the inter-consistency of models, possibly with the aid of automated tools. However, the authors of the methodology do not provide concrete guidelines or tools to support the process (Padgham and Winikoff 2002).
• PASSI supports verification in the sense that the behaviour of the implemented agent should comply with the design-level models (Cossentino 2005).

• The development process used by Adelfe is an adapted version of RUP. The classical description of RUP includes the validation of user requirements as part of the Preliminary Requirements phase and the validation of the UI prototype at the end of the Final Requirements phase. However, Adelfe does not define any mechanism to conduct these activities (Bernon et al. 2003).

• The initial requirements phase of TROPOS can be used to verify that the final product complies with the initial requirements. Also, the formal language used for requirements definition has scope for formal verification of properties (Bresciani et al. 2004).

None of the methodologies identified as supporting the validation and verification of the system provides mechanisms to validate the system against the domain problem as conceptualised by the client. The few AOSE methodologies that include the modelling of the domain knowledge using ontologies as part of their development process are O-MaSE, MESSAGE, PASSI, ASPECS, MOBMAS and MAS-CommonKADS (Cossentino 2005; Cossentino et al. 2010; DeLoach 2005; García-Ojeda et al. 2008; Garijo et al. 2005; Iglesias et al. 1996; Tran and Low 2008; Wood and DeLoach 2000). These methodologies use ontologies to represent the domain knowledge and the agent local knowledge. Additionally, MESSAGE uses ontologies to provide context and input information for agent behavioural knowledge and reasoning. O-MaSE uses ontologies in agent communication to formulate exchanged messages in terms of ontology concepts and their data types. They also assume that, as agents will use heterogeneous ontologies, mappings will be needed to support this interoperability. PASSI and ASPECS also use the ontology during communication to formulate messages in terms of the ontology concepts. Finally, although MOBMAS hints that ontologies could be used to verify the structure of models, to support interoperability and for reuse (Beydoun et al. 2006b; Tran et al. 2007, 2008; Tran and
Low 2008), no explicit mechanism is defined. It seems that ontologies are acknowledged as suitable to represent domain knowledge and are often the chosen option for this task. However, important features of ontologies (discussed in next section) that could lead to the definition of a mechanism to validate MAS models, taking into account the domain specific knowledge, are overlooked.

2.3 ONTOLOGIES BACKGROUND

To understand how ontologies can be used to validate MAS models, this section briefly reviews the concept of ontology, its main components and types of ontologies. The benefits of the application of ontologies are described. The section concludes with an overview of ontology languages and existing tools.

2.3.1 Ontology Definition and Taxonomies

The term Ontology was originally coined in ancient Greece in metaphysics, a major branch of philosophy at the time, to mean ‘the study of what is there’. This included, for example, the question of whether or not there is a god (Hofweber 2011). The Artificial Intelligence (AI) community adopted the term in the 1980s to describe a central component of knowledge systems and theory of a modelled world (Gruber 2009). In the early 1990s, Gruber (1993) defined ontology as an "explicit specification of a conceptualisation". This is the definition adopted in this thesis. With this view, an ontology specifies a domain by defining its objects, relations between them and axioms that govern the domain. The main goal of ontologies is to represent a shared view of a domain. All parties involved in their use agree on the represented conceptualisation. To model this shared view of the world, ontologies define the following main elements:

- **Classes** determine the type of concepts that will be defined in the ontology, i.e. that exist in the domain. For example, a wine ontology used by the World Wide Web Consortium (W3C) in their Ontology Web Language 1.0 guide (W3C 2004a) includes the classes Wine, Region or Grape.
• **Relations** determine the associations that can be held by concepts of a domain. For example, the relations `locatedIn` and `madeFromGrape` defined in the wine ontology of W3C represent the wine origin region and the grape that was used to produce it, respectively.

• **Axioms** model facts that are always true and hence determine the rules of the domain. For example, the wine ontology defines the axioms `Margaux locatedIn MargauxRegion`, `Margaux madeFromGrape MerlotGrape` and `Margaux madeFromGrape max 1 Thing` to establish that the Margaux wine type is produced in the Margaux region and it can be only produced using Merlot grape.

• **Individuals** are instances of classes. They need to comply with the semantics described by the axioms. For example, the individual `ChiantiClassico` is defined conforming to the domain axioms as `ChiantiClassico madeFromGrape SangioveseGrape` and `ChiantiClassico locatedIn ItalianRegion`.

Several approaches for ontology development exist. Uschold and Grueninger (1996) outline a general methodology for designing and evaluating ontologies. They highlight the importance of commencing the ontology design process by identifying the purpose and scope of the ontology. The construction of the ontology itself is broken down into three sub-tasks. First, the concepts and relationships to be represented in the ontology are identified. Second, these concepts and relations are codified in a formal language. Finally, relevant existing ontologies are integrated; this is acknowledged by the authors to be a difficult problem requiring further study. Once the ontology has been built, it must be evaluated and then documented to facilitate future reuse. Along these lines, METHONTOLOGY (Fernández-López et al. 1997) is another widely cited approach for ontology development. This methodology extends the work of Uschold and Grueninger (1996) by providing more detailed guidelines to execute the ontology design tasks. Other authors (Braun et al. 2007; García et al. 2010; Sure et al. 2002) have focused on proposing methodologies to develop ontologies in collaborative or decentralised environments. Finally,
Beydoun et al. (2011a; 2012) proposed a formalised semi-automatic approach to design maintainable ontologies, evaluate their quality and repair modelling errors.

Ontologies are classified depending on how accurate they are at characterising the conceptualisation to which they commit (Guarino 1998). A fine-grained ontology represents a conceptualisation with a high degree of detail and a rich set of axioms represented in a highly expressive modelling language. This is costlier to develop and process than a coarse-grained (or lightweight) ontology which typically contains a minimal set of axioms, requires a less expressive modelling language and is easier to develop and process (Guarino 1998). Ontologies can also be classified according to their genericity. An upper level ontology represents domain independent concepts. A domain ontology focuses on the upper level ontology concepts to conceptualise a specific domain (Guarino 1998). Examples of upper level ontologies are Cyc or DOLCE (Mascardi et al. 2006). Cyc seeks to model common sense knowledge applicable to a broad range of domains (Matuszek et al. 2006). DOLCE aims at capturing the ontological categories underlying natural language (Gangemi et al. 2002). In contrast, the Gene Ontology (Ashburner et al. 2000) and the Foundational Model of Anatomy Ontology (Rosse and Mejino 2003) are examples of domain ontologies in the fields of genomics and human anatomy respectively.

2.3.2 Benefits of Ontologies

Ontologies are more expressive than typical software models and can be processed by machines and humans, while models commonly target either humans or computers (Happel et al. 2010). Unlike models that are typically prescriptive, ontologies are descriptive and focus on the analysis of the domain (Henderson-Sellers 2011). The use of ontologies has many potential benefits (Büerger and Simperl 2008). Due to their expressiveness, their use provides an opportunity for automatic reasoning to infer implicit knowledge and detect inconsistencies; models lack the underlying semantics necessary to enable this to occur. Ontologies usually represent a widely accepted truth
while software models are intended for a particular project (Happel et al. 2010). Models assume that anything not represented in them is false. Ontologies operate under the assumption of an open world: the falsehood of unknown facts is not assumed (Henderson-Sellers 2011). Ontologies also share similarities with database schemas (Fonseca and Martin 2007), however there are important differences (Fensel 2004). Ontology languages are syntactically and semantically richer than common database languages. They represent shared and agreed upon knowledge as they are used to share and exchange information. Therefore, ontologies are appropriate to semantically integrate heterogeneous data sources. They do not only provide the structure for a data container, rather they define a domain theory. Benefits of using ontologies according to Gruninger and Lee (2002) fall into three main areas:

- **Communication.** Ontologies can assist in ensuring interoperability between software entities at data and process levels. They can also help to remove ambiguity relating to the meaning of concepts in a certain domain. Finally, their underlying formal semantics prevent unwanted interpretation, hence facilitating knowledge transfer.

- **Reuse.** Ontologies can be used to develop systematic widely accepted domain descriptions. Once an ontology is available for a certain domain, it can be reused for similar developments. This avoids the costs of developing new models and increases the quality of the final product.

- **Inference.** Ontologies are typically developed using a logic based language (e.g. Description Logics). These formal semantics permit the derivation of implicit facts from the explicitly stated knowledge. Also, knowledge models do not require a commitment to any particular design or implementation (i.e. they are independent of the implementation technology). Therefore, knowledge models are sufficiently abstract to allow the automatic generation of code to derive instantiations that suit particular architectures. Finally, inconsistencies in the modelled knowledge can be detected to prevent errors propagating
to later phases of software development. This relies on the 
derivation of knowledge that is implicitly defined in the 
semantically rich knowledge represented in the ontology, and 
requires the use of specialised software components known as 
reasoners (e.g. Pellet (Clark and Parsia 2012), FaCT++ (2012) or 
Hermit (Motik et al. 2012)). To illustrate this, consider a well-
known ‘pizza ontology’ (described in Horridge et al. (2007)) 
that models different types of pizzas, where the concept Ve-
getarianPizza is a subtype of Pizza. It defines the axioms Veget-
arianPizza not hasTopping MeatTopping and VegetarianPizza not 
hasTopping FishTopping. Another subclass of Pizza is the Quattro-
Formaggi. This concept is defined by the axioms QuattroForm-
aggi hasTopping TomatoTopping and QuattroFormaggi hasTopping 
FourCheesesTopping. A reasoner that automatically classifies the 
classes of the ontology finds that:

- TomatoTopping is subclass of VegetableTopping.
- QuattroFormaggi is subclass of CheeseTopping.
- VegetableTopping is not a subclass of MeatTopping.
- VegetableTopping is not a subclass of FishTopping.
- CheeseTopping is not a subclass of MeatTopping.
- CheeseTopping is not a subclass of FishTopping.

Then, the reasoner derives the new knowledge that:

- QuattroFormaggi is a VegetarianPizza.

Whilst simple, the example illustrates automatic reasoning 
and classification capabilities of ontologies. This underpins 
their potential, particularly as reasoning processes become 
more complex and powerful. For example, the property chains 
mechanism (W3C 2009b) enables the composition of properties 
of arbitrary length. As the properties hasFather and hasBrother 
are both within the domain and range of the class Person, the 
property hasUncle could be defined as the composition of the 
sequence of properties hasFather and hasBrother. The reasoner 
would infer that if individual X hasFather Y and Y hasBrother Z, 
then X hasUncle Z.
2.3.3 Ontology Tools and Languages

Tools and standardised language support to create and modify ontologies are required to harness the full benefits and potential that ontologies provide (W3C 2009c). Since the early 1990s, a number of ontology languages have been proposed to construct ontologies (Corcho et al. 2006). CycL is the language used to encode the Cyc ontology. It was initially created as a frame language, but it has evolved to become a declarative language (Lenat and Guha 1991). Ontolingua (Farquhar et al. 1997) extends the Knowledge Interchange Format (KIF) (Genesereth and Fikes 1992), which is based on first order logic, with frames theory. The high expressiveness of Ontolingua hampered the development of reasoning mechanisms (Corcho et al. 2006).

With the role of ontologies in the next generation of the World Wide Web (WWW) (The Semantic Web (W3C 2012a)) receiving added attention, a standard language to develop ontologies has recently been developed. This Web Ontology Language (OWL) was developed by the W3C (2004a), which is the main international standards organisation for the WWW. OWL is envisioned to extend the current WWW by encoding information in a way that is machine-understandable (Berners-Lee et al. 2001). Ontologies will enable web-based browsing of various databases for the same topic using semantic links between the databases. This functionality is illustrated with a life sciences example. Researchers belonging to different disciplines (e.g. genomics, proteomics, clinical drug trials, epidemiology) need a mechanism to integrate their heterogeneously defined databases. Biology, genomic and medicine communities are already developing ontology language standards to facilitate data integration (OBOFoundry 2012). Among other initiatives, the governments of the United Kingdom (UKGovernment 2012) and United States of America (USAGovernment 2012) are making non-personal public data available to encourage its free reuse.

There are several open-source applications for ontology editing, such as SWOOP (Kalyanpur et al. 2005) and NeOn-Toolkit (Haase et al. 2008). The most widely used OWL editor is Protégé (2012), an open-source tool developed at the University of Stanford. Reasoners
complement ontology editors with automation capabilities. Several reasoners have been developed for OWL DL. The most important reasoners are FaCT++ (2012), Pellet (Clark and Parsia 2012) and Hermit (Motik et al. 2012). While FaCT++ and Pellet perform the reasoning based on tableau calculus, Hermit uses hypertableau calculus. A comparison of reasoning times (Motik et al. 2009) showed that none of the reasoners display clear superiority in their implementation, however it was noted that the ontology structure affects greatly the performance of the reasoner.

Concomitant to OWL, W3C proposed a group of markup ontology languages (W3C 2012b). The Resource Description Framework (RDF) (W3C 2004b) was developed as an extension of the Extensible Markup Language (XML) (W3C 2008a) to describe web resources. RDF is based on the idea of triples, i.e. statements about resources in the form of subject + predicate + object. RDF vocabulary includes constructs such as Resource, Property, Statement, subject, predicate, object or type. RDF was further extended with RDF Schema (RDFS) (W3C 2004c), which provides constructs to define ontologies. Examples of RDFS constructs are Class, Literal, domain, range or subClassOf. The Web Ontology Language (OWL) was actually proposed based on RDFS to augment its expressivity. W3C defines the layered structure of the Semantic Web (W3C 2012b) (as shown in Figure 2.1), with ontologies and their technologies playing a fundamental role. They will be the central integrators of heterogeneous data drawn from diverse sources (Shadbolt et al. 2006). OWL 2 (W3C 2009a) is a revision and extension of the original OWL and the W3C current recommendation (W3C 2009d). Some of the features that make OWL 2 more expressive than OWL 1 are property chains, qualified cardinality restrictions or asymmetric, reflexive and disjoint properties (W3C 2009b).

OWL 2 offers two alternative ways of assigning meaning to ontologies: OWL 2 DL and OWL 2 Full (W3C 2009c). The former provides a meaning for ontologies in a Description Logic style, while the latter is based on interpreting ontologies as RDF graphs. In OWL 2 Full, everything is a RDF triple and the language is fully reflective. However, it is undecidable, i.e. there is no reasoner that can return an answer for any possible query. As OWL 2 DL is a syn-
tactically restricted version of OWL 2 Full, the implementation of fully decidable reasoners is possible. OWL 2 defines three profiles in addition to the DL and Full versions (W3C 2009c). All of them have been designed as computational approachable subsets of OWL 2, fulfilling the needs of existing communities. OWL 2 EL is suitable for applications needing very large ontologies, such as in biohealth sciences. These ontologies typically have a large amount of complex classes that need to be classified and applied to vast amounts of data. OWL 2 EL disallows negation, disjunction, inverse properties and universal quantification (W3C 2009c,a). OWL 2 QL is suitable for applications in which a large number of individuals has to be organised according to relatively lightweight ontologies, and data has to be accessed directly via relational queries. It can represent key features of Entity-relationship and UML diagrams, thus it is useful for representing database schemas and for integrating them via query rewriting. OWL 2 QL disallows negation, disjunction, property chain axioms and existential qualification of roles to a class expression (W3C 2009c,a). Although OWL 2 RL is similar to OWL 2 QL in the sense that it is suitable for lightweight ontologies classifying a large number of individuals, it is oriented to operate data in the form of RDF triples rather than via SQL queries. It facilitates the
enrichment of RDF data using rule-based technology and provides a partial axiomatisation. It disallows negation, disjunction or statements where the existence of an individual enforces the existence of another (W3C 2009c,a). Today, the strong support of the W3C for the markup ontology languages (W3C 2009d) and the number of development tools available (W3C 2009c) is influencing a shift in research efforts from the classic ontology languages to markup languages.

The next section discusses another use of ontologies that has received significant attention in recent years: the application of ontologies to facilitate various phases of the software development cycle, and the improvement of the quality of its workproducts (Beydoun et al. 2006b).

2.4 ONTOLOGY-DRIVEN SOFTWARE DEVELOPMENT

Guarino first made the distinction between ontologies applied at runtime (Ontology-driven Information System) and at development time (Ontology-driven Information System Development) in 1998 (Guarino 1998). Most software development processes share a core set of activities that are organised into phases (Tsui 2009). In the Analysis phase, analysts extract the system requirements from the customer and build the analysis models to understand the problem. In the Design phase, designers think in terms of the solution to define the architecture of the software without concerning themselves with low level operational details. In the Codification phase, programmers materialise the architecture defined in the previous phase using a concrete programming language. In the Testing phase, the different artefacts produced during the software development are verified and validated to assure their quality and compliance with the original requirements. Once the product has been delivered to the client, the last phase, Maintenance, is commenced. In this phase, errors are identified and fixed, and new requirements are added to the system.

The analysis phase, which is the focus of this thesis, is commonly known as the requirements engineering phase, and because of its importance it can be further divided into sub-tasks. The following
activities typically constitute most existing requirements engineering processes (Sadraei et al. 2007):

- **Elicitation**: This identifies high level goals of the target system, requirements for different groups of users and the tasks to be accomplished, along with system boundaries (Nuseibeh and Easterbrook 2000).

- **Requirement Analysis**: This analyses the requirements to uncover conflicts, ambiguities, and missing or duplicate requirements in order to identify alternatives and convert them into a structured and unambiguous representation.

- **Negotiation**: This selects between trade-offs to achieve agreement between stakeholders (Lamsweerde 2000).

- **Verification and Validation**: This examines requirements to find any deficiencies in consistency, accuracy and adequacy (Sommerville and Ransom 2005). It may also include a feasibility analysis to verify the cost of development.

- **Change Management**: This recognizes changes through continuous requirements elicitation, re-evaluation of risk and evaluation of the system in its operational environment (Nuseibeh and Easterbrook 2000), to ensure that all relevant information for each change is collected.

- **Requirement Tracing**: This manages the evolution of requirements, maintaining traces about its history to track the origins of each requirement, so that if a change has to be made to a design component, the original requirement can be located (Davis 1993).

The main focus of the research presented in this thesis is the validation and verification activity in the requirements engineering phase for MAS (with beneficial impacts for other tasks as later discussed). *Requirements Verification* is concerned with proving that every requirement has been satisfied by the system, whereas *Requirement Validation* seeks to ensure that the set of requirements is complete and consistent, that a model can be created to satisfy the requirements,
and that a real solution that complies with the requirements can be built and tested (Bahill and Henderson 2005). The work presented in this thesis is concerned with ensuring that the correct system is being developed and that the structure of models is correct. Bahill and Henderson (2005) state that it is not possible to prove that a set of requirements is complete, and that even if it were possible, it would probably be too costly. However, it is often possible to detect incompleteness in a set of requirements to some extent. The research presented in this thesis seeks to define a mechanism to identify incompleteness in models and to complete them to a certain degree. The rest of this section describes typical uses of ontologies in development as found in the literature, with a focus on the existing application of ontologies in requirements engineering.

2.4.1 Roles of Ontologies in Software Development

This section first details the roles that ontologies play when used to improve the software development process: consistency and completeness checking, facilitating understanding, integration and reuse. These roles will then be used to categorise related works. The details of the roles are:

- Ontologies typically have formal semantics that are based on logic (Corcho et al. 2006). This makes it possible to automate important modelling tasks such as consistency and completeness checking. Consistency checks (Sirin et al. 2007) avoid inconsistent use of knowledge, i.e. to assert truth and falsehood of a given fact at the same time. Automated consistency checking enables detecting such implied assertions through a chain of logical steps. Consistency checking can also make explicit any knowledge that is implicitly implied or modelled. Ontologies can therefore be used in this way to complete (Kaiya and Saeki 2005) models acting as reference frames to identify modelling gaps.

- By formally defining all relations in a domain, ontologies can facilitate understanding (Graja et al. 2011) of the system that is
modelled. They deconstruct the problem by splitting it into fragments that can be classified, which helps to comprehend the underlying structure. Annotation (Good and Su 2011) can then be used to augment the concepts of an ontology with metadata that describes them and gives further information. These annotations, combined with formal relations and structural knowledge, further facilitate subsequent complex queries against the knowledge represented by the ontology. Such semantic queries (Ray et al. 2009) harness, amongst others, the hierarchy and equivalence relations defined in the domain as described by an ontology.

- The formal mechanism that makes semantic queries possible also enables the integration (Paulheim 2009) of heterogeneous systems, i.e. facilitating that different systems interoperate and share knowledge. These systems are often built using very different technologies; ontologies are technology-independent knowledge modelling artefacts, therefore their modelled knowledge can be used in a wide range of technologies.

- Ontologies separate domain knowledge from operational knowledge. This facilitates reuse (Uschold et al. 1998) at two levels. The ontology itself can be reused as a shared knowledge conceptualisation, and it can facilitate the reuse of the artefacts that it describes.

2.4.2 Uses of Ontologies during Requirements Engineering

Some authors have considered guiding the requirements process as a whole by using an ontology to describe its metamodel (Kossmann and Odeh 2010), however most existing works focus on individual tasks of the process. Table 2.1 summarises the works according to the role that the ontology plays and the task of requirements engineering that is supported.

Caralt and Kim (2007) propose a methodology to reuse use cases in requirements elicitation. They use an ontology to annotate use cases with semantic information and to augment the queries used to re-
Table 2.1: Ontologies used during requirements engineering

<table>
<thead>
<tr>
<th>Ontology Role</th>
<th>Consistency &amp; Completeness</th>
<th>Understanding</th>
<th>Integration</th>
<th>Reuse</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Elicitation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Negotiation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Validation &amp; Verification</strong></td>
<td>Kaiya and Saeki (2005), Benevides and Guizzardi (2009), Benevides et al. (2009), Decker et al. (2005), Lin et al. (1996), Shanks et al. 2003, Pires et al. (2011), Goknil et al. (2011), Jekjantuk et al. 2010, Fanmuy et al. (2011) and Siegemund et al. (2011)</td>
<td>Kaiya and Saeki (2005), Benevides et al. (2009), Decker et al. (2005), Lin et al. (1996), Shanks et al. 2003, Pires et al. (2011), Goknil et al. (2011), Fanmuy et al. (2011) and Siegemund et al. (2011)</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>Change Management</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meng et al. (2006)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Documentation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hyland-Wood et al. (2008)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
retrieve case studies, providing a better capture, reuse and query of use cases. Although it is not the main focus of the proposal described in this thesis, the ontology by-product of the models validation process can be reused in other projects in similar domains. In Vongdoiwang and Batanov (2006), the authors discuss the use of ontologies as intermediate models to facilitate requirements elicitation from textual descriptions. They argue that their framework could be used to implement the transformations necessary to semi-automatically produce a UML class model from a textual description of a problem. However, the framework is only sketched and the development of an applicable process using the described models is only outlined as future work. The research presented in this thesis has the potential to elicit hidden requirements through reasoning about implicit knowledge. This is significantly different from the work of these authors, as their ontology only extracts explicit requirements from textual specifications.

Ontologies have also been used to integrate stakeholder views in requirements engineering. For example, Lee and Gandhi (2005) developed an ontology-based framework to assist the elicitation and analysis of requirements, in particular the integration and understanding of different stakeholders’ points of view. In their framework they include contributions of several requirements engineering techniques, such as goal-driven scenario composition, requirements domain model, viewpoint hierarchy and other domain specific taxonomies to hierarchically organise and integrate the application domain concepts, properties and their relationships. The proposal of this thesis goes a step further and not only uses the ontology to structure the requirement analysis elements, but also to make explicit knowledge that had been only implicitly modelled. Similarly, Assawamekin et al. (2008) propose a method to seamlessly integrate the views that different stakeholders provide in requirement elicitation. The authors use the original requirements expressed in plain English to build ontologies. Then, using matching mechanisms they identify synonyms to unify concepts. Similarly, the work described in this thesis uses an ontology to uniquely model the different points of views that each software model represents.
The use of ontologies to improve general quality of requirements specification, traceability or reuse has also been documented in Dietz (2005). This work proposes a system ontology as a means to reduce errors in the requirements engineering and system specification phases of software development. The ontology is introduced using a rigorous and fully systematic notation. In the process proposed by this thesis, the ontology acts an intermediary validation element to ensure that all models are consistent with each other. The idea of using ontologies to validate requirements models is not new; notable examples are presented below. However, none of them address the multi faceted usage of ontologies: consistency checking, completeness and correctness. Moreover, identified ontology-based activities often seem to be tied to a specific language, e.g. (Benevides et al. 2009; Shanks et al. 2003).

Through the use of a thesaurus, the requirement specification and first order logic mechanisms are used to detect incompleteness in the specification and to predict extensions to the software (Kaiya and Saeki 2005). These authors extend their work in Kaiya et al. (2010) by adding general knowledge to the domain ontology through web mining mechanisms. In this thesis, the domain ontology is enriched with structural features of the development paradigm. This enables the detection of both inconsistency and incompleteness issues with regards to the domain and the elements and relations of the development paradigm. In a number of studies (Benevides et al. 2009; Benevides and Guizzardi 2009; Shanks et al. 2003), ontologies are used to provide verification of models built in a specified language (e.g. OntoUML). The research presented in this thesis does not force developers to use any modelling language; rather it adapts its operators. This proposal is independent of the modelling language, as the defined operators can be adapted to a range of models. This makes the proposal applicable in different development methodologies. To illustrate this, the approach of the thesis is validated in development projects using two different MAS development methodologies.

In Jekjantuk et al. (2010), the authors use OWL ontologies to verify that models are compliant with their metamodel. However, no validation mechanism is provided to check compliance with the client’s specification, so only the correctness of the structure
of the models can be assured. Another limitation of the proposal is that the models have to be expressed in OWL. In contrast, the proposal of this thesis uses an OWL 2 ontology to validate and verify models expressed in various notations and to identify potential completeness issues. A recent survey (Fanmuy et al. 2011) reviews current industrial requirements verification practices. The authors focus on natural language description of requirements to find that common problems are ambiguity, inconsistency and incompleteness issues. Inspections are often used to verify and validate requirements, despite their high demands in terms of effort and time. The authors advocate for the use of ontologies together with natural language processing techniques as a promising hybrid approach to write high quality requirements and avoid rework. Although this thesis focuses on structured requirements models, the use of ontologies could benefit the requirements engineering activities in general. In the same line of natural language specifications, Pires et al. (2011) propose an ontology-based iterative and incremental requirements engineering process to assist in the control of requirements volatility. The ontology represents the domain and it is used as a consistency assessment tool. Requirements expressed on a controlled natural language are transformed into an OWL ontology. Reasoners are then used to check its consistency and infer new relations. However, unlike the work conducted for this thesis, they only define is a and part of relations, thus reducing the expressivity of the ontology. The reasoning process presented in this thesis is richer as it can be used to validate any kind of relation as long as it is defined in the metamodel. Also, it supports the completion of the requirements not only by inferring new subsumption relations but many others defined through chains of composite relationships.

Goknil et al. (2011) highlights the importance of requirements inter-dependencies to manage change propagation within the requirements document. They define a metamodel that describes the types of relations that exist in requirements documents, namely requires, refines, partially refines, contains and conflicts. Their metamodel is built on top of a layer of first order logic semantics that makes it possible to check requirements consistency and to infer new relations. The work presented in this thesis also defines a metamodel to V&V
analysis models while taking into account the client conceptualisation and inter-dependencies of the models. Siegemund et al. (2011) present an ontology-driven requirements engineering process. The authors build an ontology defining a requirement metamodel for Goal Oriented Requirements Engineering (GORE). The metamodel includes the elements that define the GORE process as classes and relationships. Each particular project would instantiate the domain independent metamodel. This infrastructure makes it possible to check the consistency and completeness of the requirements. Consistency rules are defined as ontology axioms and are automatically checked by the reasoner. The authors acknowledge the complexity of assuring the completeness of a requirements specification. Thus, they limit the scope of their proposal to define SPARQL rules that verify the requirements are complete in terms of metadata, e.g. every requirement contains a field defining its priority or conflicts with others. While this work bears a certain similarity to this thesis proposal, there are essential differences between them. Both proposals define a metamodel of a development process that can be instantiated for particular projects. In both cases this is the cornerstone of the requirements validation. However, the metamodel developed for this thesis defines the concepts and relationships used in the models (well structured) of a MAS development methodology while theirs models a general requirements specification (mostly textual descriptions) for GORE. While the consistency rules are similar in both cases, the completeness rules developed as part of this research are different. This thesis acknowledges the inherent difficulty of achieving absolute completeness of requirements specification (some authors argue that it is impossible (Zowghi and Gervasi 2002)), but at least the domain ontology enables the mechanism to detect and fix some of the incompleteness in terms of the client conceptualisation, rather than only in terms of requirements structure.

Some authors have considered the application of ontologies to software maintenance or to the adaptation of legacy systems. In Meng et al. (2006), an ontology-based method to support software maintenance is proposed. They argue that software comprehension is essential for its maintenance, thus, they model the comprehension process by means of an ontology. Reasoning mechanisms provide
the user with suggestions of tools suitable for certain comprehension tasks. The research presented in this thesis also facilitates the comprehension of the problem by classifying the domain concepts into their development element categories. Daga et al. (2005) tackle the problem of renovation of legacy systems from a technology-independent perspective. They focus on using ontologies to recover business knowledge, which is an important asset for the company and is often forgotten when legacy systems are adapted. Ontologies provide improved semantics, better interoperability, less complexity and technology independence. The proposal of this thesis was not originally intended for the renovation of legacy systems. However, the high quality ontology resulting from the validation process can be reused for developing applications in similar domains. As the ontology is technology independent, it could be applied to projects dealing with different development technologies.

A number of authors focus on using ontologies to facilitate the traceability of requirements to later phases of the development process, for example (Biffl et al. 2008; Decker et al. 2005; Lin et al. 1996; Noll and Ribeiro 2007). Lin et al. (1996) use first order logic to describe a formal ontology to model requirements. Their approach seeks to maintain traceability between requirements and their refined versions, detect inconsistencies and track changes. Decker et al. (2005) advocate the use of Wikis as self-organised, reusable, requirements documents. Domain ontologies semantically annotate the pages of the Wiki to manage the organic growth implied by the Wiki approach and to trace requirements. Reasoning capabilities of ontologies are used to detect inconsistencies between the pages of the Wiki. Noll and Ribeiro (2007) extend the Unified Process model to include ontologies as a means to integrate the different views of the system and to maintain traceability among artefacts throughout the software lifecycle. Finally, Biffl et al. (2008) introduce an ontology-supported component-based systems engineering approach for the production automation domain (e.g. car mounting chains) that explicitly describes stakeholder quality requirements and traces design decisions to generate new systems and software versions that implement those requirements. The ontology is expected to reconcile views from all roles involved to improve quality assurance of systems requirements.
Unlike all these works, the work presented in this thesis does not focus on requirement tracing. However, traceability is supported to some extent in the form of change propagation. Changes in models are reflected in the domain ontology, which triggers the modification of interdependent models.

2.4.3 Uses of Ontologies beyond Requirements Engineering

Outside the knowledge-based and agent systems community where the use of ontologies at run-time is common (e.g. (Nardin et al. 2008; Wallace et al. 2011)), the use of ontologies beyond analysis has been largely confined to the production of better indexing mechanisms to facilitate reuse of existing components. In other words, the ontologies are not used to create the components, but rather to more easily identify which components to reuse. The ontology itself does not contribute to the creation of the reused models, unlike the work presented in this thesis. Notable examples of this are: Happel et al. (2010) highlight that ontologies can play an important role in building repositories of contextualised development errors to generate test cases for software. Henninger and Ashokkumar (2005) use ontologies as an indexing mechanism to identify suitable design patterns and components to reuse. Happel et al. (2006) propose an ontology-based software components repository augmented with metadata and domain knowledge to enable more complex queries to more precisely retrieve the most suitable component. Khemakhem et al. (2007) present an ontology-based search engine for software components. Antunes et al. (2007) describe a software development reuse platform based on representation and domain ontologies. Keivanloo et al. (2010) propose an automatic Internet crawler that finds reusable fragments of code, automatically annotates them with regards a programming concepts ontology and enables a semantic code reuse system. Eberhart and Agarwal (2004) present a tool that semantically enriches APIs to facilitate rapid application development. All these works focus on reusing artefacts with different granularity (from functions and parameters to components) at codification level; this thesis is concerned with reusing the complete description of the
domain at analysis level. The work presented in this thesis also uses ontologies to improve the quality of and give an added value to an artefact (the domain specification) with structural knowledge.

In Wieschnowsky and Paulheim (2011), the authors argue that although ontologies are useful to integrate applications, often software developers lack the knowledge representations skills to model the ontology and mapping rules. To close this gap they present a tool that enables developers to use a graphic notation to design the mapping rules that will be translated by the tool into ontology rules. In the proposal of this thesis, the development paradigm metamodel includes rules defined as OWL 2 axioms. This metamodel is intended to be reused in different projects, hence, developers do not have to address the design of complex axioms.

Another group of authors use ontologies to integrate various activities across the software development lifecycle. Falbo et al. (2003) propose a software process framework based on a process ontology. The ontology permits the choice of the activities to be conducted within each development phase and the knowledge that is relevant in each phase. It also facilitates the integration of tools in the development process. The proposal of this thesis also formalises the structure of artefacts, but at a different granularity level. This thesis zooms in on the structure of software models localised within the analysis phase, while Falbo et al. (2003) formalise the phases of software development. The research conducted for this thesis focuses on MAS and uses the domain ontology to validate the analysis models rather than to guide the development. Hsieh and Lu (2006) propose an ontology-based method to facilitate collaboration between the domain expert and software developer. They decompose the domain problem into cohesive fragments that they refer to as analysis units. These analysis units have associated ontologies that describe their important concepts. The ontologies act as input and output workproducts and as an interface to integrate different analysis units. Each of these analysis units can be designed and codified independently of the others, as they are self-contained in terms of functionalities and involved knowledge. Once all the analysis units have been implemented, they are integrated using the ontologies. In a similar way to the approach presented in this thesis, this work
defines an iterative and incremental process where ontologies evolve over time. Both proposals decouple the domain knowledge from the software development process, and process ontologies as inputs to produce them as outputs. This thesis seeks to facilitate the collaboration between the domain expert and software developer and to provide a detailed validation mechanism. In contrast, these authors roughly describe how ontologies could guide analysis, design and codification. Another difference is that in this thesis the domain problem is not divided into several ontologies covering the sub problems. Only a single ontology is defined to minimise redundancy and facilitate change propagation in interdependent models. The elements that benefit from ontologies in this case are smaller and more localised: the models developed within the analysis phase of the software development.

Closer to the work in this thesis are works which advocate the use of ontologies to produce improved artefacts during the software development lifecycle. For example, Ahmed et al. (2011) automatically generate ontologies from database schemas. The authors define mappings to generate the conceptual model from the database schema as an intermediate artefact and then as a corresponding ontology. Additionally, the authors define a mechanism to support changes in database schemas to be propagated to the ontology while preserving any modification manually applied to the ontology. The research described by this thesis also provides a mechanism to propagate changes from the ontology to software models and among interdependent models. In a related line of work, Vongdoiwang (2010) proposes using ontologies to semi-automatically generate UML class diagrams. The ontology editor Protégé is used to merge two or more ontologies that represent the problem domain and to automatically transform the merged ontology into a class diagram. Although this work might be an interesting example of ontologies applied to facilitate design activities, the author only sketches the process. Significant aspects such as complexity of merging ontologies or the quality of the resultant class diagram are not discussed. Similarly, Wenzel (2011) proposes a framework to facilitate the development of ontology-driven systems. Following the philosophy of model-driven engineering, the author defines a mechanism to map RDF resources
to Java classes. As ontologies are associated with the software models that are derived from them, powerful semantic queries can be used to retrieve different artefacts. The software model idea is also central in the approach presented in this thesis. However, this thesis does not focus on defining transformations between them. Its goal is to produce high quality analysis models that, if deemed necessary, could become the input for a model-driven development process. Casagni et al. (2011) propose a semantic Wiki to migrate the Italian public administration documents to electronic format. Every page of the Wiki is associated with elements of the ontology that define the structure of the documents and the business processes that involve them. Similarly to the work described in this thesis, these authors also use an ontology to provide structure to other artefacts. Unlike the work of Casagni et al. (2011), in this thesis the metadata is a tool to achieve the validation of software models, rather than being the final goal. Hyland-Wood et al. (2008) present an ontology-oriented methodology for the maintainability of software systems. They encode metadata about requirements, code, tests and metrics in RDF/OWL web-accessible repositories. Their methodology is intended to improve the traceability and maintainability of software documentation. Similarly to these authors, in this thesis metadata is also used to enrich ontologies. However, the goals are different. While they focus on improving the quality of documentation, the work of this thesis uses the augmented ontology to validate the structure and content of the requirement models.

2.5 ONTOLOGY-DRIVEN AOSE

Section 2.4 discussed the application of ontologies at development-time for general software developments. This section first examines specific uses for the development of MAS and then focuses on the V&V of MAS.
2.5.1 Ontologies Applied to AOSE

As discussed in Section 2.2, a significant number of methodologies have been proposed to develop MAS, but few of them use ontologies at development-time. In Cossentino et al. (2010), an ontology is developed in the early stages of analysis to describe the problem domain concepts and to complement the requirements description in terms of use cases. This deepens the understanding of the problem, conditions some of the later models and may provoke changes in the use cases. This ontology is extended at design time using solution related knowledge, namely, information about agents and their valid interactions. The ontology development is iterative and incremental in nature as new details may have to be added during the MAS design activity. This ontology is necessary to identify the agents of the system and especially to model their interactions. The way that this methodology uses ontologies is similar to the proposal of this thesis in the sense that it can assist in understanding the domain, the ontology can change as the development proceeds and it can be used to identify key concepts of the problem. However, the authors do not define any mechanism to validate the resultant models. The work presented in this thesis uses ontologies to validate the models in high detail against the client’s conceptualisation, not only verifying the types in agent interactions. Tran and Low (2008) propose integrating ontologies in the software development lifecycles to produce systems whose components are interoperable and reusable. Ontologies are identified to model domain specific knowledge that agents will need to perform their assigned tasks. They are associated with the corresponding resources in the resource model and the corresponding agents, as beliefs, in the agent model. Additionally, it is decided whether the agents will have direct access to the ontologies or whether there will be a specific ontology manager agent. This earlier work was used in the development of workproducts, in particular in the detailed design phase, in Beydoun et al. (2011b) where an ontology-based MAS for the domain of a peer-to-peer (P2P) information sharing community is developed. Ontologies are built and used at development-time to create the models and at run-time to exchange information between agents. However,
although the authors claim that ontologies can support the verification of the produced MAS, including the structure of the models (in terms of correctness and completeness) and the contents of the messages exchanged by agents, no explicit mechanism is defined to detail the necessary steps, as eluded to in Section 2.2. Similarly, Hadzic et al. (2009) propose a methodology to develop MAS that relies on ontologies to represent the knowledge domain and facilitate agent communication by providing a common vocabulary. They also state that ontologies could be used to analyse and manipulate information consumed by agents, for example to detect redundant and inconsistent information. However, the authors do not detail how to perform these activities.

Some other works attempt to use ontologies to improve specific aspects of MAS development (Table 2.2). Some focus on the process itself. For example, by designing a reusable ontology allowing complex queries on the domain of “MAS development”, Girardi and her colleagues (Girardi and Leite 2008) propose an ontology-based multi-agent development process that can model all the phases of development of MAS. This proposal facilitates the design of a customised MAS development process by developers but does not provide mechanisms to validate the product of any development phase, in particular analysis (which is the focus of this thesis). In another work, Nyulas et al. (2008) present an architecture to develop and deploy end-to-end solutions for MAS. They focus on the deployment steps of the system, while the purpose of this thesis is assuring that the deployed system is consistent with the requirements specification.

Sensoy et al. (2010) and Nicoletta and Colombetti (2009) use OWL 2 ontologies to define the policies that rule agent behaviour. Guizzardi et al. (2007) use the foundational ontology UFO to formally model the concept of Goal with all of its associated relationships for the field of MAS. The UFO ontology was defined by Guizzardi and Wagner (2005) and models the foundational concepts on which MAS are based. Similarly, others (van Riemsdijk et al. 2009) have established the logical foundations for conflicting goals in MAS. All these authors have focused on formally defining the semantics of MAS. Although this theoretical approach is important, it is not sufficient. In this thesis a formal definition of MAS is complemented
### Table 2.2: Ontologies used for MAS development

<table>
<thead>
<tr>
<th>Ontology Role</th>
<th>Consistency &amp; Completeness</th>
<th>Understanding</th>
<th>Integration</th>
<th>Reuse</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Codification</strong></td>
<td>–</td>
<td>Hajnal et al. (2007)</td>
<td>–</td>
<td>Hajnal et al. (2007)</td>
</tr>
<tr>
<td><strong>Support</strong></td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>DeLoach (2005)</td>
</tr>
</tbody>
</table>
by practical approach to validate MAS analysis models. Hajnal et al. (2007) present a Protégé plug-in to develop MAS. They focus on the health domain, particularly on home care systems. They define and use ontologies to generate the JADE architecture for a MAS. As the ontologies are based on the K4Care system the generated code is only valid for this platform. The research conducted for this thesis does not have that limitation. It defines a validation and verification process adaptable to any MAS development methodology.

### 2.5.2 Ontologies as a Validation Vehicle in AOSE

The non-deterministic nature of MAS makes it difficult to design test cases for which correctness can be assured (Nguyen et al. 2010). Because of this, many authors have used complex logic engines to check classic logic properties on the design of MAS, for example to assure that all the states defined by an agent state machine are reachable. Recent examples of this line of research are (Boureanu et al. 2010; Broerse 2010; Dastani and Jamroga 2010; Jones and Lomuscio 2010; Lomuscio et al. 2010). Following this line, the work of Montali et al. (2011) extends an existing MAS development methodology, TROPOS (Bresciani et al. 2004), with business process knowledge expressed in a logic language to enable verification and completeness of temporal constraints. Dam and Winikoff (2011) define verification mechanisms for the maintenance phase of MAS lifecycle, focusing on the propagation of changes in MAS design models. The authors define a mechanism that automatically generates repair plans based on a metamodel for Prometheus and constraints expressed with OCL (Warmer and Kleppe 1999). The constraints express assertions about Prometheus that should be true at any moment at design time. Constraints are analysed to formalise all aspects that can be violated. Based on this analysis, repair plans tackling all of them are proposed. When actual errors occur, the system suggests the corresponding plans to the user and waits for their input. Similar to this work, this thesis also ensures that the structure and constraints of models is correct. However, the focus of the thesis is the early stages of the MAS lifecycle: the requirements analysis. Another important difference is
that this thesis does not attempt to propose solutions to the errors detected in the models. Dam and Winikoff (2011) acknowledge that because of the importance of the knowledge and particular style of the designer, it is impossible to accurately propose a solution for all errors without including the designer in the process. This thesis avoids generating plans that may never be applied and completely relies on the criteria of the analyst. Another approach to contribute to the quality of MAS, the validation of MAS models with regards to the client specification, remains largely unexplored.

Fuentes et al. (2004) review four approaches for the verification and validation of MAS. These techniques vary in their degree of formalism, from conventional to formal, none of them involving the use of ontologies. The authors conclude that each of them fail in taking into account all the features of the MAS paradigm. The approach presented in this thesis aims to overcome that research gap. Some attempts to validate MASs have been proposed based on the findings from the Fuentes et al. (2004) study. Okouya et al. (2008) present a model-driven/ontology approach to improve Operetta, a MAS development framework. The authors automatically transform MAS models into an ontology. The semantic constraints of the ontology (and of the MAS models) are verified against a MAS domain ontology. The structure of the models is verified, so it can be assured that they have been correctly built. However, the authors do not validate the models against the client specification. Similarly, Brandão et al. (2007) propose the use of ontologies to verify MAS designs. The authors use an ontology to define the MAS modelling language. These model-diagram mappings enable the automatic validation of the models to check that there are neither intra-model nor inter-model inconsistencies. They can validate the models against their theoretical structure and dynamics, but use no information about the specification or application domain and their proposal has not been properly validated. The proposal of this thesis goes beyond structural verification to focus on the compliance of the models with the client conceptualisation.
2.6 Conclusion

This literature review has highlighted an important research gap with regards to the validation of requirement models taking into account client specifications. In particular, despite their potential as knowledge representation mechanisms, ontologies have been less commonly used to improve software development activities. Rather, they have focused on a runtime role in formulating agent messages. The use of ontologies during development is relatively new, with most of the related approaches that have been identified published from the year 2000 onwards. In most proposals, the ontology plays an important role as a facilitator of understanding and structuring the domain. However, other roles vary depending on the requirements task on which the approach focuses. For example, in works where the ontology is used to facilitate requirements elicitation and analysis, its typical role tends to align with integration purposes, reuse or semantic query. In approaches that focus on validation and verification of requirements, the role of the ontology is to provide consistency and completeness mechanisms. The roles of ontologies vary considerably across the development lifecycle. Whereas at analysis time ontologies were frequently used to assist on consistency and completeness checking and to help understand and structure the domain, in later phases of software development their main roles revolve around annotation, semantic query, reuse and integration. This is consistent with the observations made during this research in the sense that, especially at design and coding time, ontologies are used as an indexing mechanism to find suitable artefacts for reuse.

Ontologies have been less commonly used to improve the development of MAS. Approaches contribute to most development phases equally and mainly use ontologies as an understanding mechanism to provide the structure of MAS, to provide a common vocabulary or to verify the structure of models and types of parameters exchanged by agents. The validation of MAS models, in terms of fixing inconsistencies and incompleteness while taking client specifications into account, has been particularly neglected. The proposal of this thesis aims to contribute to this research gap. Although its details will be thoroughly described in later chapters of this thesis, it is relevant
to outline here how the thesis fits within the existing research by discussing the following two issues:

- How the thesis contributes to assisting in the various requirements engineering phases; and

- Identification of the most prominent roles played by the ontology in relation to the categories described in this chapter.

The proposal presented in this thesis impacts on most of the requirements engineering tasks (as described above), with the degree of impact varying across the tasks. The ontology can assist analysts to elicit hidden requirements. An automatic reasoner makes explicit relationships that had been only implicitly modelled in the ontology. Some of them will be significant for the purpose of the application under development and will be included as requirements in the client specification. Another application of the reasoner is to detect inconsistencies in the domain ontology. Requirements analysis is done by comparing the inconsistency-free ontology with the models. Inconsistencies in the models will be detected as differences with the ontology. In terms of requirements negotiation, the ontology represents the application domain without the redundancy inherent in models. It can bridge the communication gap between clients and analysts and guide their negotiations. The main contribution is in the verification and validation tasks of requirements engineering. Similarly to other proposals, the structural correctness of requirements according to the development paradigm is assured. However, unlike other works, a mechanism is provided to validate the models against the client specification and to improve their completeness. The approach presented in this thesis can also support change management of requirements. It is not uncommon for requirements to undergo changes during the requirements engineering process. The ontology makes it easier to discover the models that have to be changed as a consequence of modifications to other models.

The roles that the ontology plays in the proposal presented in this thesis can be analysed according to the classification described in this chapter. The automatic reasoning mechanism provided by the formal semantics underlying OWL 2 enables the consistency of
the ontology to be checked and implicit knowledge to be extracted from explicitly asserted relationships. As ontology and models are compared, discrepancies between them can uncover inconsistencies and incompleteness in the model. To develop the domain ontology, it is necessary to analyse the domain. This leads to a better understanding of the problem, which will help to produce better quality models. This understanding is enhanced by extending the ontology with paradigm specific knowledge (as part of the V&V process). Although it is not the focus of the research described by this thesis, the structured domain ontology provides the necessary infrastructure to use complex semantic queries using RDF-based languages such as SPARQL (W3C 2008b). The ontology gathers the points of view of different stakeholders in a single artefact while avoiding the redundancies of interdependent models. In this sense, the ontology can be seen as an integration mechanism for the models. The use of the ontology abstracts from the development technology, as comparison operators adapted to varying methodologies are compared to a unique ontology. Finally, the metamodel ontology and the domain ontology by-product of the validation process can be reused, the former in any other project dealing with the same development paradigm, i.e. MAS, and the latter in similar projects on the same domain.

2.7 CHAPTER SUMMARY

This chapter reviewed the background underpinning the research presented in this thesis and existing related works. First, a background to MAS and the state of the art of Agent Oriented Software Engineering (AOSE) were presented. A background on ontologies was then introduced. Existing software engineering applications of ontologies were examined with special emphasis on the requirements engineering phase, which is the focus of this thesis. This analysis was detailed for AOSE and in more detail for the V&V activities of MAS development. The literature review showed that despite the relative youth of the ontology engineering field, there is a significant body of research focusing on uses of ontologies at various stages of software
development. However, the distribution of approaches throughout the software development lifecycle is not even. In particular there are few approaches tackling model validation.

Ensuring error-free analysis models motivates this research, because the cost of fixing errors increases as development proceeds. This is particularly important for MAS because they are very prone to modelling errors due to their complexity. The literature review revealed that, while some authors have proposed mechanisms to verify that MAS models are correctly built, none have tackled the problem of their validation against the user conceptualisation. This thesis will contribute to filling this gap. In the next chapter, the thesis will describe the research methodology that will lead to proposing a mechanism to validate and verify MAS models. The initial user conceptualisation will be represented as a domain ontology and will be used as a reference to detect inconsistencies with the MAS models to detect modelling errors.
In Chapter 2, current best practice in applying ontologies in software development was reviewed. Although interest in the research area of *Ontology-driven Information Systems Development* is growing, there are still important research gaps to be filled, as discussed in Chapter 2. The focus of this thesis is ontology-driven validation and verification (V&V) tasks in the requirements analysis phase for MAS. This chapter describes the design of this research in detail. In Section 3.1, the goals pursued by this research are formalised and the five research phases planned to achieve those goals are summarised. Each of these phases is then detailed in Sections 3.2 to 3.6. Section 3.7 concludes the chapter.

### 3.1 Overview of Design Science Research Method

Hevner et al. (2004) identify two main research approaches in Information Systems (IS). The first is based on *behavioural science* and is concerned with understanding and predicting human and organisational behaviour. The second is based on *design science* and pursues the design of artefacts with novel functionalities to solve human and organisational problems. This thesis uses the *Design Science Approach*.

Through the construction of those artefacts in design science, the knowledge and understanding of the problem domain is also achieved. In the Information Systems (IS) field, the term artefact refers to one of the following four items:

1. ** Constructs**: vocabulary and symbols that provide the language used to describe and communicate IS problems and solutions.

2. ** Models**: abstractions and representations that use constructs to represent an IS problem and its solution. They facilitate understanding of the domain and often connect the problem with the solution.
3. *Methods*: encompass a wide spectrum of artefacts, including formal algorithms, processes or recommended practices, but they all provide a guide to solve a certain problem.

4. *Instantiations*: proof of concepts. They are realisations of any of the other artefacts and aim to prove that they are feasible.

Hevner et al. (2004) define two complementary activities in IS design science research: *building* and *evaluation* of the artefacts. Once the artefacts are built, they should be evaluated with respect to the utility provided in solving the identified problem. Further iterations of the refinement and evaluation are required until the artefacts can adequately solve the problem for which they are designed. The aim of the IS design research presented in this thesis is to design a reliable MAS models V&V process. The synthesis of the novel process requires mitigating significant validity threats to ensure that the synthesis is both domain- and developer-independent. Furthermore, the process is required to be sufficiently general to be applicable in a multitude of AOSE methodologies. This threat of AOSE methodology specificity is an important one to mitigate as many researchers have proposed MAS development methodologies but none of these methodologies have, so far, emerged as an accepted standard. The purpose of this thesis is to present a MAS models V&V add-on process that can be used with any of the extant MAS development methodologies. Therefore, the evaluation activities in this thesis will ensure that the three validity threats are mitigated and the research is designed to test and ensure all three aspects: developer-, domain- and methodology-generality.

Following design science research methodology, the research is organised into five phases (shown in Figure 3.1). *Phase 1* focuses on defining the research problem. In *Phase 2*, the V&V process that addresses the research problem is defined. *Phase 3* evaluates the V&V process. In *Phase 4* an automatic mechanism to apply the V&V process is developed. *Phase 5* evaluates the automated V&V process resulting from Phase 4. According to the principles of design science research, following evaluations, prior phases are revisited to refine the V&V process and its automation as required.
3.2 PHASE 1: PROBLEM IDENTIFICATION

The design science research methodology commences with a study of the most important works on topics related to the project. The purpose of the study is threefold: first, to identify the specific research problem; second, to contextualise the research and to gain a broad view of existing approaches proposed by experts in the field; and third, to rationalise the importance of the solution. Phase 1 of this thesis is composed of three corresponding activities (see Figure 3.2), which involve conducting the literature review to inform the formalisation of the research problem. The goal and significance of the research was presented in Chapter 1. A literature review was presented and discussed in Chapter 2.
The literature review was initiated by examining the foundational concepts of the thesis, namely ontology and MAS. In particular, a set of models commonly used across several notable MAS development methodologies was identified. These models were sufficiently representative for a standard MAS and were also chosen to be used as initial input for a set of validation operators in Phase 2 of the research. Works using ontologies in different activities of the software development lifecycle were also studied. Although the main focus of the analysis was the requirements engineering activities, and in particular the validation and verification of requirements, other development activities were also reviewed to better frame the thesis. Finally, works considering the uses of ontologies for MAS development were reviewed. The literature review confirmed that there is an important research gap in the use of ontologies to validate requirement models taking into account client conceptualisation, especially in MAS developments.

The thesis will contribute to Agent Oriented Software Engineering with a generic semi-automatic V&V ontology-based process that can be used as an add-on to support requirement analysis in MAS development methodologies. The cornerstone of the V&V process will
be a domain ontology providing a conceptualisation of the problem that the system will solve. It may come through interviews with the user or the analysis of related domain documents. It is assumed that an appropriate ontology is available, or alternatively that an ontology can be sourced (adapted or developed from scratch). The V&V process will focus on the improvement of the requirements workproducts and their analysis. The validation of the models will be based on a comparison with the ontology to detect discrepancies. Having the models and the ontology developed by the same developer, based on only one interpretation of the problem, could result in a weak validation, as the same errors could easily recur in both artefacts. Therefore, to avoid the subjectivity of the developers involved, the process will require that the ontology and software models are developed independently.

The proposed V&V process will help elicit overlooked requirements in certain cases (as will be discussed in Chapter 4). However, the main target of this research is not to develop a requirement elicitation mechanism per se, as this has been widely discussed in the literature (Fuentes-Fernández et al. 2009; Goguen and Linde 1993; Yu 1997). This research targets the validation of the requirement analysis models against a domain ontology that represents the client’s conceptualisation. The process created in this thesis aims to support developers during the model development by detecting potential errors and inconsistencies. It will not replace developers’ decision-making during the analysis. It is their role to decide whether the inconsistencies highlighted by the automated process are due to errors in the requirement models or in the ontology itself. If developers judge the errors to be due to the requirement models, they will also need to decide, perhaps in consultation with the client, if the requirements are to be extended or the inconsistency problem is inconsequential and can be safely ignored. The output of the process is a set of validated analysis models. The transition from analysis to design will then be based on decisions about the architecture of the software system. This transition is outside the scope of this thesis.
3.3 Phase 2: Initial V&V Process Definition

Following the design science paradigm, this phase produces an artefact to solve the problem identified in Phase 1. Using the artefacts taxonomy given in Hevner et al. (2004), the artefact output of this activity is a method. It provides directions to solve a problem. Figure 3.3 shows a high level description of the solution to be produced at the end of the research.

![Figure 3.3: High level V&V process description](image)

The literature review showed that a hindrance in adopting MAS is that their complexity can often be a source of errors. As MAS development proceeds, errors become costlier to fix. The purpose of the V&V process sketched in Figure 3.3 is to validate and verify the models as soon as they become available. The goal is for most analysis errors to be fixed before the models move on to design phases. The literature review also found that some approaches try to verify the structure of MAS systems (Brandão et al. 2007; Okouya et al. 2008). However, none of the approaches takes into account whether the domain knowledge used to validate that the system under development matches the client expectations. For the V&V process developed for this thesis, it has been chosen to model the domain problem as conceptualised by the client using ontologies. Ontologies can then be used to validate the MAS models and also be exploited in terms of automatic reasoning, verification and inference of further validation knowledge.
As no system can replace human criteria for decision making, the V&V process is conceived as a decision support mechanism. The V&V process produces a report that highlights differences between models and ontology. The client and developer jointly discuss the report, and the models and ontology are modified accordingly. This comparison and modification cycle iterates until the MAS models are considered correct and valid.

In developing the process, a central aim is to ensure that it has sufficient semantic richness to cover a broad spectrum of agent models. Based on the literature surveyed in Phase 1, a set of commonly used agent models is selected. In other words, the first version of the V&V process is designed based on the research question formulated as a result of Phase 1 and the literature review. A set of operators is created to support their validation. It is expected that this set of operators will enable validation of most of the models of most AOSE methodologies. It is also expected that the performance of the validation process can be improved by adding methodology specific operators. This first version is conceived as a manual process for a support tool that will be developed in Phase 4 of the research. Indeed, the development of the support tool is paramount for the efficient application of the V&V process in real world projects. It is well known that the processing of ontologies is a highly complex and resource-consuming task (Cuenca-Grau et al. 2007). It is not expected that the manual process on its own could provide an efficient mechanism to V&V MAS models for large projects. This phase will be revisited to refine the V&V process as result of the validation conducted in Phase 3.

3.4 PHASE 3: V&V PROCESS EVALUATION

According to the framework for design science, it is critical to evaluate whether the designed artefact can effectively solve the problem that motivated its creation (Hevner et al. 2004). In this thesis, Phase 3 uses a development case study to validate the V&V process output of Phase 2. The case study deploys the V&V process during the development of a MAS employing a specific AOSE methodology.
The methodology is iterative. Any inadequacies observed in the V&V process are resolved during iterations of the development. The validation in this phase focuses on the structure of the process to ensure it captures the increments of the development. Refinement of the process paves the way for investing the effort in automating it. The validation of Phase 3 does not deal with the validity threats alluded to earlier in this chapter. Those will be targeted in Phase 5, once the process is partially automated.

Based on the literature review (Beydoun et al. 2009, 2006a; Tran et al. 2006; Tran and Low 2005) the ROADMAP (Juan et al. 2002) methodology is chosen for this initial evaluation phase as ROADMAP models are not atypical, rather they include common agent analysis models. Deploying the process in ROADMAP required minimal effort. It was a simple process to instantiate the validation operators for ROADMAP, so it was possible to detail how elements of its models can benefit from the ontology to detect ill-modelled aspects. This also constituted a preliminary validation of the set of operators. ROADMAP is also iterative, facilitating multiple refinement opportunities of the V&V process. The chosen MAS simulation application in this case study is also a real world application pursued in collaboration with an industry partner, to create a simulator for estimating the airport delays during aircraft turnaround activities. Validation activities and modelling activities are interleaved to facilitate the iterative development. In this phase of the research, the process is manually applied. The extent of the benefits to modelling and the effectiveness of V&V activities are assessed. Scope for improving the V&V process is identified. Some theoretical issues are uncovered, such as allowing the evolution of the ontology along the validation process, and these affect the very design of the validation process. Practical issues derived from the manual application of the V&V process are confirmed. These include the scalability of ontology augmentation, the application of validation operators and the management of interdependency of the models. These issues will be addressed by the automation and the subsequent implementation of the supporting tool.
3.5 PHASE 4: AUTOMATING THE V&V PROCESS

In Phase 4, the benefits of the formal semantics of ontologies are explored to automate components of the V&V process. Harnessing the reasoning capabilities of ontologies is a key motive of their use in this thesis and was contemplated in the original definition of the V&V process. Specifically, two key aspects of the V&V process are automated in this phase: the augmentation of the domain ontology with MAS knowledge; and the application of the validation operators. Concepts and relations used in MAS modelling as identified in the literature review are used to augment the domain ontology automatically to enable its use. A report that collects the discrepancies between the MAS models and the ontology is also automatically generated. With this, a supporting tool is created. In terms of the taxonomy defined by Hevner et al. (2004), this tool instantiates the process and proves its feasibility.

The V&V process evaluation conducted in Phase 3 identifies the scope for automating key activities. The augmentation of the ontology is one such activity that depends on the size of the domain. It requires every ontology concept related to a MAS analysis paradigm. This task can quickly become intractable and it is critical to offer automatic support to undertake it. The automatic V&V process is implemented as a decision support tool that automatically augments the domain ontology. The tool then automatically applies the operators designed in phases 2 and 3 to compare the ontology with the MAS models. The tool generates a report automatically to highlight any discrepancies.

3.6 PHASE 5: EVALUATION OF THE AUTOMATIC V&V PROCESS

Phase 5 evaluates the V&V process and its automatic instantiation with further case studies that focus on mitigating the validity threats that were identified earlier. Different developers produce analysis models corresponding to different MAS development methodologies for various domain problems. The V&V process is used during the analysis with the support of the automatic tool.
The evaluation is designed to span several dimensions addressing internal and external validity threats. Internal validity refers to whether an experiment makes a difference, whereas external validity refers to whether the result of the experiment is generalisable (Campbell and Stanley 1963). The dimensions considered in the evaluation are the following:

- **Formal dimension.** The literature review (Beydoun et al. 2009, 2006a; Tran et al. 2006; Tran and Low 2005) is used to identify a set of sufficiently representative MAS models and MAS development methodologies that prescribe their use. This dimension supports external validity, as the identification of commonly used models facilitates the easy adaptation of the V&V process to a wide range of MAS development methodologies.

- **Tool dimension.** A manual application of the process is compared with a tool-assisted one. This dimension affects the internal validity, as the outcome of the validation process varies greatly in terms of time depending on whether a manual or automatic approach has been followed.

- **Domain dimension.** The V&V process is applied to validate MAS development for different domains. This dimension supports external validity as the concern of having developed a process only suitable for a narrow type of application in one domain is mitigated. This dimension also helps by mitigating the internal validity threat that the developer could avoid modelling errors based on previous experience developing applications in the same domain.

- **User dimension.** The V&V process is applied in case studies undertaken by different developers. This dimension contributes to the assurance of external validity, as it has been illustrated that the developers’ skills or temperament do not impact its outcome.

- **Methodology dimension.** The V&V process is applied, supporting different MAS development methodologies. This dimension
illustrates the adaptability of the process to different development methodologies and therefore mitigates an external validity threat. Similarly to the domain dimension, developing models following a different methodology prevents the developer from avoiding errors based on previous experiences.

The research activities in this phase are organised as follows:

- **Activity 5.1: V&V Process Domain Generality Evaluation.** A development case study based on a domain of analysis different to the one used during Activity 3.2 (a wine broker application) is set up and conducted. The application domain is chosen to be very different from the aircraft turnaround problem to test the flexibility of the validation process in different domains. The development methodology is maintained to be ROADMAP. However, the MAS models are not validated manually, but rather use the support tool. Additionally, a new developer is engaged in this case study to assess whether the process depends on the developers involved.

- **Activity 5.2: V&V Process Methodology Independence.** The V&V process is used with a second MAS development methodology, MOBMAS (Tran and Low 2008). This first requires adapting some of the ontology-based operators. Again, the automatic support tool is used (unlike Phase 3). The aircraft turnaround is revisited to develop a new set of models using MOBMAS. The focus of this validation is to test the flexibility of the V&V process and support tool in terms of MAS development methodology.

### 3.7 Chapter Summary

This chapter reviewed the research methodology used in this thesis. A design science approach is followed to create and automate an ontology-guided process for V&V of MAS models taking into account client conceptualisation. The research is organised in five iterative phases. The development of the process and a semi-automatic support tool will be interleaved with successive validations using
various case studies. This is summarized in Figure 3.4, which shows the research flow and details activities conducted within each phase.

Figure 3.4: Iterative design and evaluation research process summary

Related work preliminary to the synthesis of the validation process was compiled in Chapter 2 of this thesis. This was used to define the research problem. Chapter 4 will present phases 2 and 3 of the research and will produce the first version of the V&V process. This first version will be used and evaluated with the ROADMAP methodology. Chapter 5 will present Phase 4 of the research. It will further evolve the V&V process according to the output evaluation undertaken in Phase 3. The most notable aspect of this evolution is the co-evolution of ontology and MAS models. The support tool is developed to automate part of the V&V process. Finally, Chapter 6 will present Activity 5.1 and 5.2 of Phase 5. The V&V process is
used with ROADMAP and MOBMAS to demonstrate the methodology independence of the process, its automaticity, its developer independence and its domain independence. Chapter 7 of the thesis presents the conclusions of the research and outlines future work.
AN ONTOLOGY-BASED MAS MODELS V&V PROCESS

The literature review presented in Chapter 2 confirmed that there has been a lack of research on mechanisms to support the validation and verification (V&V) of Multi-Agent System (MAS) analysis models. Ontologies were identified as suitable artefacts to describe a problem domain (Henderson-Sellers 2011). Nevertheless, approaches that harness the capabilities of ontologies to validate MAS analysis models are scarce. The literature survey also showed that there is a set of concepts and relationships common to most of the existing MAS development methodologies (Beydoun et al. 2006a, 2009; Tran and Low 2005). Based on the conclusions drawn from the literature review, this chapter presents the first version of the ontology-based MAS models V&V process. The V&V process and ontology-based operators are then validated in the context of a real-world case study involving the development of an aircraft turnaround simulator.

This chapter is structured as follows: In Section 4.1 the ontology-based MAS models V&V process is presented. Section 4.2 describes the ontology-based comparison operators that enable the validation of MAS models. In Section 4.3, the case study utilised to evaluate the V&V process is presented. Section 4.4 discusses the outcomes of the V&V process evaluation. The chapter concludes with Section 4.5.

4.1 DEFINITION OF THE ONTOLOGY-BASED V&V PROCESS

The literature review presented in Chapter 2 revealed the existence of many competing agent oriented software engineering (AOSE) methodologies. While a wide range of options can be beneficial in the early stages of development of a new paradigm, such as AOSE, an agreed approach is necessary to reach maturity (Beydoun et al. 2009). Defining MAS model V&V add-on processes that are only applicable to certain methodologies would only prolong the lack of consensus
in the AOSE field. This thesis aims to define an ontology-based MAS model V&V add-on process that can be applied to any of the extant AOSE methodologies. To inform the design of such a process, notable AOSE methodologies were analysed during the literature survey discussed in Chapter 2. A common feature across all studied methodologies is the iterative character of their development process. As Chapter 2 discussed, this process model is highly suitable due to the inherent complexity of MASs and their development. The use of iterative modelling is therefore convenient, allowing individual models to be reviewed before adding new aspects to them or creating new ones. As such, the V&V add-on process developed in this thesis is designed to support iterative and incremental development processes. This is depicted in Figure 4.1. The figure represents the analysis models required for a given MAS development (in the horizontal axis) versus the iterations required to achieve a complete set of analysis models (in the vertical axis). The figure shows that only a subset of incomplete models is necessary to commence the V&V process. However, as different AOSE methodologies may define different iteration times or leave this to the discretion of the developer, the V&V add-on process does not establish iterations at fixed times. Rather, it intertwines with the development process and is applied to the models as they spiral toward acceptance and completion (as illustrated in Figure 4.1). Thus the V&V add-on process ensures that models are validated as they are used during the development process, and thus prevents compounding modelling errors. This lowers MAS development costs, given that it is well known that the cost associated with errors dramatically increases as the software development process proceeds (Westland 2002). In effect, the V&V overhead cost is a fully justified investment.

The ontology-based MAS V&V add-on process is proposed as an iterative sequence that consists of the following five tasks to complement the MAS model development process: Ontology Acquisition, Ontology Augmentation, Ontology Validation, MAS Software Models Validation and MAS Software Models Improvement. This sequence is depicted in Figure 4.2. Although the MAS model development task is not in essence part of the V&V process, it has been included in Figure 4.2 to show how both aspects are intertwined. As soon as the
models are sufficiently mature, the V&V process can be initiated and conducted in parallel with the development process as shown earlier in Figure 4.1. The remainder of this section details the process and the role each of the five tasks.

4.1.1 Task 1: Ontology Acquisition

In Task 1, Ontology Acquisition, a suitable ontology is retrieved from an existing repository; if this is not possible, one is built using the most suitable ontology engineering techniques. This task does not enforce the use of any particular ontology engineering methodologies. Examples of methodologies that can be applied to produce the domain ontology were discussed in Chapter 2. The sub-tasks typically described by ontology development methodologies are (Noy and McGuinness 2001):

- Determining the domain, scope and intended uses of the ontology.
- Studying whether an existing ontology can be reused.
- Enumerating important terms in the ontology.
• Defining classes and class hierarchy (subsumption relationships).

• Defining the properties of classes (or the type of relationships that can be established amongst the classes).

• Assigning important characteristics to properties (such as domain, range or cardinality).

• Populating the ontology by adding instances to classes.

Occasionally, an ontology modelling the domain of interest of the AOSE project has been previously developed elsewhere and is already available. In such a case, reusing the whole or part of the domain ontology can reduce the V&V process application time.
example, the search engine Swoogle (2007) retrieves ontologies from public repositories that match input search keywords.

This task initiates the rest of the V&V process. Input to this task comes through elicitation techniques such as interviewing clients and acquiring any documents that can describe their business processes. For example, in the case study described in Section 4.3, in addition to the interviews, diagrams provided by the client to describe the existing timeline for an aircraft turnaround process are used. The output of the task, i.e. the domain ontology, acts as a benchmark to detect modelling errors in the MAS models. Communication with the client has to be initially intensive to ensure that the modelled domain is as detailed as required and matches the conceptualisation of the client. If the ontology lacks detail then its effectiveness in validation and in providing modelling assistance to software developers is reduced.

4.1.2 Task 2: Ontology Augmentation

In Task 2, Ontology Augmentation, the ontology produced in Task 1 is augmented to represent features related to the AOSE paradigm. Domain concepts are linked to paradigm concepts and relations between domain concepts are created according to existing relations defined for the paradigm. The literature review presented in Chapter 2 identified concepts that typically characterise the AOSE paradigm at analysis level. These invariably include: Goal, a functional requirement of the system; Role, any capacity that the system requires in order to achieve its goal; Activity, any work carried out by a role in order to fully or partially fulfil its goal; Environment Entity, any entity which is not part of the system but it is needed by the roles to achieve their goals; and Agent, a proactive or reactive component of the system that plays one or more roles. For the purpose of this thesis, no distinction is made between reactive or deliberative agents. The internal design of agents is typically part of the detailed design phase of AOSE methodologies, hence out of the scope of this thesis (see Section 1.2).
Table 4.1: AOSE relations for ontology augmentation

<table>
<thead>
<tr>
<th>Domain</th>
<th>Relation</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>has subgoal</td>
<td>Goal</td>
</tr>
<tr>
<td>Role</td>
<td>responsible for</td>
<td>Goal</td>
</tr>
<tr>
<td>Role</td>
<td>participates in</td>
<td>Activity</td>
</tr>
<tr>
<td>Role</td>
<td>is peer</td>
<td>Role</td>
</tr>
<tr>
<td>Role</td>
<td>controls</td>
<td>Role</td>
</tr>
<tr>
<td>Role</td>
<td>is controlled by</td>
<td>Role</td>
</tr>
<tr>
<td>Role</td>
<td>uses</td>
<td>Environment Entity</td>
</tr>
<tr>
<td>Agent</td>
<td>plays</td>
<td>Role</td>
</tr>
<tr>
<td>Activity</td>
<td>fulfils</td>
<td>Goal</td>
</tr>
<tr>
<td>Activity</td>
<td>needs</td>
<td>Environment Entity</td>
</tr>
<tr>
<td>Activity</td>
<td>precedes</td>
<td>Activity</td>
</tr>
<tr>
<td>Activity</td>
<td>follows</td>
<td>Activity</td>
</tr>
</tbody>
</table>

Typical relations between those concepts have also been identified in the literature review, as summarised in Table 4.1. They are the following: the relation has subgoal establishes how goals can be decomposed in subgoals. The relations responsible for, participates in and uses link roles with the goals that they pursue, the activities that require their participation and the environment entities that they use, respectively. The relations is peer, controls and is controlled by model the authority relations amongst roles. The set of roles that is assigned to agent is modelled through the relation plays. The relations fulfils and needs describe the goals that are fulfilled by activities and the environment entities that are required during the execution of activities, respectively. As agents are time-aware, every decision agents make and every action they carry on have to fit in a certain sequence. To specify this sequence, the relations precedes and follows establish order between various agent activities.

An augmented ontology will always be larger than the input domain ontology, as links between domain concepts and AOSE concepts and other relationships are added. For example, assume that a domain ontology that models certain activities conducted as part of the daily routine of an airport includes the following knowledge:

- Airline cleaning staff is peer Airline catering staff
• Airline catering staff uses Galley truck
• Airline cleaning staff responsible for Aircraft service
• Airline catering staff responsible for Aircraft service
• Airline cleaning staff participates in Aircraft cleaning
• Airline catering staff participates in Aircraft catering
• Aircraft cleaning precedes Aircraft catering
• Aircraft cleaning fulfils Aircraft service
• Aircraft catering fulfils Aircraft service
• Aircraft turnaround has subgoal Aircraft service

Then the augmentation process would add the following knowledge:

• Airline cleaning staff is a Role
• Airline cleaning staff is an Agent
• Airline cleaning (Agent) plays Airline cleaning (Role)
• Airline catering staff is a Role
• Airline catering staff is an Agent
• Airline catering staff (Agent) plays Airline catering staff (Role)
• Galley truck is an Environment Entity
• Aircraft cleaning is an Activity
• Aircraft catering is an Activity
• Aircraft service is a Goal
• Aircraft turnaround is a Goal
• Aircraft catering needs Galley truck
• Airline catering Staff is peer Airline cleaning staff
• Aircraft catering follows Aircraft cleaning

• Aircraft cleaning fulfils Aircraft turnaround

• Aircraft catering fulfils Aircraft turnaround

In the above example, relations are added to the domain ontology to classify the domain concepts from an AOSE perspective. Additionally, relations that can also be derived from explicitly stated relations in the domain ontology can also be added to the augmented domain ontology. For example, the relation Aircraft catering needs Galley truck is derived from the fact that the Airline catering staff is the only participating role in the activity Aircraft catering and this role uses the Galley truck. Hence, a Galley truck is an environment entity needed by the activity Airline catering. Generally, the decision of what relations should be added to the domain ontology depends on the ontology modeller and their understanding of the problem from an AOSE perspective. For example, in the example, it has been decided that each role will be played by a separate agent. However, an alternative decision could have been made to define an agent Aircraft service staff and to include the relations Aircraft service staff plays Airline catering staff and Aircraft service staff plays Airline cleaning staff. In a later refinement of the V&V process presented in Chapter 5, a systematic mechanism to derive implicit relations from explicitly stated knowledge will be defined. The augmentation of the domain ontology entails the manual classification of each domain entity. This may pose a challenge for large ontologies. The first case study (see sections 4.3 and 4.4) will illustrate this issue. The automatic V&V process presented in Chapter 5 will resolve this limitation.

4.1.3 Task 3: Ontology Validation

In Task 3, Ontology Validation, the augmented domain ontology produced in Task 2 is validated. The goal of this task is to ensure that the ontology is compliant and accommodating of the conceptualisation of the client. This task is important because the ontology will be used as a canonical artefact to which each model will be compared.
for discrepancies, i.e. modelling errors. Two types of errors are distinguished for the purpose of this task: Errors of the first type violate the axioms described in the ontology and can be detected by the automatic reasoning mechanisms supported by ontologies; Errors of the second type are rooted in an erroneous conceptualisation and cannot be automatically detected by reasoners.

Before the ontology is discussed with the client to assess its compliance with the problem conceptualisation, a filter will be applied to the ontology to remove all instances of the first type of error. This filter is the consistency checking mechanisms of automatic reasoners that harness the formal semantics that underpin ontologies (as discussed in Chapter 2). For example, if the ontology includes the relationships \textit{Pilot controls Crew}, \textit{Crew controls Passenger} and \textit{Passenger controls Pilot}, the reasoner will raise an inconsistency alarm, as a cycle in the authority hierarchy is detected. Further details on inconsistencies and associated detection mechanisms will be discussed in Chapter 5.

In case the ontology being validated (or part of it) is a taxonomy, existing approaches can facilitate improving its quality. In particular, the author of this thesis has been involved in research to produce a semi-automatic methodology to build, evaluate and improve the quality of taxonomies. The approach is underpinned by a mathematical definition of quality metrics that measure, amongst other things, similarity between classes based on the values of their attributes. Top-down and bottom-up taxonomy building mechanisms are defined to produce the closest upper (generalisation) or lower (specialisation) class, respectively. The methodology can also be applied to assess whether the subsumption relationships in an existing taxonomy are correct or violate the axioms defined with regards to concepts distance. Further detail on these works can be found in Beydoun et al. (2011a) and Beydoun et al. (2012).

Once the consistency of the ontology is guaranteed, the client examines it to validate that it complies with their conceptualisation. For example, if the ontology models that the role \textit{Engineer} participates in the activities \textit{Aircraft routine maintenance} and \textit{Aircraft non-routine maintenance}, the client could argue that the engineer is only needed in the non-routine maintenance activity because a technician is qual-
ified to conduct the aircraft routine maintenance. Any issues raised by the client during this process are used to improve the ontology before proceeding to next task, during which the ontology is used to validate the MAS models themselves.

4.1.4 Task 4: MAS Software Models Validation

In Task 4, **MAS Software Models Validation**, the MAS models are validated against the augmented ontology for consistency and compliance with the client’s specification. This task has two inputs, the MAS models and the augmented domain ontology as validated in Task 3. Elements of each MAS model are compared against corresponding elements in the ontology. Results of this comparison determine the need for further modelling and validation iterations. A new iteration is necessary if any discrepancy is found between the ontology and models, and hence, if any recommendation is made to improve the quality of the models. Not all the models can be validated to the same extent using the ontology. Some may be very structured and the use of the ontology provides specific instructions to improve them. Other models may be composed of free text, for which the use of the ontology is only able to provide a guideline for the analyst to interpret. Section 4.2 describes the comparison operators defined for a standardised set of MAS models. These operators will discover discrepancies between the models and the ontology. For example, during the validation of the environment model, the comparison operators reveal that in the ontology, it has been defined that the role *Airport ground staff* uses the environment entities *Bulk cargo loader* and *Bulk cargo train* whereas the model states that the role *Airport ground staff* only uses the environment entity *Bulk cargo loader*. A recommendation is generated to add the relation *Airport ground staff uses Bulk cargo train* to the environment model. At the end of this task a report compiling the recommendations to improve the MAS models is produced.
4.1.5 **Task 5: MAS Software Models Improvement**

In Task 5, *MAS Software Models Improvement*, a recommendation report rooted in the application of the operators during Task 4 is analysed by the developers to decide what changes the models should undergo and which recommendations can be safely ignored. Recommendations are generated because discrepancies between the ontology and the MAS models are detected. All discrepancies have to be solved to finalise the V&V process. Discrepancies may occur because of issues regarding consistency or completeness:

1. **Discrepancies due to inconsistency.** These discrepancies detect different interpretations of the domain held by the ontology engineers and modellers. They may occur at any time and tackling them as soon as possible will prevent compound errors. To solve them, either the ontology or the model must be modified. As the ontology has been validated with the client, it should be compliant with the problem conceptualisation and hence be considered correct. However, it may happen that the client conceptualisation changes during the system development. In this case, the ontology may have to be modified to comply with the new requirements. *Figure 4.2* reflects the possibility of having to revisit the domain ontology development by the dashed arrow that returns the workflow from Task 5 to Task 1. It is expected that the preliminary V&V process evaluation (Section 4.3) will confirm the need for revisiting Task 1.

2. **Discrepancies due to incompleteness of the models.** These discrepancies detect missing details from the models and are more frequent in the early stages of validation, when the models are immature. As the models develop, most of these discrepancies will naturally disappear. However, if these discrepancies are detected when the models are mature, they are genuine errors by the developers and should be treated in consequence. They will occur when the application of a comparison operator detects that something is modelled only in the ontology.
3. **Discrepancies due to incompleteness of the ontology.** These discrepancies may occur in later phases of the development. As discussed for discrepancy type 1 (discrepancies due to inconsistency), if the client conceptualisation changes, the domain ontology design may have to be revisited. If this happens, the models reflect the real problem better than the ontology. The evolution of the initial requirements will make the ontology evolve.

4. **Discrepancies for requirements elicitation.** Similar to discrepancy type 2 (discrepancies due to incompleteness of the models), this type also contributes to the completion of the models. However, these discrepancies are not caused by developer errors; they occur because the ontology contains details that were not considered by the client at specification time. They should be presented to the client for approval. Depending on the purpose of the final application, they may or may not be relevant.

   For example, the recommendation illustrated above relating to the *Bulk cargo train* would be an instance of discrepancy type 2, as it aims to complete the environment model. The developer could decide (alone or in consultation with the client) that the ontology is correct and that the model must be modified to incorporate the relation *Airport ground staff uses Bulk cargo train*. Alternatively, consulting the client perhaps, the developer can decide that adding that relationship to the model adds details that are unnecessary for the purpose of the application being developed and should be ignored.

   The development case study discussed in Section 4.3 will present examples of all four types of discrepancies and will confirm the need for revisiting Task 1 to incorporate new requirements into the original problem conceptualisation.

   After improving the quality of the MAS models according to chosen recommendations, the new set of models will be used as input for Task 4 in the next iteration. Requirements analysis is repeated as part of each iteration along the sequence of workproducts required by the chosen development methodology. Problems in reviewed models are fixed before they are fully developed. Any models yet to be commenced in that iteration will take advantage of the recommendations, avoiding compounded errors. For example, it could
happen that during the first iteration, the V&V process is undertaken on the goal and role models. The operators detect that the role Technician is defined in the ontology but not in the goal or role models. The developer reviews the recommendation report and agrees with the addition of the role to both models. Then, the second iteration commences with the development of the organisation model, which also includes the roles to model another aspect of the system. As the modelling error involving the Technician has already been detected and addressed, the developer takes into account that role for the organisation model, thus avoiding later rework.

4.2 MAS MODELS ONTOLOGY-BASED VALIDATION OPERATORS

In order to perform the validation described by Task 4 in Section 4.1.4, a set of MAS models has to be selected and the corresponding operators have to be defined. The literature review presented in Chapter 2 showed that there is a set of common concepts and relationships across existing MAS methodologies. In fact, the literature review exposed that the most common elements amongst notable AOSE methodologies are the following: All considered AOSE methodologies define system activities, agents and the interactions between them; 80% of the methodologies identify the environment entities that interact with the agents; 70% of them identify the roles that agents play; 60% of them define authority relationships amongst agents; and half of them map agents with the goals that they pursue. This set of common concepts and relationships is also included in the generic metamodel for MAS development that Beydoun et al. (2009) synthetised after analysing over 20 AOSE methodologies. However, there is no similar consensus with regards to the definition of the MASs that use these concepts and relationships. For example, system functionalities are described by the use case model in Adelfe, by actor and rationale diagram in TROPOS, by role model in Gaia, by functionality descriptor in Prometheus or using the system requirements model in PASSI (Tran and Low 2008). The literature review showed a set of MAS models commonly defined by AOSE methodologies that completely cover the above discussed concepts.
and relationships. In decreasing order of popularity are the following: Agent model (90%), goal model (60%), interaction model (60%), scenarios (50%), organisation model (40%), role model (30%), and environment model (30%). The set of models described in this section is chosen to synthesise all these concepts and relationships without loss of generality. In what follows, each of the models is described, along with the operators to validate it against the ontology. To facilitate the description of the models and operators, for each MAS model an illustrative example is provided. The examples are extracted from the aircraft turnaround domain, which is the problem domain of the case study described in detail in Section 4.3. The lack of consensus within the AOSE community with regards to MAS models also extends to the notation used to describe them. Without loss of generality, the notation chosen for the examples corresponds to the ROADMAP methodology (Sterling and Taveter 2009), as it adequately conforms with the description of the MAS models given in the following sections.

4.2.1 Goal Model and its Ontology-based Validation Operators

A Goal Model describes a hierarchy of goals that describe the purpose of the system and that have as a root the main objective of the system to be developed (Bresciani et al. 2004). They are intended to be the vehicle used by clients and developers to discuss the goals of the system and the roles required to fulfil them. The elements that compose them (goals and roles) and their notation are intended to be intuitive and simple (e.g. Figure 4.3) because they have to be understood by the client, who may have no background on software engineering. Goals are representations of functional requirements of the system. Roles are the capacities that the system requires in order to achieve its goals. Goals can be decomposed into smaller sub goals to represent separate aspects of the larger goal. Roles are extracted by analysing goals to decide the functionalities that are required to meet them. More than one role may be required by each goal and different goals may need the same roles. Goal models are used in later phases of system modelling to define interactions
between roles. In the example of Figure 4.3, the general purpose of
the system is modelled with the goal Aircraft turnaround, which in
turns is divided into smaller sub goals that cover different aspects
such as Prepare arrival, Disembark or Prepare departure. The sub goals
Prepare arrival and Prepare departure are further decomposed into
other sub goals. Goals are associated with the roles that encompass
the functionalities needed to fulfil them. Roles Crew and Passenger
are associated with the goal Disembark, as both of them are necessary
to meet the goal. In later phases of system modelling, when roles are
materialised as agents, this will imply that the agents encompassing
the functionality of the roles Crew and Passenger collaborate to meet
the goal Disembark. The role Ground staff is associated with the goal
Prepare departure, meaning that it is the only role required by all the
sub goals of Prepare departure.

![Figure 4.3: A goal model example using the ROADMAP notation](image)

The ontology can ensure that all the specified goals are accounted
for, and that the integrity and hierarchy of the roles is maintained.
The goal model validation consists of the following proposals:

1. To add to the model any goals that are defined in the ontology
   but not used in the goal model, and to remove those not defined
   in the ontology.
2. To add to the model any hierarchical relation *goal is sub goal of goal* that is defined in the ontology but not used in the model, and to remove those not defined in the ontology.

3. To add to the model any relation *role responsible for goal* that is defined in the ontology but not used, and to remove those not defined in the ontology.

### 4.2.2 Role Model and its Ontology-based Validation Operators

Roles were initially defined and associated with goals in the context of goal models. The *Role Model* extends their description. In general, roles define what is expected to be done to fulfil goals, both by themselves and in concert with other roles. A role model consists of five elements (Sterling and Taveter 2009):

- **Role ID**: A unique code to refer to the particular role in later models.
- **Role Name**: A name that identifies the role.
- **Role Description**: A textual description of the role.
- **Role Responsibilities**: The list of responsibilities that the role must perform in order to fulfil its associated goals.
- **Role Constraints**: A list of conditions that have to be taken into account during the execution of the functionalities encompassed by the role.

In the example depicted in Figure 4.4, the role *Ground staff* is detailed as having the responsibilities of processing the baggage handling and preparing arrival and departure of an aircraft in the context of the operations associated with the aircraft turnaround. In this example, the role has no constraints that limit the exercise of its responsibilities. Several fields of this model are free text strings; this restricts the degree to which the ontology can help validating it. The ontology can check that a role model has been defined for all the roles, and give the analyst some ’hints’ about its associated responsibilities. The validation consists of the following proposal:
1. To add to the model set any role that is defined in the ontology but not in corresponding models, and to remove any role model without a corresponding role defined in the ontology.

Regarding the responsibilities, a report compiling all the roles defined in the ontology along with their associated goals is generated. If a role pursues a goal, this means that it has some responsibility for it. It is possible to use this report as a checklist to ensure that every goal has been covered by at least one responsibility. Two or more responsibilities could cover just one goal, but no goal should remain uncovered.

### 4.2.3 Organisation Model and its Ontology-based Validation Operators

The *Organisation Model* reflects the relationships between the roles defined in the role model and can be useful to define the interactions between agents later in the development process (Taveter and Wagner 2005). The organisation model is composed of an organisation diagram for each role $R$ defined in the role model. Each diagram describes the authority relations of the role $R$ with the rest of roles. The relation *is peer* establishes that none of the roles participating in the relationship exerts control over the other. If, in the organisation diagram of role $A$, it is asserted the relationship $A$ *is peer* $B$, then in the organisation diagram of role $B$ the symmetric relationship, $B$ *is peer* $A$, should be included. The relations *controls* and *is controlled by* are the inverse of each other and describe that one role exerts control over the other one. If, in the organisation diagram of role $C$,
it is asserted the relationship \( C \) controls \( D \), then in the organisation
diagram of role \( D \) there will be a relationship \( D \) is controlled by \( C \). In
the example of Figure 4.5, the role Ground staff is peer with the Pilot
and the Crew and is controlled by the Manager. In the organisation
diagrams of the roles Pilot and Crew it should be stated that they are
peer with the Ground staff and in the organisation diagram of the
role Manager, it must control the Ground staff. The ontology can help
by checking that relations between roles are correct and that models
are internally consistent. The validation consists of the following
proposals:

1. To add an organisation model for any role defined in the ontol-
ogy that does not have a corresponding organisation model,
and to remove those not having a corresponding role defined
in the ontology.

2. To add to the organisation model of Role1 any relation that is
defined in the ontology of the type Role1 is peer with / controls /
is controlled by Role2 but that is not included in any organisation
model, and to remove those not defined in the ontology.

4.2.4 Environment Model and its Ontology-based Validation Operators

A MAS is always situated in some environment (Zambonelli et al.
2005). This environment could be physical, virtual or often a com-
bination of both. Physical environments include elements such as

Figure 4.5: An organisation model example using the ROADMAP notation
sensors and actuators so the MAS can receive information from other devices and manipulate them. Virtual environments encompass other software entities, such as information systems, web services or database management systems. In any case, the environment has to be modelled to detail the resources needed by the MAS to fulfil its goals. The Environment Model describes each environment entity used by the MAS. For each environment entity the following fields are detailed:

- **Environment entity name**: A name that identifies the environment entity.
- **Environment entity ID**: A unique code to refer to the particular environment entity in later models.
- **Environment entity description**: A textual description of the environment entity.
- **Environment entity attributes**: A list of important aspects of the environment entity that could be used by later models. For each attribute, the name, type and description are provided.
- **Environment entity roles involved**: A list of roles (defined in the role model) that require the particular environment entity to carry out their activities.

In the example depicted by Figure 4.6, the environment entity *Flight plan* is described as the information that details aircraft landing area, arriving and departure times. In the attribute list, the important aspects that are modelled are the ramp area in which the aircraft will park, and the estimated arrival and departure times. The roles that need the *Flight plan* to conduct their associated activities are detailed in the *Roles involved* list and include amongst others the *Pilot* or the *Crew*.

The validation consists of the following proposals:

1. To add to the model any environment entities defined in the ontology that are not used in the environment model, and to remove those not defined in the ontology.
D1: Flight plan
Name: Flight plan
Environment entity ID: D1
Description: Flight plan is to indicate to where an aircraft is supposed to park and the time an aircraft will stay at an airport.
Attributes:
1. Flight Number: Flight number is used to identify a flight and its data type is string.
2. Airport ID: Identifying the airport in which turnaround happens and its data type is string.
3. Estimated Arrival time, time
4. Estimated departure time, time
5. Ramp area ID, string
Roles involved:
1. Pilot
2. Passenger
3. Crew
4. Airport Ground Staff
5. Airline Ground Staff
6. Fueller
7. Manager
8. Airline Cleaning Staff
9. Airline Catering Staff
10. Engineer

Figure 4.6: An environment model example using the ROADMAP notation

2. To add to the list of involved roles of an environment entity any role that is defined in the ontology as participating in a relation Role uses Environment entity where the Role is not already included in such list, and to remove those that have no associated relationship defined in the ontology.

4.2.5 Agent Model and its Ontology-based Validation Operators

Agent Models define the individual agent classes that will cover the roles defined in the system. Each role has to be mapped to (or played by) at least one agent. Roles are derived from the system goals and are responsible for achieving them. Enforcing the rule that each role is played by at least one agent helps with ensuring that the goals of the system are fulfilled (DeLoach and Kumar 2005). For each agent, the functionalities of the roles that are encompassed by the agent are described as activities, along with their preconditions (or Trigger) and postconditions (or Action). The environment entities required by the roles played by the agent are also included as a list and as a diagram. An example of the agent model depicted by Figure 4.7 shows the agent Crew. As the agent only plays the role Crew, the model includes only the functionalities of the role Crew: embark and disembark the
passengers, embark and disembark themselves, and perform pre-departure cockpit checking. For example, the activity *Disembark crew* refers in its precondition to the finalisation of the passengers’ disembarkation and in its postcondition to the commencement to the cleaning and catering operations. The environment entities listed as used by the agent are the *Aircraft, Airbridge, Flight plan* and *Passenger information*. The ontology validates that activities defined for each agent comply with the specification, and that each agent plays the correct roles and participates in the correct activities using the necessary environment entities. The validation consists of the following proposals:

1. To add to the model set any agents defined in the ontology that do not have corresponding models, and to remove any agent models without a corresponding agent defined in the ontology.

2. To add to every agent model any missing activities associated with any of the roles played by agents, and to remove any listed activities that are not associated with any of the roles played by the corresponding agent (as shown in the ontology).

3. To update the trigger or action fields to correct the preconditions and postconditions of any activity in the ontology whose
preconditions or postconditions do not match any of the ones described by the trigger and action fields (any activity may have several preconditions or postconditions). If fields are incomplete, propose completion with the suitable activities as in the ontology.

4. To add to the environment list in every agent model any missing environment entities that are used by any of the agent roles or in any of the activities in which the roles participate, and to remove any listed environment entity not defined in the ontology as used by any of the agent roles or needed in any of the activities in which the agent participates.

4.2.6 Interaction Model and its Ontology-based Validation Operators

Modelling interactions is a crucial activity in AOSE. The interactions between roles are derived from the responsibilities defined in the role model and show which roles participate in the activities of the system (Padgham and Winikoff 2005; Tran and Low 2008). Responsibilities listed in the role diagrams of two or more roles indicate that the participation of those roles is necessary to accomplish the associated activities. The Interaction Model will set the basis for the definition of protocol models later in the design phase of the system, and will detail the exchange of messages between instances of agents. The example depicted in Figure 4.8 shows two interactions corresponding to the activities Disembark passengers and Embark passengers. While both activities require the participation of the roles Passenger and Crew, in the activity Embark passengers there is an additional participating role, the Airline ground staff. The validation consists of the following proposal:

1. To add to the model any relation linking roles that participate in activities defined in the ontology that are not used in the interaction model, and to remove those not defined in the ontology.
Scenario (Model) and its Ontology-based Validation Operators

Scenarios are detailed descriptions of particular example sequences of events associated with a particular goal or sub-goal of the system (Padgham and Winikoff 2005). Their typical use is with a high level scenario modelling the main goal, and smaller ones modelling the sub-goals in greater detail. Scenarios are described by a name and a unique identifier. Important fields include the goal of the scenario (as defined in the goal model), the role that initiates the sequence of activities and the event that triggers it. The precondition and postcondition of the scenario describe the situation before and after the occurrence of the sequence of events detailed in the scenario. The core of the scenario is its description, with includes the sequence of activities that achieve its goal. The order in which the activities of the scenario are executed is described as parallel or sequential, and for each activity the participating roles are listed. In the example scenario depicted in Table 4.2, the scenario corresponding to the goal Prepare departure is shown. The Pilot is the role that initiates the scenario when the aircraft is ready for departure. The precondition of the scenario is that the aircraft has been refuelled, maintenance activities performed, passengers embarked and baggage loaded. At the end of the scenario, the tug will be attached to the aircraft. All the activities of the scenario have to be conducted sequentially and all of them involve the roles Pilot and Ground staff. The validation consists of the following proposals:

1. To replace the role initiator of the scenario with another role initiator if, in the ontology, the existing role initiator does not
Table 4.2: A scenario model example using the ROADMAP notation

<table>
<thead>
<tr>
<th>Scenario ID</th>
<th>S8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Prepare departure</td>
</tr>
<tr>
<td>Goal</td>
<td>Prepare departure</td>
</tr>
<tr>
<td>Initiator</td>
<td>Pilot</td>
</tr>
<tr>
<td>Trigger</td>
<td>Pilot announces that aircraft is ready for departure</td>
</tr>
</tbody>
</table>
| Precondition| Aircraft has been refuelled  
                            Aircraft has been maintained  
                            Passengers have embarked  
                            Baggage has been loaded |
| Postcondition| Aircraft is ready for departure |
| Description | |

<table>
<thead>
<tr>
<th>Condition</th>
<th>Step</th>
<th>Activity</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>1</td>
<td>Pilot asks the Ground staff to remove the airbridge</td>
<td>Ground staff and Pilot</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Ground staff informs the Pilot that the airbridge has been removed</td>
<td>Ground staff and Pilot</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>The Pilot asks the Ground staff to remove the wheel chocks</td>
<td>Ground staff and Pilot</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>The Ground staff informs the Pilot that the wheel chocks have been removed</td>
<td>Ground staff and Pilot</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>The Pilot asks the Ground staff to attach the tug</td>
<td>Ground staff and Pilot</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>The Ground staff informs the Pilot that the tug has been attached</td>
<td>Ground staff and Pilot</td>
</tr>
</tbody>
</table>
take part in any of the activities that fulfil the goal of the scenario.

2. To add to the model any activity defined in the ontology as pursuing the achievement of the goal of the scenario but not used in the scenario, and to remove those activities not fulfilling the goal of the scenario.

3. To add to the model any relation that is defined in the ontology of activities involving roles if the activity is included in the model but the associated role is not present, and to remove those roles that do not participate in the activity according to the ontology.

The rest of fields are mainly composed of free text and do not necessarily directly match particular elements in the ontology. To support their validation, a report is extracted from the ontology to show the correct order of the execution of activities. The developer will use this report as a guideline to ensure that the scenario correctly models the order of activities and to determine whether they can be executed sequentially or in parallel. Also, the report will be used to check that the trigger and precondition fields are related to the activity preceding the first activity of the scenario, and the postcondition field is related to the activity following the last activity of the scenario.

4.3 PRELIMINARY VALIDATION OF OPERATORS

A development case study is conducted to evaluate the adequacy of the MAS models V&V process and validation operators. The development case consists of an AOSE development experience where the operators are applied to V&V MAS models that are developed using the process detailed in Section 4.1. The development experience is part of a process to develop an agent oriented aircraft turnaround simulator to estimate the delays of large passenger planes between interstate routes. The development process uses the ROADMAP development methodology (Juan et al. 2002).
4.3.1 The Aircraft Turnaround Problem

Aircraft turnaround refers to the process of preparing an arriving aircraft for departure after it lands. Figure 4.9 depicts the activities comprised by the aircraft turnaround as described by the client specification. The aircraft turnaround process commences once the aircraft has landed and taxied to its ramp area and it concludes once the aircraft is prepared to take-off. The manoeuvres of aircraft landing, parking and take-off are out of the scope of the problem domain. The experiment concerns modelling the flow of activities, therefore modelling the activities internally is out of the scope of the experiment.

![Diagram of aircraft turnaround activities]

Figure 4.9: Specification of activities of the aircraft turnaround problem

The aircraft turnaround process commences with the arrival preparation. This includes setting the wheel chocks to ensure that the aircraft will remain stopped and positioning the airbridge to access...
the cockpit of the aircraft. The airport ground staff are responsible for carrying out those two activities following which five parallel workflows (shown in Figure 4.9) are conducted: three workflows refer to technical aircraft maintenance activities, one refers to the handling of passengers and the last refers to the handling of baggage.

The aircraft maintenance engineers and technicians carry out the routine and non-routine maintenance activities. The routine maintenance involves activities such as changing the oil, cleaning the filters and conducting other pre-flight inspections as required by regulations. The non-routine maintenance involves activities such as fixing leaking fluids, worn brakes or burned out light bulbs. Fuelling can be carried out by the airport ground staff and involves refilling the fuel tank for next flight.

The activities related to the baggage handling are also conducted by the airport ground staff. The activities within this workflow must be performed sequentially. First, the ground staff unload the baggage belonging to the arriving passengers. Once the cargo deck is cleared, the next activity in the flow can commence. The ground staff load the baggage belonging to the departing passengers that are embarking at the same time.

The last parallel workflow involves managing the passengers and preparing the cockpit of the aircraft. The first activity is the disembarkation of passengers and then of the crew. Cleaning and catering staff work then in parallel to clean the cabin and cockpit and cater the aircraft. Only when cleaning and catering are completed can the new crew then embark and perform a routine cockpit check (this includes, for example, checking the air-conditioning system). The last activity of the workflow, embarking passengers, is a coordinated effort of the crew (on-board) and the airline ground staff (at the boarding gate).

When all the parallel workflows of the aircraft turnaround are concluded, the final sequence of activities prepares the aircraft for departure. In the first activity, the ground staff remove the airbridge that gives access to the aircraft. The ground staff then remove the wheel chocks that secured the parked aircraft in its ramp area. Finally, the aircraft is ready to take-off so the ground staff attach a tug to position the plane in its assigned departure position.
It is highly desirable to minimise the time that the aircraft remains in the airport, as longer stays mean higher costs for the airline. The MAS simulation is expected to identify how the process can be optimised, completing a faster turnaround with fewer resources (i.e. staff). Turnaround-related operations vary in duration and handover within the sequence of activities. There is scope for decentralisation and parallelisation. This makes the domain an excellent candidate for a MAS simulation. To develop this MAS simulation, the V&V of the models was conducted by one validation team based at the University of Wollongong. The MAS models were supplied by another team of developers that also included developers from the University of Melbourne in the first two iterations.

4.3.2 Task 1 of the V&V Process: Ontology Acquisition

The first task of the ontology-based V&V process is the development or retrieval of an adequate ontology to model the problem domain as conceptualised by the client. A search in Swoogle (2007) showed that although there are some ontologies that model parts or types of aircrafts, none of them was appropriate. The aircraft turnaround process was not modelled by any of them and their description of aircrafts was too detailed for the purpose of developing a simulation that involves only high level features and behavioural descriptions of an aircraft. It was decided that it was actually easier in this case to develop an ontology from scratch based on the documentation that the client provided for the case study, as well as several follow on interviews with them.

An OWL 2 domain ontology was prepared using the development environment Protégé (2012). To avoid unnecessary complexity, only aspects relevant for the MAS application domain are modelled in the ontology. Four subsumption structures are defined under the root of the ontology:

- Aircraft activity defines the type of activities that are conducted as part of the operation of an aircraft, in particular during the turnaround process. Beside the activities defined in Section 4.3.1, activities corresponding to the landing, parking and
taking-off of the aircraft have also been modelled to contextualise the turnaround process.

- **Aircraft component** defines parts of the aircraft that are relevant for the turnaround problem, such as *Aircraft fuel deposit* and *Aircraft cargo hold*.

- **Temporal concept** defines time related concepts relevant for the simulation of the aircraft turnaround process, such as *Shift start time*, *Shift end time* and *Duration*.

- **Turnaround resource** defines the resources that are necessary for the aircraft turnaround process: *Turnaround human resource* and *Turnaround material resource*. The material resources include pieces of equipment used during the turnaround activities, such as *Bulk cargo loader* and *Wheel chocks*. The human resources correspond to the roles participating in the process. For each type of role, a team leader that coordinates the work of the team members has been defined. For example, the class *Airline ground staff* has two subclasses corresponding to the *Airline ground team leader* and *Airline ground worker*.

In addition to the subsumption relations defined by the above taxonomy, the following important relations have been modelled as object properties:

- The property *has a* defines a class as part of another, for example, the subclasses of *Aircraft activity* all have certain *Duration*.

- The properties *precedes* and *follows* are the inverse of each other and establish the sequential order between activities. For example, *Unload baggage precedes Load baggage* and *Load baggage follows Unload baggage*.

- The properties *requires* and *required by* relate activities with the equipment resource that they need and vice versa. For example, the activity *Attach tug requires Tow tractor* and the resource *Tow tractor required by Attach tug*. 
• Similarly, the properties carried out by and in charge of relate activities with their participating roles and vice versa. For example, the activity Catering carried out by Catering staff and the role Catering staff in charge of Catering.

These object properties are used to establish relations between all elements of the domain. Figure 4.10 shows an excerpt of the domain ontology illustrating how the domain classes are interrelated. The class Turnaround activity has been duplicated to improve readability.

![Figure 4.10: Case study 1 ontology excerpt](image)

4.3.3 Task 2 of the V&V Process: Ontology Augmentation

For the next task of the ontology-based V&V process, the domain ontology is augmented to provide it with semantic links to the MAS paradigm. Every domain class is annotated with the corresponding concept in the MAS paradigm. Additional properties typically used in MAS are also added (as discussed in Section 4.1). Figure 4.11 shows how the fragment of the domain ontology depicted by Figure 4.10 is augmented. To improve readability of Figure 4.11, the class Turnaround activity has been duplicated and only the relations associated with the MAS paradigm are shown.

The augmentation is performed manually. The modeller first examines each concept defined in the domain ontology to decide which MAS category it can be classified under. Taking the relationships
defined in the domain ontology as a basis, the elements of each class of concepts are then related using the MAS-dependent object properties summarised in Table 4.1. In the case of the example illustrated by Figure 4.10 and Figure 4.11, the modeller classifies the concepts Position wheel chocks and Position airbridge as instances of the MAS class Activity. The concept Turnaround activity is classified as a Goal. The concepts Wheel chocks and Airbridge are both identified as belonging to the class Environment Entity. The modeller finally decides that the concept Ground staff represents a Role and the corresponding Agent encompassing the functionalities described by the role. Based on this classification and the previous links between concepts in the domain ontology, the modeller adds the new MAS-dependent relations. Relations that were previously defined remain unchanged. For example, as the sequence of activities was already modelled in the domain ontology using the relations follows and precedes, there is no need to add new relations. In some cases, a relation defined in the original domain ontology may have the same semantic as one defined for the augmentation process, for example the relationships Position airbridge requires Airbridge (original relationship) and Position airbridge needs Airbridge (augmented relationship). Finally, there are relations that are derived from the existing knowledge but that were not explicitly

Figure 4.11: Case study 1 augmented ontology excerpt
modelled in the original domain ontology. For example, based on the assertions Position Airbridge needs Airbridge and Position Airbridge fulfils Turnaround activity, the modeller adds new relationships to model that Ground staff uses Airbridge and Ground staff responsible for Turnaround activity.

4.3.4 Task 3 of the V&V Process: Ontology Validation

In this task, the modeller discusses the augmented domain ontology with the client again in case the new semantic annotations trigger further domain input from the client. Although the client and the modeller had agreed earlier upon the general definition of the ontology, a few details of the client’s problem conceptualisation had not been covered by the ontology. The definition of roles presented the greatest opportunity for further client input. The role of the Technician was eliminated, as the client considered it unnecessary for the purpose of the simulation to distinguish between Engineer and Technician. The role of the Passenger was added as a participant to the embarking and disembarking activities. Ground staff was incorrectly listed as the role conducting the refuelling of the aircraft; a new role had to be defined – the Fueller. Finally, a new activity to Allocate resources had to be added to the ontology; this activity is executed by a new role, the Manager. The system had been modelled as having only one goal, the Turnaround activity, however, the client required the problem to be split into different sub-goals covering the Arrival preparation, Aircraft maintenance, Aircraft service, Passenger handling and Baggage handling. The modeller incorporated the changes suggested by the client into the augmented domain ontology before proceeding to Task 4 of the V&V process.

4.3.5 Task 4 of the V&V Process: MAS Software Models Validation – Iteration 1

The models provided for the first iteration of Task 4 of the V&V process were relatively immature and incomplete. The differences between interaction and environment models and the domain onto-
logy were very significant. The goal and role models seemed closer
to the conceptualisation represented by the ontology; this may be
because they lacked detail. The agent model had not been developed
yet, and only a high level general scenario was defined. Because of
the early state of development of the models and major discrepan-
cies with the ontology, a detailed application of the operators was
omitted in favour of providing general guidelines to tackle the most
important modelling problems. Examples of these guidelines that
concerned the organisation, environment and interaction model are:

- The organisation diagrams or roles presented a significant
degree of incompleteness that did not occur in the ontology.
Examples of incompleteness in the model affected the role
diagrams of Engineer and Manager. It was stated that Engineer
is peer Pilot and Manager is peer Pilot, respectively. However, in
the diagram of the Pilot the corresponding relationships were
not modelled.

- The environment models presented differences from the on-
tology in terms of the environment entities definitions, for
example all the personnel involved in the turnaround opera-
tion were defined as environment entities. However, a more
important issue was that the definition of environment entities
did not include the roles involved in their use.

- In the interaction model, the role of the Manager had been
modelled as a mediator for most interactions in the system.
The ontology did not include this role as participating in all in-
teractions. Indeed, having a role coordinating most exchanges
defeated the purpose of developing a MAS. A centralised sys-
tem neutralises some of the most relevant characteristics of
MAS, such as autonomy and robustness.

Recommendations along the above lines were produced for the
developer to improve the MAS models in Task 5 of the ontology-
based V&V process.
4.3.6 Task 5 of the V&V Process: MAS Software Models Improvement – Iteration 1

The developer analysed the recommendation report produced during Task 4 to amend the initial set of MAS models analysed by the developer. Most recommendations were directly accepted. Decisions about the rest of the recommendations were deferred until they could be discussed with the client.

Interaction and environment models underwent major changes to improve their quality. The developer removed the definition of the stakeholders as environment entities from the environment model. Additionally, the roles that used environment entities were compiled and added to the definition of each environment entity. The role interactions in the interaction model were revisited to decentralise the aircraft turnaround process. Although the Manager was still considered an authoritative figure responsible for overseeing the aircraft turnaround, its participation was limited to certain activities such as the allocation of resources prior to the commencement of the turnaround manoeuvre.

In addition to the refinement of the interaction and environment model, the developer extended the general scenario with a detailed description of its sub-scenarios. The agent model was also made available for the first time for the V&V process.

4.3.7 Task 4 of the V&V Process: MAS Software Models Validation – Iteration 2

In the second iteration of Task 4, the ontology-based operators were applied to the set of MAS models that had been refined and expanded by the first iteration of Task 5. The application of the operators revealed that the models that had undergone an iteration of V&V presented few discrepancies with the ontology, i.e. they were more mature and closer to the client conceptualisation. However, a large number of recommendations was generated for the new models, agent model and extended scenario.
Common issues detected for the agent model involved the association of environment entities with agents and the order of activities. Often, environment entities needed by certain agents to fulfil their goals were not included in the corresponding diagrams. For example, although the Agent ground staff required the environment entities Bulk cargo loader and Bulk cargo train to carry on its activities, the diagram did not include the environment entities. Furthermore, the order of execution of activities as specified by the fields trigger and action (the precondition and postcondition respectively) often did not match the sequence described in the ontology. For example, the precondition of the activity Embark aircraft carried out by the agent Crew had to be modified to indicate that the activity can only be conducted after the aircraft cleaning and catering activities were concluded. The postcondition of the activity should be that the crew is on board and can commence the cockpit checking.

With regards to the scenario, the main issues exposed by the application of the operators concerned the participation of roles in activities and the order of execution of activities. Frequently, roles that were modelled as participating in certain activities in the ontology were not included in the scenario as participating in them. For example, the Pilot should have been modelled as participating in the activity Attach tug. In some cases the operators showed that the sequence of activities had gaps. For example, this happened in the sub-scenario describing the set of activities for embarkation of crew and passengers. There were two missing steps at the beginning of the scenario to model that the crew had to embark themselves and check the cockpit before proceeding with embarking the passengers. Differences between the ontology and scenario in terms of activity sequencing also affected the preconditions and postconditions of some scenarios. For example, the precondition of the scenario modelling the baggage handling should have described the activity required before the particular sub-scenario starts (the airbridge positioning has concluded) and the postcondition should refer to the next activity that can be carried out once the sub-scenario finishes (the airbridge is ready for removal).

Earlier recommendations produced during the first iteration of Task 4 helped to prevent compound errors in models initiated in
this second iteration. For example, the recommendation made in the first iteration to remove the mediating role of Manager from most interactions on the interaction model permitted developers to avoid rework by developing the sub-scenarios in a more decentralised fashion. This was possible because scenario and interaction models both include relations between roles and the activities in which they participate. In other words, a recommendation to remove the role Manager from being a participant in the handover between activities Unload baggage and Load baggage on the interaction model prevents the inclusion of the role Manager in the corresponding step of the sub-scenario describing the baggage handling.

4.3.8 Task 5 of the V&V Process: MAS Software Models Improvement – Iteration 2

The recommendations produced by the second iteration of Task 4 were analysed by the developer to refine the MAS models. Unlike the first iteration, the developer did not accept all of the recommendations in the new version of the models. The continuous interaction of the developer with the client during the development process uncovered new requirements and triggered changes to existing ones. The ontology did not reflect these changes as it was based on the initial client conceptualisation. Some of the recommendations issued were based on obsolete aspects of the domain and, therefore, were rejected by the developers. This confirmed the discussion of Section 4.1.5 with regards to different types of discrepancies and the need of revisiting the ontology acquisition task to incorporate new requirements. As the client conceptualisation of the problem had changed, it was no longer appropriate to consider the differences between ontology and models recommendations to improve the models, however they were useful to identify discrepancies that could potentially improve the quality of the models or ontology. The discrepancies highlighted by the application of the operators in the second iteration of Task 4 of the V&V process could be classified according to the four categories discussed in Section 4.1.5: discrepancies due to inconsistency, discrepancies due to incompleteness of the models,
discrepancies due to incompleteness of the ontology and discrepancies for requirements elicitation. The ontology engineer and models developer met to discuss how to address the discrepancies detected by the application of the operators. Any discrepancy of the above described categories could be solved by modifying the ontology or the models. The following discussion provides examples of how the different types of discrepancies were managed.

Incompleteness in the Models

The discrepancies detected in the organisation model involved the symmetric relation is peer. This problem had already been detected in the first iteration of the V&V process, but two discrepancies remained unsolved because of cognitive limitations in the modelling process. The developer and ontology engineer agreed on modifying the model to solve these discrepancies.

Incompleteness in the Ontology

The discrepancies due to incompleteness of the ontology were caused by new requirements that were not part of the original client conceptualisation. To allow the evolution of the ontology to comply with the new conceptualisation, all discrepancies were solved by modifying the ontology. For example, because of the discrepancies with the agent and environment models, the ontology was extended to include new environment entities and their use by agents and roles. This was the case for the environment entity Staff schedule that described the shifts of the personnel involved in the turnaround manoeuvre and its use by Engineer or Ground staff (both roles and agents).

Inconsistencies between ontology and models

Discrepancies due to inconsistency between the ontology and the models were scarcer than the ones aimed at completing either of the artefacts. Some of the discrepancies were solved by changing the models. For example, in the organisation model, it was modelled that Pilot is peer Manager and Manager is peer Pilot. However, in the ontology it was stated that Manager controls Pilot and Pilot controlled
by Manager. The inconsistency was solved by modifying the model as in case of dispute the ultimate authority during the turnaround activity should belong to the Manager.

However, other discrepancies were solved by changing the ontology. For example, the goal hierarchy modelled by the goal model did not fully match the one defined in the ontology. In the ontology, the goal Deboard had two sub-goals, Deboard passenger and Deboard crew, and the goal Board crew and crew check had two sub-goals, Board crew and Check aircraft. The developer argued that splitting the goals into two sub-goals was not necessary for the purpose of the system so the ontology had to change accordingly.

Elicitation of New Requirements

The discrepancies for requirements elicitation introduce a novel use of ontologies and provide an added value to the ontology-based V&V process. The relevance of the new requirements suggested by the V&V process for the purpose of the system under development is to be judged by developers and clients. In the case study, most of the discrepancies were solved by changing the models. It was considered that although the new requirements were relevant for the domain, the added complexity outweighed the benefits for the system being sought.

Examples of this type of discrepancy included:

- Additional detail in the definition of preconditions and post-conditions in the agent model and scenario. For instance, the precondition of activity Remove airbridge included the finalisation of the activities Embark passengers, Load baggage, Fuelling, Routine maintenance and Non-routine maintenance in the ontology. It was decided that the precondition in both models (Embark passengers) was sufficient to fulfil the goals of the client.

- Refinement of roles participating in certain activities. For some activities in the scenario (e.g. Notifying the crew that the aircraft catering had concluded), only a role in a high position in the role hierarchy was specified as participating, i.e. the Manager. The ontology showed that other roles situated lower in the
hierarchy were also involved in the activity (the Catering staff). Developers justified their decision of not changing the models by arguing that, as the included role (Manager) controlled the other roles (Catering staff), functions could be delegated dynamically, and hence, adding the detail of the subordinate roles was not essential.

- Use of environment entities by agents. For example, in the agent model it was detected that in the model the Pilot did not use the Ramp area whereas in the ontology it did. These discrepancies were solved by changing the ontology: the developers considered that it was unnecessary to make the associations explicit because those environment entities were associated with others that were already related to the agents. In the case of the Pilot, it was already included in the model the requirement of the Airbridge, which was considered to be associated with the Ramp area.

Although the developer opted for a conservative approach to decide changes in the models, in some cases the discrepancies were solved by modifying the models. For example, the scenario changed to include several activities (Crew embarking and Crew cockpit checking) that had been defined in the ontology and were considered relevant for the final application. Similarly, the organisation model also changed to model the hierarchical relation between two roles that developers had not considered (Manager controls Crew and Crew is controlled by Manager).

4.3.9 Task 4 of the V&V Process: MAS Software Models Validation – Iteration 3

Before Task 4 of the V&V process could be carried out, the ontology was modified to comply with the new conceptualisation of the problem. The application of the ontology-based operators showed no differences between the ontology and goal, role, organisation, scenario and interaction models. This set of models was considered validated. However, some inconsistencies between the agent and
environment models presented some new discrepancies that had to be addressed. The idea of roles and their corresponding agents using environment entities is represented in the agent and environment models from different perspectives. In the agent model, a list of the environment entities used by each agent is recorded (i.e. arranged by agent). In the environment model, a list of the roles that use each environment entity is provided (i.e. arranged by environment entity). This knowledge was not consistently modelled in both artefacts; for example, the agent diagrams of *Airline cleaning staff, Fueller* and *Engineer* included the use of the environment entity *Aircraft*. However, in the environment model, the *Aircraft* was defined as utilised solely by *Pilot, Passenger* and *Crew*. Although the ontology-based V&V process does not define a direct mechanism to check the inter-model consistency, the ontology provides an indirect way of ensuring the consistency. During the execution of the second iteration of Task 5, the ontology was modified to solve the discrepancies detected with the agent model and the environment model. In particular, the relation *Engineer uses Aircraft* as defined in the agent model was added. When the ontology-based operators were applied in the third iteration, the statement *Engineer uses Aircraft* provoked a discrepancy with the environment model.

4.3.10 Task 5 of the V&V Process: MAS Software Models Improvement – Iteration 3

During the third iteration of Task 5, the developer reviewed the discrepancy report compiled during Task 4. All discrepancies between the ontology and the agent and environment models were solved by modifying the models.

4.3.11 Task 4 of the V&V Process: MAS Software Models Validation – Iteration 4

The ontology-based operators are applied to the MAS models for the last time. No discrepancies between the ontology and the models
were detected. All models had converged with the ontology. They were considered validated and the case study concluded.

### 4.4 Conclusions of the Case Study

This case study represents the first application of the ontology-based V&V process to a real set of MAS models. The MAS models converged with the client conceptualisation only after three iterations. The fourth iteration of Task 4 was necessary to confirm that full convergence had been achieved and that the V&V process had concluded. Allowing the implementation of changes in the ontology to adapt to changes in the problem conceptualisation gave the client scope to better articulate the system requirements. Indeed, the rapid convergence of the MAS models and client conceptualisation suggests that the changes in the ontology were not due to major modifications of the client requirements. Rather it seems that the ontology-based V&V process facilitated the mental process of the client, leading to the improvement of the requirements specification.

The ontology-based operators defined in Section 4.2 were successfully applied to identify discrepancies between the ontology and the models. These discrepancies proved to be useful for detecting modelling flaws in the models, which led to an improvement in the overall quality of models. However, as foreseen in Chapter 3, the manual application of the process introduced a significative overhead that limits the usefulness of the V&V process. Table 4.3 provides an estimation of the overhead introduced by the V&V process broken down into its tasks. As Task 5 involves the development of new MAS models and refinement of the existing ones, its time is not counted as overhead introduced by the process. However, the execution of the second iteration of Task 5 has been included as part of the overhead introduced by the process because of the decision-making discussion held between the developer and ontology engineer, and the subsequent modification of the ontology.

The augmentation of the ontology involved classifying each concept and adding new relationships relevant for the AOSE paradigm. The manual augmentation of the ontology (Task 2) required approxim-
Table 4.3: Case study 1 application times

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1: Ontology Acquisition</td>
<td>4 hrs.</td>
</tr>
<tr>
<td>T2: Ontology Augmentation</td>
<td>4 hrs.</td>
</tr>
<tr>
<td>T3: Ontology Validation</td>
<td>2 hrs.</td>
</tr>
<tr>
<td>T4: Models Validation</td>
<td>2 hrs.</td>
</tr>
<tr>
<td>T5: Models Improvement</td>
<td>—</td>
</tr>
<tr>
<td>Iteration 2</td>
<td></td>
</tr>
<tr>
<td>T4: Models Validation</td>
<td>3 hrs.</td>
</tr>
<tr>
<td>T5: Models Improvement</td>
<td>5 hrs.</td>
</tr>
<tr>
<td>Iteration 3</td>
<td></td>
</tr>
<tr>
<td>T4: Models Validation</td>
<td>3 hrs.</td>
</tr>
<tr>
<td>T5: Models Improvement</td>
<td>—</td>
</tr>
<tr>
<td>Iteration 4</td>
<td></td>
</tr>
<tr>
<td>T4: Models Validation</td>
<td>3 hrs.</td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
<tr>
<td></td>
<td>26 hrs.</td>
</tr>
</tbody>
</table>

At the same time as the original ontology acquisition (Task 1). For larger domains, the effort of preparing the ontology for the V&V process could match or even exceed the time required for the process itself. This was expected (see Chapter 3) and indeed justifies the use of ontology automatic reasoning mechanisms (see Chapter 5). Nearly half of the time spent on the case study was due to the manual application of the ontology-based operators. The manual comparison of the ontology with the MAS models using the operators was another task of the process that was costly but has scope for automation (see Chapter 5). Finally, the change in the client conceptualisation made some of the recommendations made by the application of the operators in the second iteration invalid. This triggered a discussion and modification to the ontology that would have not happened if the requirements specification had been frozen before the start of the modelling process.

The ontology-based V&V process improved the overall quality of the models and avoided rework by preventing compound errors (as discussed in Section 4.3.7). Complex models (namely scenario, agent and environment models) presented more discrepancies than did simpler models (such as the goal or role model). Due to the relative high degree of overlap between models, discrepancies detected for one model reoccurred in others. This means that a decision made
with regards to a certain discrepancy could be applied to solve the depending discrepancies. The detection of dependencies between models will be further studied and automated in Chapter 5.

The case study highlighted important aspects of the ontology-based V&V process worth discussing. Iterations are a paramount aspect of the validation process as this is the mechanism that ensures inter-model consistency. Models are not compared directly with each other for consistency, but indirectly through the ontology. Models often represent aspects of the domain from different points of views. This multi-faceted knowledge must be consistent across models. The ontology provides a unique representation of the domain that can ensure that, when all models comply with it, no inconsistency will be created. The third iteration of the case study illustrated this mechanism. In Task 4, discrepancies between the ontology and the agent and environment models were detected because of the modification of the ontology in the second iteration.

The V&V process has another beneficial effect on overlapping aspects of models: The early validation of models prevents the propagation of errors to other models that have not yet been developed. As the development process proceeds, fixing errors becomes costlier, as more parts of the system are potentially affected. If errors are detected before they have been replicated in other models or in later phases of the lifecycle, rework is avoided and the chance of discovering bugs in systems in production is minimised. An example of this can be found in the second iteration of Task 4, when the extended version of the scenario was developed. This meant that it was possible to avoid the error detected in the first iteration with regard to the interaction model: the overcentralisation of the aircraft turnaround activity through the role of the Manager.

Discrepancies that lead to the identification of new requirements introduce an innovative use of ontologies: as tools for assisting in requirements elicitation. These discrepancies are similar to the discrepancies leading to the identification of incompleteness in the model. However, discrepancies detecting incompleteness were somehow represented in one or another model, whereas the ones eliciting new requirements had not been considered before during the modelling process. The developer highlighted the added value of these
discrepancies as they pointed out facts that had not been taken into account during the initial modelling process. All discrepancies of this type were considered correct, but their relevance to the simulator was individually assessed. Some of them were incorporated in the models, but many others created additional complexity that was unnecessary for the purpose of the final application. This assessment would arguably be very different in the case of developments of critical applications where most fine-grained discrepancies would be relevant for the modelling process and safety of the final system.

Discrepancies leading to complete the models and to elicit new requirements uncovered another innovative benefit of the ontology: an acceleration of the modelling process. If the incomplete aspects are critical for the development of the system, developers will eventually find them without help from the ontology. However, the ontology will speed up this process, potentially reducing the development time and consequently costs. An essential assumption of the V&V process was that the ontology accurately reflects the initial requirements specification and it would remain unchanged throughout the development process. Section 4.1.5 foreshadowed that it might be necessary to revisit the design of the domain ontology to allow the adaptation of the ontology due to changes in the client conceptualisation. The case study has confirmed (as seen in Section 4.3.8) that initial requirements often undergo significant changes as the models evolve and develop. Sometimes, developers’ work uncovers hidden requirements that had not been initially considered by the client. The V&V process should be sufficiently flexible to enable changes in the ontology to adapt to the evolving conceptualisation of the client. In this sense, it could be said that models and ontology co-evolve; they are progressively adjusted until they converge. Once the development of the software has concluded, the ontology remains as part of the documentation, and can be used for maintenance purposes or for the development of similar products.
4.5 \textbf{CHAPTER SUMMARY}

In this chapter the research question identified in the literature review was addressed, proposing an initial version of the ontology-based MAS models V&V process (Section 4.1). A domain ontology was built to model the client conceptualisation of the problem and augmented with knowledge specific to the AOSE paradigm. This ontology is used as a reference to detect discrepancies with the MAS models, i.e. modelling gaps and inconsistencies. Comparison ontology-based operators were developed for a set of MAS analysis models (Section 4.2) that the literature review identified as adequately synthetising the most widely accepted agency features. This chapter also described the first evaluation conducted to assess the adequacy of the V&V process and the ontology-based operators to undertake V&V of MAS models using ontologies (Section 4.3). The evaluation consisted of running a development case study on a real MAS iterative development and interleaving the use of the V&V process. The development case study consisted of four iterations. At the end of the case study, the MAS models were considered validated. The chapter concluded by discussing important aspects highlighted by the case study and describing the next steps of the research. These next steps will be considered in Chapter 5: The definition of the V&V process has to be modified to permit the modification of the ontology when the client changes the conceptualisation of the problem; and automatic reasoning capabilities of ontologies have to be harnessed in the form of an automatic V&V process to alleviate the burden of a manual application. This automated approach had been foreseen as necessary in Chapter 3 and it will focus on the augmentation of the domain ontology, the application of operators and the detection of inter-model dependencies.
The case study presented in Chapter 4 highlighted three key points: (1) the validation and verification (V&V) process must permit the co-evolution of models and ontology, allowing the modification of the domain ontology to comply with changes in the problem conceptualisation; (2) the manual augmentation of the domain ontology limits the scalability of the process; and (3) the dependencies between models are too numerous and complex to maintain manually. Each of these three points identifies a shortcoming in the process. The first shortcoming requires that an improvement be made to the V&V process in order to adapt to changes in the ontology. The second and third shortcomings are addressed by providing automatic support and avoiding the manual application of the V&V process. A decision support tool has therefore been developed to execute a refined version of the comparison operators.

This chapter is organised as follows: Section 5.1 refines the V&V process to deal with the first above shortcoming; Section 5.2 presents an improved version of the MAS comparison operators necessary for automating the V&V process; in Section 5.3, the automatic V&V process covering domain ontology augmentation and operators application is designed; Section 5.4 describes the architecture of the tool that implements that design; and Section 5.5 concludes the chapter.

5.1 CO-EVOLVING THE ONTOLOGY AND THE MAS MODELS

The original V&V process assumed that the conceptualisation of the problem could be frozen before the model development process commenced. This assumed that a correct ontology was used as a reference artefact after being validated by the client. However, the case study in Chapter 4 highlighted that the client’s conceptualisation of the system requirements evolved as the development proceeded.
Discussions between the client and the developer regarding the models under development can facilitate an improved understanding of the problem and lead to the discovery of new, previously hidden, aspects of the system being sought. This additional information changes the requirements specification and needs to be reflected in the MAS models. The added requirements are not necessarily accommodated by the original ontology, and can lead to discrepancies between the original ontology and the MAS models. Thus, the V&V process has to be modified to allow the ontology to evolve, thereby maintaining its consistency with the new version of the requirements specification. Any new requirements will be incorporated into the ontology. When the developer processes the discrepancy report, any discrepancies that have originated from a new requirement are solved by updating the ontology to reflect the new requirements. That is, if a new requirement is added to a model in iteration $i$, the ontology incorporates it in iteration $i+1$. For example, in the case study presented in Chapter 4, the original requirements specification did not include the environment entity *Staff schedule*, therefore the ontology did not define it either. However, interactions between developers and clients led to the addition of this environment entity to the models. When the ontology was compared with the environment model in iteration 2, the removal of this environment entity was recommended, as it was not defined in the ontology. However, as the environment entity was a new requirement of the client, the ontology had to change to incorporate the environment entity. Iteration 3 of the V&V process was based on the updated ontology. Rather than a *recommendation report*, the V&V process generates a *discrepancies report*. Developers analyse this report, in conjunction with the client at times, to decide whether to change the ontology or the model to solve the discrepancies detected. As changes in the ontology are ultimately driven by discussions between the developers and clients, the final set of MAS models will comply with the client’s specification.

The new V&V process is shown in Figure 5.1 (c.f. Figure 4.2 in Chapter 4). The V&V process has two main tasks: *Ontology Preparation* (Task 1 in Figure 5.1) and *MAS Models V&V* (Task 2 in Figure 5.1). Details of these tasks are shown at different levels of detail in Figures 5.2 to 5.5. The process remains independent of the problem domain.
and of the agent oriented software engineering (AOSE) methodology used. The two tasks in the V&V process are still add-ons to the process of the AOSE methodology used. The actual *MAS Models Development* is the core activity of the AOSE methodology and is not part of the V&V process itself. Input to Task 1 is the problem description. Output from Task 1 is a domain ontology validated by the client and augmented with AOSE concepts. This ontology becomes the input to Task 2 and underpins the V&V process. Each MAS model is compared to the ontology to detect discrepancies. The output of Task 2 is then used to improve the quality of both MAS models and the domain ontology. Task 2 is iteratively executed until the MAS models are deemed validated. The key modification to the V&V process in this chapter is the flow of data from Task 2 to Task 1. Upon conclusion of the first iteration of Task 2, it is possible for Task 1 to receive feedback about the ontology as new input. This can trigger modifications in the ontology to adjust to changes in the client conceptualisation, and can potentially lead to further iterations of the process.

![Figure 5.1: Overview of the refined V&V process](image)

The subtasks within Task 1 are shown in Figure 5.2. The first step of the ontology preparation (Task 1) is the *Domain Ontology Acquisition* task (Task 1.1). The input to Task 1.1 is the problem specification, to acquire an appropriate ontology describing the problem domain. The domain ontology can be retrieved from a repository or developed...
from scratch. The domain ontology acquired is then input to Task 1.2, the Domain Ontology Augmentation. This task uses AOSE concepts and relations to produce an augmented ontology where the specific domain knowledge is structured according to the MAS concepts. The process backtracks (to Task 1.1) if the ontology has any design problems, e.g. inconsistencies. Otherwise, the ontology is validated through consultations with the client (Task 1.3). Again, the process can backtrack to Task 1.1 if the client disagrees with some of the knowledge in the ontology. At the end of this process, the ontology is ready to be used to V&V the MAS models in Task 2.

Figure 5.2: Subtasks within Task 1: Ontology Preparation

Task 1.2 is depicted in Figure 5.3. As anticipated in Chapter 3 and confirmed in Chapter 4, completing this task manually adds a significant overhead to the V&V process. To overcome this limitation, an automatic ontology augmentation process is defined here as outlined in Figure 5.3. This will support Task 1.2 through an associated software tool. Details of Task 1.2, leading to this support tool, are provided in Section 5.3. The key underpinning of the automatic process is the use of an AOSE metamodel to structure the domain knowledge. The first step is to merge the domain ontology with an AOSE metamodel (Task 1.2.1). This metamodel is defined, taking into account the most widely used concepts and relationships amongst AOSE methodologies (as shown in the literature review, see Chapter 2). The new merged ontology is then used to automatically explicitly classify the domain knowledge in terms of the MAS concepts (Task 1.2.2). If the ontology presents inconsistencies or if it violates constraints defined by the AOSE metamodel, the execution flow returns to Task 1.1 where the ontology is refined.
Task 2 (Figure 5.4) consists of two subtasks. Task 2.1, *Ontology-Models Comparison*, receives the MAS models and the augmented domain ontology as inputs; it compares them and produces a discrepancies report. As previously discussed, this task is too complex to be processed manually. In this chapter, an automatic comparison operator application mechanism and an associated tool are defined to automate it. The report automatically produced in Task 2.1 is presented to the developer in Task 2.2. The developer uses this report to decide which discrepancies should be fixed by changing the models and which require modifications to the ontology.

Task 2.1 is further detailed in Figure 5.5. Its subtasks are supported by the automatic comparison tool, which is described in Section 5.4. The tool relies on an ontology (*Selection Ontology*) that identifies the elements that should be validated for each model. This ontology also enables detection of shared concepts or relationships across different models to highlight inter-model dependencies. Task 2.1.1
compares each model with the augmented ontology with regards to the elements specified in the Selection Ontology, and based on this it generates a list of discrepancies. In Task 2.1.2, dependencies between models are found and added to the list of discrepancies in the shape of a discrepancy report. This discrepancy report facilitates informed decision making in Task 2.2.

Figure 5.5: Subtasks within Task 2.1: Ontology-Models Comparison

5.2 REFINING THE COMPARISON OPERATORS

As the V&V process is refined to allow the evolution of the ontology, differences between the ontology and the models can no longer be called recommendations. The operators are reformulated to adjust to the refined process. They no longer recommend modifying the models when they do not match the ontology. They only identify the aspects between the ontology and the models that need to correspond. To also facilitate the development of the automatic support, a more efficient approach is used to directly compare elements of the models with the equivalent elements in the ontology. In what follows, the reformulated operators for the commonly used seven AOSE models are described:

Goal Model Operators: These now use a different relationship (pursues instead of responsible for) linking roles and goals to comply with the mechanism defined in Section 5.3. A new operator is also added to ensure that the role definitions match in the ontology and model. The operators ensure the following:
1. Every *Goal* included in the model has been defined in the ontology and vice versa.

2. Every *Role* included in the model has been defined in the ontology and vice versa.

3. Every relation *Goal is sub goal of Goal* included in the model has been defined in the ontology and vice versa.

4. Every relation *Role pursues Goal* included in the model has been defined in the ontology and vice versa.

*Role Model Operators:* To automate the comparison of the role model with the ontology, it is assumed that role responsibilities correspond to role goals. Goals and relationships between roles and goals are part of the operators. The operators ensure the following:

1. Every role model corresponds to a *Role* defined in the ontology and vice versa.

2. Every *Goal* included in any role model has been defined in the ontology and vice versa.

3. Every relation *Role pursues Goal* included in any role model has been defined in the ontology and vice versa.

*Organisation Model Operators* ensure the following:

1. Every *Role* included in any organisation model has been defined in the ontology and vice versa.

2. Every relation *Role is peer Role* included in any organisation model has been defined in the ontology and vice versa.

3. Every relation *Role is controls Role* included in any organisation model has been defined in the ontology and vice versa.

4. Every relation *Role is controlled by Role* included in any organisation model has been defined in the ontology and vice versa.

*Environment Model Operators* ensure the following:

1. Every *Environment Entity* included in the environment model has been defined in the ontology and vice versa.
2. Every Role included in the environment model has been defined in the ontology and vice versa.

3. Every relation Role uses Environment Entity included in the environment model has been defined in the ontology and vice versa.

Agent Model Operators: New operators are added to ensure that roles, activities and environment entities match in the model and the ontology. Relationship operators are also modified to facilitate the automatic mechanism in Section 5.3. They ensure the following:

1. Every Agent included in any agent model has been defined in the ontology and vice versa.

2. Every Role included in any agent model has been defined in the ontology and vice versa.

3. Every Activity included in any agent model has been defined in the ontology and vice versa.

4. Every Environment Entity included in any agent model has been defined in the ontology and vice versa.

5. Every relation Agent plays Role included in any agent model has been defined in the ontology and vice versa.

6. Every relation Agent participates in Activity included in any agent model has been defined in the ontology and vice versa.

7. Every relation Activity1 precedes Activity2 included in the fields Trigger (precondition) or Action (postcondition) of any activity for any agent model has been defined in the ontology and vice versa.

8. Every relation Agent uses Environment Entity included in any agent model has been defined in the ontology and vice versa.

Interaction Model Operators: New operators have been added to ensure that roles and activities match in the ontology and the model. They now ensure the following:
1. Every *Role* included in the interaction model has been defined in the ontology and vice versa.

2. Every *Activity* included in the interaction model has been defined in the ontology and vice versa.

3. Every relation *Role participates in Activity* included in the model has been defined in the ontology and vice versa.

**Scenario (model) operators:** New operators are added to ensure that goals, roles and activities match in the model and the ontology. Relationship operators are also modified to facilitate the automatic mechanism described in Section 5.3. They ensure the following:

1. Every *Goal* included in each scenario has been defined in the ontology and vice versa.

2. Every *Role* included in each scenario has been defined in the ontology and vice versa.

3. Every *Activity* included in each scenario has been defined in the ontology and vice versa.

4. Every relation *Activity pursues Goal* included in every scenario has been defined in the ontology and vice versa.

5. Every *Activity* (*Activity1*) included in the fields *Trigger* and *Precondition* of each scenario is defined in the ontology as preceding the first activity of the scenario (*Activity1 precedes FirstActivity*).

6. Every *Activity* (*Activity2*) in the *Postcondition* of each scenario is defined in the ontology as following the last activity of the scenario (*LastActivity precedes Activity2*).

7. Every relation *Activity1 precedes Activity2* included in any scenario (execution order) has been defined in the ontology and vice versa.

8. Every relation *Role participates in Activity* included in any scenario has been defined in the ontology and vice versa.
5.3 AUTOMATING THE V&V PROCESS

As discussed, two difficulties hinder the manual application of the MAS models V&V process: the effort required in the augmentation of the domain ontology to incorporate MAS concepts; and the extent of the inter-model dependencies. This section discusses these difficulties and the automatic mechanism to stultify them.

Table 5.1 shows an example of how a small set of concepts from the domain ontology would be augmented for the MAS paradigm (only concept classification is shown). The number of elements to be classified is equal to the number of concepts defined in the domain ontology. The augmentation time is thus a polynomial function as depicted in Figure 5.6. Manually augmenting a medium-sized domain ontology has already been shown to be a cumbersome task, limiting the size of the models that can be manually validated. Each iteration of the new process now also allows this task to be revisited, and will likely provoke additional changes in the augmented ontology. Therefore, the already cumbersome task is further complicated. It is virtually impossible for this augmentation to be conducted or maintained manually for large domain ontologies.

<table>
<thead>
<tr>
<th>Augmented Domain Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DOMAIN CONCEPT</strong></td>
</tr>
<tr>
<td>Passenger handling</td>
</tr>
<tr>
<td>Disembark passengers</td>
</tr>
<tr>
<td>Embark passengers</td>
</tr>
<tr>
<td>Crew</td>
</tr>
<tr>
<td>Passenger</td>
</tr>
<tr>
<td>Airline ground staff</td>
</tr>
</tbody>
</table>

A second limit to the manual application of the V&V process is related to the high number of interdependencies between models. Even for a relatively small domain, the overlap between models is significant. Some aspects of the problem are represented from different points of view in more than one model. Modifying a model to fix an error may create inconsistencies with a dependent model
if the changes are not propagated correctly. Consider the scenario depicted in Figure 5.7. The V&V process has been applied to the validated models M1 and M2. Two discrepancies, d1 and d2, have been identified between the models and the ontology. Discrepancy d1 concerns only model M1 and the ontology. Solving discrepancy d1 requires changes in either M1 or the ontology. In any case, there are no further repercussions in M2. Discrepancy d2 involves the ontology and both models, as the aspect in which the ontology and M1 differs is also modelled in M2. Solving d2 requires reviewing M1 and M2 to ensure that no inter-model inconsistency has ensued. Maintaining the consistency between models becomes quickly arduous, especially in complex systems with large models. It is not realistic to rely only on human awareness to propagate changes and maintain a consistent set of models.

The rest of this section details mechanisms that use automatic reasoning with ontologies to overcome the two limitations above.
5.3.1 The AOSE Metamodel

As discussed in Chapter 2, ontologies make it possible to use logical mechanisms underlying automatic reasoning. In particular, OWL 2, which is the ontology language currently recommended by W3C (2009d), fully supports automatic reasoning mechanisms. W3C defines two alternative ways of assigning meaning to OWL 2 ontologies (W3C 2009a). The first uses highly expressive RDF-based semantics. However, using these, it is not always possible to quickly establish the truth or falsehood of a logical statement. The second uses a subset of Description Logics (DL) (Nardi et al. 2007) to define semantics. This limits the expressive power but is much quicker to reason with. The key feature of DL to note here is the difference between Terminological (TBox) and Assertional Box (ABox). The TBox describes the concepts (or classes) and the relationships amongst them. The ABox describes the individuals that belong to the classes and their relationships (Giacomo and Lenzerini 1996). This distinction is important because DL reasoners perform different tasks at each of these levels. The reasoning capabilities that are applied to solve the ontology augmentation problem (see Section 5.3.2) are consistency checking, automatic classification and inference of implicit knowledge:

• Consistency checking refers to the detection of inconsistent knowledge in an ontology. Often ontologies grow organically, guided by an evolving conceptualisation of the domain. New knowledge may contradict previously asserted facts. In large domains where the modelled knowledge is complex, it is not easy to immediately detect that a certain chain of assertions will lead to an inconsistency. These errors typically remain hidden until test or production time, and therefore become costly to fix. For example, assume that in the domain ontology being used for the V&V process it is modelled that the execution of activity $A_1$ always precedes activity $A_2$. Later in the development process, a new requirement is elicited stating that activity $A_0$ precedes $A_1$ and $A_0$ follows $A_2$. Inadvertently, an execution cycle has
been introduced in the process model. The reasoner will be able to identify this as an inconsistency and raise an alarm.

- **Automatic classification** is the capability of the reasoner to discover the category in which concepts or individuals of the ontology fit best. This classification is made through the analysis of the definition of the containing class (intentional definition) and the characteristics of the element to be categorised. It can establish explicit subsumption relationships that were previously only implicit. Assume for instance that it is stated in an ontology that the relation \( R \) always relates elements of the class \( C_1 \) with elements of the class \( C_2 \). Then, if the statement \( A R B \) is added to the ontology, the reasoner will be able to automatically infer that \( A \) belongs to the class \( C_1 \) and \( B \) to class \( C_2 \).

- **Inference of implicit knowledge** is the capability of the reasoner to make explicit knowledge that had previously only been implicitly modelled through the definition of the relationships linking concepts or individuals. The characteristics of relationships that make this knowledge extraction possible are transitivity, symmetry or inverse among others. If an ontology models that the relation \( R_1 \) is the inverse of relation \( R_2 \), and that the individual \( A \) is related with the individual \( B \) through \( R_1 \), then the reasoner can infer that the relationship \( B R_2 A \) is also true.

At TBox level, the reasoner can detect inconsistencies in class definitions with particular regard to class disjointness. For example, the classes *Man* and *Woman* are defined as disjoint: no individual can belong to both classes at the same time. Any subsumption relationships between classes can also be established. For example, the class *Man* is a subclass of the class *Person* and it inherits all the characteristics of the super class (as every *Person* has a heart, every *Man* must also have a heart). At ABox level, the reasoner can classify individuals into their corresponding classes. For example, *John* is an instance of the class *Man*. The reasoner can also infer new relationships based on the explicit relations between individuals. For
example, if the individuals *John* and *Rex* are related through the property *has a pet*, then the reasoner can infer that *Rex* is an instance of the class *Pet* and that *Rex* has an owner called *John* (dependent on the detail on the definition of the property *has a pet*). Finally, the reasoner can also find inconsistencies in the use of properties by individuals. For example, if the property of having a biological mother is defined as having a cardinality of 1 (each person can only have one biological mother), then the assertions *John has biological mother Mary* and *John has biological mother Jane* would be inconsistent (if *Mary* and *Jane* are different individuals).

Common concepts and relationships used to define the comparison operators (presented in Section 5.2) are explicitly represented to enable automatic augmentation of the domain ontology in the V&V process. They are represented as a metamodel (shown in Figure 5.8) which will underpin the automation of the ontology augmentation via automatic reasoning. The metamodel describes typical concepts of the AOSE paradigm as classes (boxes in Figure 5.8), relationships that are modelled as object properties (Figure 5.8: the directed arcs between boxes, with the label being the name of the property) and axioms that establish the constraints of the domain using classes and object properties (Figure 5.8: the triples composed by two boxes and a linking arc). The AOSE metamodel is a subset of the FAML metamodel (Beydoun et al. 2006b,a, 2009). FAML is a generic metamodel that describes most extant AOSE methodologies at analysis and design level. The use of the complete FAML metamodel is out of the scope of this thesis, as the use of all its features at analysis and design level would introduce an unnecessary complexity in the automation of the V&V process. Figure 5.9 details an example axiom of the metamodel: *Activity* and *Goal* are classes, *ActivityAchievesGoal* is the object property, and the axiom is the triple that can be interpreted as ’every activity achieves at least one goal’.

There are six concepts in the metamodel: *Goal*, *Activity*, *Role*, *Agent*, *Event* and *EnvironmentEntity*. The class *Goal* represents the objectives that will be achieved by the system. The class *Activity* represents the tasks that must be performed in order to satisfy certain goals. The class *Role* represents a stakeholder of the system that is able to perform certain functions necessary for the achievement of goals. The
Figure 5.8: AOSE metamodel
class Agent represents an entity in the system capable of performing the functions of one or more roles. The class Event represents a significant occurrence in the environment which activates an agent to pursue its goals, or changes the agent’s course of actions in achieving the goals. Finally, the class EnvironmentEntity represents any resource required to perform the system tasks.

The relationships described in the metamodel are modelled as object properties in the ontology and are commonly used by extant AOSE methodologies (see literature review in Chapter 2). They have been defined by selecting a descriptive base word that describes their purpose, e.g. achieves in Figure 5.9, concatenated to a prefix (Domain) and a suffix (Range). The classes that are described by an object property are called Domain. The classes that characterise the individual state of the domain through the object property are called Range. In the example of Figure 5.9, the domain of the property ActivityAchievesGoal is Activity and its range is Goal. The property can only associate individuals of the class Activity with individuals of the class Goal.

![Diagram](image-url)

Figure 5.9: One of the axioms of the AOSE metamodel

The relationships between classes are better described by means of the 27 axioms that they define. These axioms are constraints over each class that specify the governing rules of the problem domain. An important element of the axiom is the cardinality that defines how many elements of the second class can be associated with each element of the first class. The class Goal defines the following seven axioms:

- Each goal is achieved by at least one activity.
- Each goal is pursued by at least one agent.
- Each goal is pursued by at least one role.
- Each goal may be the sub goal of other goals.
- Each goal may be the super goal of other goals.
• Each goal may need environment entities.

• Each goal may conflict with other goals.

The class Activity defines the following eight axioms:

• Each activity may be a sub activity of other activities.

• Each activity may be the super activity of other activities.

• Each activity achieves at least one goal.

• Each activity involves at least one agent.

• Each activity involves at least one role.

• Each activity may follow other activities.

• Each activity may precede other activities.

• Each activity may need environment entities.

The class Role defines the following nine axioms:

• Each role may be a sub role of other roles.

• Each role may be the super role of other roles.

• Each role participates in at least one activity.

• Each role is played by exactly one agent.

• Each role pursues at least one goal.

• Each role may be peer with other roles.

• Each role may be controlled by other roles.

• Each role may control other roles.

• Each role may use environment entities.

The class Agent defines the following five axioms:

• Each agent plays at least one role.

• Each agent participates in at least one activity.
• Each agent pursues at least one goal.

• Each agent may react to events.

• Each agent may use environment entities.

The class *Environment Entity* defines the following four axioms:

• Each environment entity is needed for at least one goal.

• Each environment entity is needed by at least one activity.

• Each environment entity is used by at least one role.

• Each environment entity is used by at least one agent.

The class *Event* defines the following axiom:

• Each event stimulates at least 1 agent.

The axioms show that some object properties are the inverse of others. This means that if two properties, \( p_1 \) and \( p_2 \), are inverse of each other and it is asserted that individual \( i_1 \) participates through the property \( p_1 \) with the individual \( i_2 \), then it can be automatically inferred that individual \( i_2 \) participates through the property \( p_2 \) with individual \( i_1 \). Table 5.2 summarises the inverse properties as defined in the metamodel.

Depending on the cardinality of the axiom, different OWL 2 constructs have been used. There are three different cardinalities in the metamodel. Elements of class \( A \) participate through the relation \( p \) with exactly one element of class \( B \), with one or more, or with zero or more. Examples of these cases are correspondingly:

• Each role is played by exactly one agent.

• Each role pursues at least one goal.

• Each role may control other roles.

OWL 2 constructs that express these are restrictions on property cardinality, universal and existential quantifier. The existential quantifier, \( \text{ObjectSomeValuesFrom} \), is read as "some". When applied to an
Table 5.2: AOSE metamodel inverse properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Inverse</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentPlaysRole</td>
<td>RolePlayedByAgent</td>
</tr>
<tr>
<td>AgentPursuesGoal</td>
<td>GoalPursuedByAgent</td>
</tr>
<tr>
<td>AgentParticipatesInActivity</td>
<td>ActivityInvolvesAgent</td>
</tr>
<tr>
<td>AgentUsesEnvironmentEntity</td>
<td>EnvironmentEntityUsedByAgent</td>
</tr>
<tr>
<td>RolePursuesGoal</td>
<td>GoalPursuedByRole</td>
</tr>
<tr>
<td>RoleParticipatesInActivity</td>
<td>ActivityInvolvesRole</td>
</tr>
<tr>
<td>RoleUsesEnvironmentEntity</td>
<td>EnvironmentEntityUsedByRole</td>
</tr>
<tr>
<td>RoleControlsRole</td>
<td>RoleControlledByRole</td>
</tr>
<tr>
<td>GoalAchievedByActivity</td>
<td>ActivityAchievesGoal</td>
</tr>
<tr>
<td>GoalNeedsEnvironmentEntity</td>
<td>EnvironmentEntityNeededForGoal</td>
</tr>
<tr>
<td>GoalIsSubgoalOfGoal</td>
<td>GoalIsSupergoalOfGoal</td>
</tr>
<tr>
<td>ActivityNeedsEnvironmentEntity</td>
<td>EnvironmentEntityNeededInActivity</td>
</tr>
<tr>
<td>ActivityPrecedesActivity</td>
<td>ActivityFollowsActivity</td>
</tr>
<tr>
<td>ActivityIsSubActivityOfActivity</td>
<td>ActivityIsSuperActivityOfActivity</td>
</tr>
<tr>
<td>RoleIsSubRoleOfRole</td>
<td>RoleIsSuperRoleOfRole</td>
</tr>
<tr>
<td>AgentReactsToEvent</td>
<td>EventStimulatesAgent</td>
</tr>
</tbody>
</table>

axiom $C_1 p C_2$, it means that for every element of class $C_1$, there will be at least one assertion relating it through the property $p$ with an individual of class $C_2$. It can also be part of other assertions related through the property $p$ with elements of other classes. This quantifier is typically used to model incomplete knowledge. The universal quantifier, $\text{ObjectAllValuesFrom}$, can be read as "only". When applied to an axiom $C_1 p C_2$, it means that if an element of class $C_1$ participates in an assertion with the property $p$, it necessarily has to be related to an element of class $C_2$. However, this quantifier does not force the existence of individual participation in the condition.

The exact cardinality restriction, $\text{ObjectExactCardinality}$, when applied to an axiom $C_1 p N C_2$, means that for every element of class $C_1$, there will be exactly $N$ assertions relating it through the property $p$ with an individual of class $C_2$. However, it does not imply that there are other assertions relating an individual of class $C_1$ through the property with individuals of other classes different from $C_2$. The axiom type "zero or more" can be modelled through the direct application of the universal quantifier as shown in Listing 5.1 (the IRIs have been truncated to improve readability). However, to express the
Listing 5.1: OWL 2 code for zero or more cardinality

```xml
<SubClassOf>
  <Class IRI="#Role"/>
  <ObjectAllValuesFrom>
    <ObjectProperty IRI="#RoleControlsRole"/>
    <Class IRI="#Role"/>
  </ObjectAllValuesFrom>
</SubClassOf>
```

Listing 5.2: OWL 2 code for one or more cardinality

```xml
<SubClassOf>
  <Class IRI="#Role"/>
  <ObjectIntersectionOf>
    <ObjectSomeValuesFrom>
      <ObjectProperty IRI="#RolePursuesGoal"/>
      <Class IRI="#Goal"/>
    </ObjectSomeValuesFrom>
    <ObjectAllValuesFrom>
      <ObjectProperty IRI="#RolePursuesGoal"/>
      <Class IRI="#Goal"/>
    </ObjectAllValuesFrom>
  </ObjectIntersectionOf>
</SubClassOf>
```

Axiom type "one or more" the existential and universal quantifiers must be used in combination. For instance, in the case of the axiom "each role pursues at least one goal", it is necessary to state that a role pursues at least one goal (existential quantifier) and only a goal, no other class (universal quantifier). Listing 5.2 (the IRIs have been truncated to improve readability) shows the OWL code corresponding to this axiom. Similarly, the exact cardinality restriction (illustrated by Listing 5.3 with truncated IRIs to improve readability) needs the universal quantifier to constraint the value of the object individual.

A more versatile alternative to the above is the OWL 2 specification of property chain. This is a composition of several properties, where the object of an assertion becomes the subject of the next one. A typical example is the definition of the property hasUncle as the composition of the properties hasFather and hasBrother (shown in Figure 5.10, which uses a dashed line to denote the inferred property). If John, Michael and Paul are instances of the class Person, and John
5.3 Automating the V&V Process

Listing 5.3: OWL 2 code for exactly one cardinality

```xml
<SubClassOf>
    <Class IRI="#Role"/>
    <ObjectIntersectionOf>
        <ObjectAllValuesFrom>
            <ObjectProperty IRI="#RolePlayedByAgent"/>
            <Class IRI="#Agent"/>
        </ObjectAllValuesFrom>
        <ObjectExactCardinality cardinality="1">
            <ObjectProperty IRI="#RolePlayedByAgent"/>
            <Class IRI="#Agent"/>
        </ObjectExactCardinality>
    </ObjectIntersectionOf>
</SubClassOf>
```

hasFather Michael and Michael hasBrother Paul, then it can be inferred that John hasUncle Paul. In the context of the AOSE metamodel, property chains are defined to make the inference of implicit knowledge possible based on explicit assertions. Table 5.3 shows the properties that have been defined in the metamodel as a composition of others. The symbol ‘◦’ is used to express that the property that precedes it is combined with the property that follows it. The symbol ‘≡’ expresses the idea of equivalence. These property chains have been defined by analysing the metamodel to discover which properties were always equivalent to the combination of others. Figure 5.11 depicts an excerpt of the metamodel illustrating one of the property chains for the object property AgentPursuesGoal (the dashed line denotes the inferred property).

![Figure 5.10: Example of property chain](image)

Figure 5.10: Example of property chain
### Table 5.3: Metamodel property chains

<table>
<thead>
<tr>
<th>Property</th>
<th>Equivalent Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActivityAchievesGoal</td>
<td>ActivityAchievesGoal \circ GoalIsSubGoalOfGoal \circ ActivityIsSubActivityOfActivity \circ ActivityAchievesGoal</td>
</tr>
<tr>
<td></td>
<td>ActivityInvolvesAgent \circ RolePlayedByAgent \circ ActivityIsSuperActivityOfActivity \circ ActivityInvolvesAgent</td>
</tr>
<tr>
<td>ActivityInvolvesRole</td>
<td>ActivityIsSuperActivityOfActivity \circ ActivityInvolvesRole \circ ActivityInvolvesRole</td>
</tr>
<tr>
<td>AgentParticipatesInActivity</td>
<td>AgentPlaysRole \circ RoleParticipatesInActivity</td>
</tr>
<tr>
<td>AgentPursuesGoal</td>
<td>AgentPlaysRole \circ RolePursuesGoal \circ GoalIsSubGoalOfGoal \circ AgentPlaysRole \circ RoleParticipatesInActivity \circ ActivityAchievesGoal \circ GoalIsSubGoalOfGoal \circ AgentParticipatesInActivity \circ ActivityAchievesGoal \circ AgentParticipatesInActivity \circ ActivityAchievesGoal \circ AgentPlaysRole \circ RoleParticipatesInActivity \circ ActivityAchievesGoal \circ AgentPlaysRole \circ RoleParticipatesInActivity \circ ActivityIsSubActivityOfActivity \circ ActivityAchievesGoal</td>
</tr>
<tr>
<td>AgentUsesEnvironmentEntity</td>
<td>AgentPlaysRole \circ RoleUsesEnvironmentEntity</td>
</tr>
<tr>
<td>EnvironmentEntityNeededForGoal</td>
<td>≡</td>
</tr>
<tr>
<td>EnvironmentEntityNeededForGoal</td>
<td>≡</td>
</tr>
<tr>
<td>EnvironmentEntityNeededForGoal</td>
<td>≡</td>
</tr>
<tr>
<td>EnvironmentEntityNeededForGoal</td>
<td>≡</td>
</tr>
<tr>
<td>EnvironmentEntityUsedByAgent</td>
<td>≡</td>
</tr>
<tr>
<td>GoalAchievedByActivity</td>
<td>≡</td>
</tr>
<tr>
<td>GoalNeedsEnvironmentEntity</td>
<td>≡</td>
</tr>
<tr>
<td>GoalNeedsEnvironmentEntity</td>
<td>≡</td>
</tr>
<tr>
<td>GoalPursuedByAgent</td>
<td>≡</td>
</tr>
<tr>
<td>GoalPursuedByAgent</td>
<td>≡</td>
</tr>
<tr>
<td>GoalPursuedByAgent</td>
<td>≡</td>
</tr>
<tr>
<td>GoalPursuedByAgent</td>
<td>≡</td>
</tr>
<tr>
<td>GoalPursuedByAgent</td>
<td>≡</td>
</tr>
<tr>
<td>GoalPursuedByAgent</td>
<td>≡</td>
</tr>
<tr>
<td>GoalPursuedByRole</td>
<td>≡</td>
</tr>
<tr>
<td>GoalPursuedByRole</td>
<td>≡</td>
</tr>
<tr>
<td>GoalPursuedByRole</td>
<td>≡</td>
</tr>
</tbody>
</table>
RolePursuesGoal ≡ RoleParticipatesInActivity ◦ ActivityAchievesGoal
≡ RoleParticipatesInActivity ◦ ActivityAchievesGoal ◦ GoalIsSubGoalOfGoal
≡ RolePursuesGoal ◦ GoalIsSubGoalOfGoal
≡ RoleIsSubRoleOfRole ◦ RoleParticipatesInActivity ◦ ActivityAchievesGoal ◦ ActivityIsSubActivityOfActivity
≡ RoleParticipatesInActivity ◦ ActivityIsSubActivityOfActivity ◦ ActivityAchievesGoal
≡ RoleIsSubRoleOfRole ◦ RoleParticipatesInActivity ◦ ActivityAchievesGoal
5.3 Automating the V&V Process

Figure 5.11: Definition of property chain for AgentPursuesGoal

Figure 5.12 illustrates how the knowledge involving a set of individuals is affected by this property chain (dashed lines denote the inferred knowledge). Ag₁ is an instance of the class Agent, R₁ of the class Role, A₁ of the class Activity and G₁, G₂ and G₃ instances of the class Goal. The asserted relations between the individuals are the ones depicted in Figure 5.12 by solid lines: Ag₁ plays R₁, R₁ participates in A₁, A₁ achieves G₁, G₁ is a sub goal of G₂ and G₂ is in turn a sub goal of G₃. In this situation, the reasoner can automatically establish the logical implications of the property chain: Ag₁ plays R₁, R₁ participates in A₁, A₁ achieves G₃ and G₃ is a sub goal of G₂. Consequently it will conclude that Ag₁ pursues G₂. As the property is sub goal of is transitive and G₂ is a sub goal of G₁, Ag₁ also pursues G₁. However, this property chain does not lead to the inference that Ag₁ pursues G₃, because the last link of the chain, GoalIsSubgoalOfGoal, always has to be followed to conclude the relation AgentPursuesGoal. Another property chain without the last link, GoalIsSubgoalOfGoal, makes the inference Ag₁ pursues G₃ possible.

Equivalent property chains have not been defined for every object property because the truth of the equivalence can not always be assured. For example, it could be considered that the object property ActivityNeedsEnvironmentEntity is equivalent to the property chain ActivityInvolvesRole and RoleUsesEnvironmentEntity, because if a role participates in an activity and the role uses an environment entity, then that environment entity will be necessary for that activity. However, a role may participate in two activities, A₁ and A₂, and use several environment entities for A₂. That does not mean that A₁ needs those environment entities as well.
Figure 5.12: Example of inference for the one of the AgentPursuesGoal property chains

5.3.2 Revisiting the Ontology Augmentation

The AOSE metamodel alleviates the burden of the manual augmentation of the domain ontology. Instead of classifying each concept defined in the domain ontology into its corresponding category, the focus is shifted to the object properties defined in the domain ontology. In large domains, the number of concepts will grow accordingly. However, the number of object properties linking them will remain constant. The number of object properties defined in the AOSE metamodel is the upper limit for the number of properties relevant for the V&V process in the domain ontology. Every property defined in the domain ontology can be established as a sub property (i.e. a subtype) of the corresponding property in the metamodel. Whenever two instances are related through the sub property, the reasoner can infer that they are also related through the super property. Intuitively, if hasSon is a sub property of hasChild, and if it is asserted that Michael hasSon John, the reasoner will infer that Michael hasChild John. However, asserting that two individuals are related through the super property, e.g. Michael hasChild Jane, does not lead the reasoner to infer that John hasSon Jane. A consequence of this is that the sub property also has the characteristics defined for the super class, in particular the domain and range. In the previous example, if the property hasChild has been defined as having the class
5.3 Automating the V&V Process

Table 5.4: Sub-property mapping

<table>
<thead>
<tr>
<th>Domain Property</th>
<th>Metamodel Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>subgoalOf</td>
<td>sub property of</td>
</tr>
<tr>
<td>achieves</td>
<td>sub property of</td>
</tr>
<tr>
<td>precedes</td>
<td>sub property of</td>
</tr>
<tr>
<td>participatesIn</td>
<td>sub property of</td>
</tr>
<tr>
<td>needs</td>
<td>sub property of</td>
</tr>
<tr>
<td>subgoalOf</td>
<td>GoalIsSubgoalOfGoal</td>
</tr>
<tr>
<td>achieves</td>
<td>ActivityAchievesGoal</td>
</tr>
<tr>
<td>precedes</td>
<td>ActivityPrecedesActivity</td>
</tr>
<tr>
<td>participatesIn</td>
<td>RoleParticipatesInActivity</td>
</tr>
<tr>
<td>needs</td>
<td>ActivityNeedsEnvironmentEntity</td>
</tr>
</tbody>
</table>

*Parent* as the domain and the class *Child* as range, then as *Michael* and *John* are related through the property *hasSon*, the reasoner will infer that *Michael* and *John* belong to the classes *Parent* and *Child* respectively. This is extrapolated to the AOSE metamodel, where the characteristics of object properties – domain and range, amongst others – are already defined. Figure 5.13 shows an excerpt of a domain ontology for the aircraft turnaround problem (the individuals *Bulk Cargo Loader* and *Train Cargo Loader* have been duplicated to improve readability). It is relatively lightweight, and hence quick to develop. No characteristics have been defined for the properties and all concepts are undistinguished individuals (instances of the generic class *Thing*). The object properties defined in this ontology are established as sub properties of the ones defined in the metamodel (Figure 5.8) according to Table 5.4.

![Figure 5.13: Example of domain ontology](image)

*Figure 5.13: Example of domain ontology*
Table 5.5: Domain and range of metamodel properties

<table>
<thead>
<tr>
<th>Metamodel Property</th>
<th>Domain</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>GoalIsSubgoalOfGoal</td>
<td>Goal</td>
<td>Goal</td>
</tr>
<tr>
<td>ActivityAchievesGoal</td>
<td>Activity</td>
<td>Goal</td>
</tr>
<tr>
<td>ActivityPrecedesActivity</td>
<td>Activity</td>
<td>Goal</td>
</tr>
<tr>
<td>RoleParticipatesInActivity</td>
<td>Role</td>
<td>Activity</td>
</tr>
<tr>
<td>ActivityNeedsEnvironmentEntity</td>
<td>Activity</td>
<td>Environment Entity</td>
</tr>
</tbody>
</table>

In the metamodel, object properties specified in Table 5.4 have been defined as having the domain and ranges shown by Table 5.5. Due to the relation between super and sub properties, the object properties defined in the domain ontology share their domain and range with their super properties (shown in Table 5.6). The reasoner can then classify the domain ontology individuals as instances of the metamodel classes corresponding to the domain and range of the property to which they relate, as depicted by Table 5.7.

With this change of perspective, the number of elements to be classified does not grow with the size of the domain, but remains constant and limited to the maximal number of relevant properties for the MAS paradigm. Another benefit of this approach comes in terms of consistency checking. In the metamodel, the classes Goal, Activity, Role, Agent, Event and EnvironmentEntity have been defined as disjoint classes. This means that an individual belonging to the class Goal cannot belong to any other disjoint class. Assume that the domain ontology contains the assertions Load baggage needs Bulk cargo loader and Load baggage subgoalOf Aircraft turnaround. When the reasoner tries to classify the instances an inconsistency is triggered, as Load baggage has to be classified as an individual belonging to both classes Activity and Goal. This happens because the domain of the property subgoalOf is Goal and the domain of needs is Activity, therefore an individual participating as subject in both properties has to simultaneously belong to classes Goal and Activity, which is explicitly forbidden by the disjointness axioms.

The domain ontology has been defined as a set of individuals (ABox) instead of as a set of classes (TBox) to be able to infer implicit knowledge. Only at ABox level can the reasoner make explicit pre-
Table 5.6: Inferred domains and ranges of the domain ontology properties

<table>
<thead>
<tr>
<th>Domain Property</th>
<th>Domain</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>subgoalOf</td>
<td>Goal</td>
<td>Goal</td>
</tr>
<tr>
<td>achieves</td>
<td>Activity</td>
<td>Goal</td>
</tr>
<tr>
<td>precedes</td>
<td>Activity</td>
<td>Goal</td>
</tr>
<tr>
<td>participatesIn</td>
<td>Role</td>
<td>Activity</td>
</tr>
<tr>
<td>needs</td>
<td>Activity</td>
<td>Environment Entity</td>
</tr>
</tbody>
</table>

Table 5.7: Domain concepts classification into AOSE classes

<table>
<thead>
<tr>
<th>Metamodel Class</th>
<th>Domain Individuals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Aircraft Turnaround, Handle Baggage</td>
</tr>
<tr>
<td>Activity</td>
<td>Unload Baggage, Load Baggage</td>
</tr>
<tr>
<td>Role</td>
<td>Airport Ground Staff</td>
</tr>
<tr>
<td>Environment Entity</td>
<td>Bulk Cargo Loader, Train Cargo Loader</td>
</tr>
</tbody>
</table>

Previously implicit knowledge as new assertions. For example, given two inverse properties \( p_1 \) and \( p_2 \), and two individuals \( i_1 \) and \( i_2 \), if it is asserted that \( i_1 p_1 i_2 \), then the reasoner can infer that \( i_2 p_2 i_1 \). This implicit knowledge cannot be extracted at TBox level. A significant number of inverse properties (16) and property chains (37) has been defined in the metamodel. If there is a minimal set of connections linking all classes, most of the missing connections can be inferred through the inverse properties and property chains. Figure 5.14 illustrates a minimal set of properties that would make inferring the rest of relations possible. Solid lines represent the type of relations that are explicitly asserted between individuals. Dashed lines denote the assertions that the reasoner can infer based on the asserted knowledge. However, it is not possible to define property chains for every property in the metamodel (see discussion in Section 5.3.1). This means that not every configuration of minimal sets of connections will permit the extraction of all knowledge.

For each individual defined in the ontology that participates in relations involving only 13 types of domain ontology properties, 21 additional properties will be automatically inferred. This enriches the domain ontology considerably, producing a more complete artefact for V&V and to complete the MAS models.
5.3.3 Operators Application and Inter-model Dependency Management

The operators presented in Section 5.2 are operationalised in a selection ontology that relates models with MAS concepts and relationships. This ontology details the concepts and relationships that have to be extracted from the domain ontology and then compared with each MAS model for the V&V process. In the excerpt of the selection ontology depicted in Figure 5.15, classes are represented as rounded rectangles and their individuals (concepts, relationships or models) as rectangles. From Figure 5.15, it can be seen that the goal model uses the concepts Role and Goal and the relationships RolePursuesGoal, GoalIsSuperGoalOfGoal and GoalIsSubGoalOfGoal. The selection ontology also facilitates the identification of elements used in the different MAS models. For example, role and goal models share the use of the concepts Role and Goal and the relationship RolePursuesGoal. It can be said that both models are co-dependent in terms of these elements. The modification of any role in the role model must be propagated accordingly to the same role in the goal model. The selection ontology can thus be used during the V&V process to identify the models that need to be revisited as a consequence of deciding that concepts or relationships have been ill-defined in others.
Table 5.8 summarises the concepts and relations that are used in each model, as defined in the selection ontology. Names of concepts and relations correspond to the ones defined in the metamodel presented in Section 5.3.1.

The next section outlines how the description of the automated process discussed in Section 5.3 is used to create a support tool for the V&V process.

5.4 A SUPPORT TOOL FOR AUTOMATING THE V&V PROCESS

The programming language used to develop the support tool was Java. The main reason for selecting Java was that it is the only language in which the most widely used APIs, the OWL API (Horridge and Bechhofer 2011) to manipulate ontologies and the HermiT API (Motik et al. 2012) to provide reasoning mechanisms, can interact with ontologies. The OWL API has been extensively used. In this thesis, only the details of the Java implementation that are relevant to justify certain decisions are discussed. The focus is on describing the tool. It is composed of two modules. The first module augments
the domain ontology, which is then used by the second module for V&V of the MAS models.

5.4.1 Domain Ontology Augmentation Module

The first module automates the augmentation of the domain ontology with AOSE related concepts and relationships. Section 5.1 described Task 1.2: Domain Ontology Augmentation as composed of two subtasks (Figure 5.3): Task 1.2.1: Domain Knowledge Scaffolding and Task 1.2.2: Automatic Inference. These two subtasks are automated by the domain ontology augmentation module. However, because of certain characteristics of the reasoners and of OWL\textcopyright 2 itself (discussed below), the reasoner cannot by itself produce the augmented
ontology and additional steps are also implemented. Indeed, only steps 3 and 6 correspond to applications of the reasoner, requiring the support of the tasks conducted in the rest steps. While Task 1.2.1 can be directly mapped to Step 1 of the domain ontology augmentation module (see below), Steps 2 to 7 are necessary to perform the automatic inference as described by Task 1.2.2 (shown in Figure 5.16). The architecture of the module divides its functionalities into seven packages according to the steps depicted in Figure 5.16. A simple text-based interface provides access to the augmentation mechanism. For the purpose of this thesis, the functionality of the tool is described in terms of Steps 1 to 7, as follows:

![Figure 5.16: Mapping between domain ontology augmentation module and V&V process](image)

**Step 1: Merge of Metamodel and Domain Ontology** initiates the augmentation process by merging its two inputs, the lightweight domain ontology and the formally defined AOSE metamodel. To enable the application of the automatic reasoner, the object properties of the domain ontology are established as sub object properties of the corresponding ones in the metamodel. This is translated in the ontology by the creation of new relationships of the form DomainObjectProperty subObjectPropertyOf MetamodelObjectProperty. For the purpose
of the proof of concept developed for this thesis, it is assumed that the name of every property in the domain ontology is a fragment of the corresponding property in the MAS metamodel. The mapping of properties is thus automated using a simple string-comparison-based search. The output of this step is a new ontology that includes the AOSE metamodel (with its classes, object properties and axioms) and the domain ontology (with its individuals and object properties assertions).

**Step 2: Addition of Implicit Hierarchy Relations** is motivated by a limitation of existing reasoners: An object property cannot be defined as asymmetric and transitive at the same time. The AOSE metamodel defines the object property $\text{RoleControlsRole}$ and its inverse $\text{RoleControlledByRole}$ to establish hierarchy relations between roles. For example, the roles $\text{Pilot}$ and $\text{Crew}$ are related as $\text{Pilot RoleControlsRole Crew}$ and $\text{Crew RoleControlledByRole Pilot}$. Semantically these relations should be asymmetrical and transitive. If the symmetric relation $\text{Pilot RoleControlsRole Crew}$ does not hold, then $\text{Crew RoleControlsRole Pilot}$. Similarly, if $\text{Pilot RoleControlsRole Crew}$ and $\text{Crew RoleControlsRole Passenger}$, then the relation $\text{Pilot RoleControlsRole Passenger}$ also holds. However, as the reasoner does not support the definition of object properties as asymmetric and transitive simultaneously, the metamodel defines these properties only as asymmetric. This has a certain impact for the purpose of the ontology augmentation, as shown in [Figure 5.17](#).

![Figure 5.17: Role hierarchy inconsistency](#)
Figure 5.17 (a) shows an initial situation where four roles have been defined, Role1, Role2, Role3 and Role4, along with control relationships between them (the object property RoleControlsRole has been shortened to controls to facilitate the discussion): Role1 controls Role2, Role2 controls Role3 and Role3 controls Role4. By defining the object property controls as transitive, the reasoner can automatically infer the relations Role1 controls Role3, Role1 controls Role4 and Role2 controls Role4 (depicted by dashed lines in Figure 5.17 (b)). If a new relation Role4 controls Role1 is asserted (Figure 5.17 (c)), the reasoner will detect that, as controls is asymmetric, it is inconsistent with one of the previous inferred relations, Role1 controls Role4. It is therefore necessary to define the object property controls as transitive to enable the extraction of implicit knowledge and asymmetric relationships to detect inconsistencies. Step 2 overcomes the limitation of the reasoner by analysing the asserted relationships involving the object properties RoleControlsRole and RoleControlledByRole and adding the assertions that should have been inferred if the properties were transitives. In the example of Figure 5.17, Step 2 would add the relations denoted by dashed lines with regards to RoleControlsRole.

Step 3: Initial Classification and Relation Inference uses the ontology produced in Step 2 as input to the automatic reasoner. There are four essential tasks that the reasoner performs in this step, namely, consistency checking, individual classification, extraction of implicit assertions and materialisation of the resulting ontology. Typically, the reasoner will discover inconsistencies where the domain ontology defines individuals as simultaneously belonging to disjoint classes or when asserted object properties violate the rules defined by the metamodel (see example discussed in relation to Figure 5.17). The existence of inconsistencies in the domain ontology interrupts the augmentation process until these inconsistencies are eliminated. If the ontology is free of inconsistencies, the reasoner automatically classifies the individuals as instances of the AOSE metamodel classes (see discussion in Section 5.3.2). Additionally, the knowledge that has only been implicitly defined is made explicit. For example, given the object property assertions defined in Figure 5.17 (b), and as it has been modelled that controls is the inverse of controlledBy, the reasoner will generate the relations: Role4 controlledBy Role3, Role4

...
controlledBy Role2, Role4 controlledBy Role1, Role3 controlledBy Role2, Role3 controlledBy Role1 and Role2 controlledBy Role1. The knowledge that has been inferred by the reasoner during this step only exists in its working memory. The final task of the reasoner during this step is to materialise the inferred knowledge into an output ontology, which is further processed in later steps.

Step 4: Augmentation with Relations between Roles and Environment Entities tackles the problem discussed in Section 5.3.1. It is not possible to add a property chain to infer certain relations unconditionally. In particular, this step seeks to complete the ontology to some extent by making explicit assertions that link individuals belonging to the class Role with instances of the class EnvironmentEntity through the object property RoleUsesEnvironmentEntity. The module examines the ontology to identify the activities that have only one participating role. The environment entities used in those activities can be associated with the role participating in the activity, through the relationship RoleUsesEnvironmentEntity, to produce the output ontology.

Step 5: Augmentation with Agents creates agents covering the functionalities defined for the roles, and adds them to the ontology. The original domain ontology defines the stakeholders of the problem as roles. Agents are the realisations of the roles at design time and therefore they are not part of the client conceptualisation. However, the concept of agents is essential in AOSE and is involved in a significant number of relationships in the AOSE metamodel. In this step, the module adopts a simplistic approach and adds an agent (as an individual) for each existing role in the ontology. In more sophisticated approaches, a developer can consider that the functionalities of several roles may be covered by a single agent.

Step 6: Second Reasoning to Infer Agent-related Relations uses the ontology produced in the previous step as input to the reasoner to make explicit assertions in relation to the agents. First, all new individuals are classified as members of the class Agent. Then, assertions involving roles are extended to include agents, i.e. if Agent1 plays Role1, new assertions are created to relate Agent1 with the goals pursued by Role1, the activities that involve it and the environment entities that it uses. If the automated process has been followed and the ontology has undergone no manual modifications since Step
the reasoner will detect no inconsistencies in the knowledge in this step. However, if the ontology has been manually modified, for example to change the mapping agent-role, the reasoner may detect inconsistencies that will have to be solved by the ontology engineer. The reasoner materialises the inferred knowledge to create the output ontology in this step.

**Step 7: Cardinality Violation Detection** is the final step in the automated augmentation process. It checks that the domain knowledge fulfils the cardinalities of the relationships as defined in the AOSE metamodel. This cannot be done automatically by the reasoner because of OWL 2 open world assumption. It is assumed that ontologies can be incomplete, therefore non-asserted knowledge cannot be assumed to be false, only to be unknown. In an axiom, the metamodel defines that every goal is achieved by at least one activity. If the instance Goal1 belonging to the class Goal is not achieved by any activity in the domain ontology, the reasoner does not detect this as a cardinality violation, but simply as missing knowledge: there might be an unknown activity that achieves Goal1. To solve this problem, this step looks at the object assertions defined in the ontology to ensure that the cardinalities of all object properties are satisfied. For example, the individuals Activity1 and EnvironmentEntity1 belong to the classes Activity and EnvironmentEntity respectively. If Activity1 is not related to any environment entity through the relation ActivityNeedsEnvironmentEntity, no violation will be detected because the cardinality of the object property is zero or many. However, if EnvironmentEntity1 is not related to any activity through the relation EnvironmentEntityNeededInActivity, a violation will be detected as the cardinality of the object property is one or many. The output of the step is a list of violations, indicating which individuals are violating which relationships. If this list of violations is not empty, the ontology engineer is required to solve them in the original domain ontology and reinitialise the automatic augmentation process. The actions leading to resolving the violations depend on the application domain and may require consultation with the client. The ontology augmentation process concludes with Step 7 producing an inconsistency free ontology that satisfies the axioms defined for the AOSE paradigm. This output ontology can then be discussed...
with the client to ensure that it accurately represents their problem conceptualisation. The augmented domain ontology will be the input of the ontology-models comparison module, used to detect discrepancies between artefacts.

5.4.2 Ontology-Models Comparison Module

The second module developed as part of the automatic V&V process compares each MAS model with the ontology to detect discrepancies. The inputs of the module are the augmented ontology and the models to be validated. For the purpose of this thesis, some model input functionalities of the module have been simplified to focus on the V&V process itself. Another important aspect of the module that should be highlighted is that the comparison of models and ontology should have a semantic character. For example, the concepts Club Member and Special Client could be semantically equivalent in certain domains. However, the implementation of a semantic comparison mechanism is a complex task that would require very significant research effort. For the purpose of this thesis, the proof of concept tool implements a simple comparison mechanism based on strings of characters. This means that the concepts Club Member and Special Client would be different for the module described in this section. Ontology and models need to undergo a manual revision to ensure that equivalent elements have the same name. The steps carried out by the ontology-models comparison module roughly corresponds to the tasks described in Section 5.1. Task 2.1: Ontology-Models Comparison was described as composed of two subtasks. Task 2.1.1: 1-on-1 Model-Ontology Comparison received the MAS models and the augmented domain ontology as input, and compared each model with the ontology to produce a list of discrepancies. Task 2.1.2: Models Interdependencies Detection looked at the discrepancy list to identify which models could be affected by solving the problems identified in the list. The module also performs other steps necessary to initiate and conclude these core tasks. The architecture of the module divides its functionalities into the four steps depicted in Figure 5.18. Their details are as follows:
Step 1: Working Environment Preparation is a preliminary step that sets up the environment of the ontology-models comparison module by loading the ontologies and MAS models. Whereas the load of the ontologies is a direct application of the OWL API, loading the MAS models has been designed in such a way that the module can support any AOSE methodology. Chapter 2 identified a set of concepts and relationships used by most extant AOSE methodologies. However, because of the variety of idiosyncrasies underpinning different AOSE methodologies, there is a generalised lack of consensus about the exact content of each model. For the purpose of the thesis and to avoid technical issues that could obscure the purpose of the proof of concept, the ontology-models comparison module focuses on models developed for the ROADMAP AOSE methodology. Nevertheless, to facilitate the extension of the tool for other AOSE methodologies the design of this step has followed a general approach. First, the module retrieves the development methodology that the MAS models follow. This is currently hardcoded, but it could be easily modified to be a run-time user decision. Then, the selection ontology is queried to determine which models are required by the given AOSE methodology. Instances of those models are created at run-time and populated with the concepts and relations corresponding to the input MAS models. The description and population of the source MAS models has been hardcoded to facilitate the development.

Step 2: Model-Ontology Comparison performs the actual comparison between model and ontology. This is done taking into account only matching literal string of characters (see discussion above). For each
model, different lists are generated for each type of concept (Role, Activity, ...) and for each type of relationship (RoleParticipatesInActivity, GoalIsAchievedByActivity, ...) used in the model. Equivalent lists are generated from the augmented domain ontology. Each element that is contained simultaneously in two corresponding lists (extracted from the ontology and model) is removed from both lists. When no further elements can be removed, the remaining elements (if any) of the ontology list denote elements defined in the ontology but not in the model. Conversely, remaining elements of the model lists correspond to elements defined in the models but not in the ontology.

Step 3: Dependency Analysis takes the lists generated in Step 2 as input to discover how changes provoked by solving these discrepancies would affect other models. For each element in each list for each model, the Selection Ontology is queried to identify which other models use that concept or relationship. These dependent models are compiled to produce the final report in next step.

Step 4: Report Compilation summarises the outcome of the comparison process by producing a report that specifies which elements have been defined in the ontology but not in the models, and vice versa. For each discrepancy, the report also states which models share the use of the discrepant element. For the purpose of this thesis, a simple graphic user interface (GUI) has been developed to present the analysis of the report.

The GUI (Figure 5.19) presents the information in three main areas. On the left side (marked as Number 1), the GUI presents the list of models that are required for the development methodology. The upper right side of the interface (Numbers 2 to 7) deals with the definition of elements corresponding to concepts in the AOSE paradigm (Roles, Agents, Goals, ...). Number 2 lists concepts that have been defined in the current model (the one selected in 1) but not in the ontology. Number 5 lists concepts defined in the ontology but not in the model. Numbers 3, 4, 6 and 7 deal with the inter-model dependencies associated with the discrepancies selected in 2 and 5 respectively, as follows:
Number 3: the model is considered correct, and the concept is removed from the ontology. Thus, some other models that include the type of concept that caused the discrepancy may present new discrepancies in the next iteration of the V&V process.

Number 4: the model is considered wrong, and the concept has to be removed from the model. Other models that also include the concept that caused the discrepancy have to change accordingly.

Number 6: the ontology is considered wrong, and the concept is removed from the ontology. Other models that also use this kind of concept may present new discrepancies in the next iteration.

Number 7: the ontology is considered correct, and the concept is added to the model. As a consequence, other models that also use the same type of concept should be reviewed.
### Table 5.9: Types of ontology-based operators

<table>
<thead>
<tr>
<th>Model</th>
<th>Concept Operator</th>
<th>Relationship Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal Model</td>
<td>1 and 2</td>
<td>3 and 4</td>
</tr>
<tr>
<td>Role Model</td>
<td>1 and 2</td>
<td>3</td>
</tr>
<tr>
<td>Organisation Model</td>
<td>1</td>
<td>2, 3 and 4</td>
</tr>
<tr>
<td>Environment Model</td>
<td>1 and 2</td>
<td>3</td>
</tr>
<tr>
<td>Interaction Model</td>
<td>1 and 2</td>
<td>3</td>
</tr>
<tr>
<td>Agent Model</td>
<td>1, 2, 3 and 4</td>
<td>5, 6, 7 and 8</td>
</tr>
<tr>
<td>Scenario</td>
<td>1, 2 and 3</td>
<td>4, 5, 6, 7 and 8</td>
</tr>
</tbody>
</table>

The bottom right-hand section of the interface presents an analogous structure but instead of concepts, it deals with relationships. This means that Number 8 lists the relations that have been defined in the model but not in the ontology (similar to Number 2) and Number 11 lists the relations that have been defined in the ontology but not in the model (similar to Number 5). Numbers 9, 10, 12 and 13 are equivalent to Numbers 3, 4, 6 and 7 for relationships.

In relation to the ontology-based operators refined in Section 5.2, the model chosen in Number 1 establishes which set of operators will be applied to the V&V of the corresponding model. For each model, the operators defined in Section 5.2 can be divided into operators for the detection of discrepancies in the definition of concepts and in the definition of relationships (Table 5.9). Numbers 2 and 5 are completed by the application of the concept-related operators for the selected model. Numbers 3, 4, 6 and 7 are not directly related to the operators, but depend on the discrepant concept selected in 2 or 5. Numbers 8 and 11 are completed by the application of the relationships-related operators for the selected models. Numbers 9, 10, 12 and 13 show the model dependencies depending on the discrepant relationships selected in 8 or 11.

### 5.5 Chapter Summary

In Chapter 4, the V&V process was presented and validated through a case study. That validation led to a refinement of the V&V process, so the process would allow the modification of the ontology to adapt
to changes in the requirements specification as the development proceeds. Section 5.1 presented this new refined version of the V&V process including this conceptual modification. As a consequence of the improvements to the V&V process, the comparison operators were refined in Section 5.2. Another important conclusion of the validation in Chapter 4 was that the manual application of the V&V process was overly arduous and was likely to be unviable for very large projects. To address this issue, Section 5.3 harnessed the formal semantics of ontologies to automate a significant part of the V&V process, namely, the ontology augmentation and comparison operator application. To test the viability of the automatic V&V process, a support tool was designed and introduced as a proof of concept in Section 5.4. The adequacy of the automatic V&V process and support tool will be evaluated through two case studies in the following chapter. The first case study will test the developer and domain independence of the V&V process by changing the problem and the participants of the experiments. To remove another validity threat to the design of the V&V process, the final case study will be established and conducted to test whether the automatic V&V process can be applied to models produced with a different AOSE methodology.
EVALUATION OF THE AUTOMATIC V&V PROCESS

The initial evaluation of the process and operators through a development case study in Chapter 4 allowed the process to be refined and confirmed the need for its automation. Chapter 5 introduced the mechanisms that enabled the automation of two critical activities of the ontology-based V&V process: the augmentation of the ontology and the application of the ontology-based operators. In this chapter, two new development case studies are presented to evaluate the automated V&V process. The first development deals with a new domain problem to assess the impact of the support tool on the V&V process. This case study is designed to mitigate important validity threats to the experiment: such as user independence and domain independence. In the second case study, a final evaluation is conducted to ensure that the ontology-based MAS models V&V process can be applied to different agent-oriented software engineering (AOSE) methodologies. In other words, this tests the AOSE methodology independence of the V&V process. The new AOSE methodology chosen is MOBMAS, a modern AOSE methodology proposed to address the limitations of extant AOSE methodologies.

This chapter is structured as follows: in Section 6.1, the refined version of the V&V process and the support tool are validated with a new case study; Section 6.2 discusses the outcome of the case study; Section 6.3 details minor tweaks to the ontology-based operators and associated tool to support the MOBMAS methodology; Section 6.4 describes the final evaluation for the V&V process using MOBMAS; Section 6.5 discusses the outcome of the V&V process evaluation; and the chapter concludes with Section 6.6.
6.1 Validating the Automatic V&V Process

This section describes the case study used to evaluate the refined version of the V&V process and its automatic support tool. The V&V process support tool presented in Chapter 5 was designed to be independent of the application domain. This case study confirmed that the automatic V&V process was capable of augmenting a domain ontology and using it for the V&V of a set of MAS models. The support tool required no modifications to conduct these tasks and showed that the scalability of the ontology augmentation was no longer a challenge (see discussion in Section 6.2). The ontology augmentation module automatically detected inconsistencies in the domain ontology and violations of the AOSE axioms (see examples in sections 6.1.3 and 6.1.8). These were the only aspects of the augmentation process that required human intervention. To neutralise internal validity threats the process domain independence is tested by changing the problem domain. By changing the persons responsible for the development of the ontology and MAS models, the stakeholder independence is also validated.

6.1.1 The Wine Broker Domain

The case study focused on applying the automatic V&V process to the models of a MAS to support the activities of a wine broker company. This company specialises in establishing links between wineries and their potential clients, to facilitate wine distribution. The goals of the sought MAS are to improve the wine broker clients’ satisfaction and to reduce costs. By automating many of the services provided by the wine broker, interactions between wine buyers and sellers will become faster. By also eliminating a middleman, the final price of wine will be reduced. The business model of the wine broker is described below and is depicted graphically in Figure 6.1.

Purchase wine: Clients can browse an on-line catalogue of wines and purchase those that best fit their needs. All clients can get advice from an automatic system called the Intelligent Wine Advisor that will search the catalogue to recommend appropriate products based
on the client’s preferences. Corporate clients and wine club members can also request an individualised wine consultancy with one of the wine experts associated with the wine broker. Once a client has placed an order, the system confirms that all the items requested by the client are in stock; if the items are not in stock, the system will contact the corresponding winery to order stock. The order will be delivered anywhere in the country. The shipping and handling costs will be waived for corporate clients and wine club members.

Organise cellar door visit: Any client can arrange a visit to any of the associated wineries through the wine broker company, depending on the availability of time slots at the winery. This allows the client to taste and purchase different types of wines directly at the winery. The wine broker does not organise transportation for the client to the winery.

Join wine club: Regular clients can apply for membership of the wine club. Wine club members commit to buying a certain amount of
wine regularly and, in exchange, they obtain benefits such as special prices on some products, preferential invitations to special events and waiving of delivery costs.

Organise home tasting: Corporate clients and wine club members are entitled to apply for a home tasting. A wine consultant from the wine broker will bring a prearranged set of wines to a home for a group of clients to taste and purchase.

Participation in special events: The wine broker periodically organises special events, such as wine tasting in scenic environments or a combined dinner and wine tasting. Corporate clients and wine club members are offered tickets (with reduced fares) for the special events before they are made available to the general public.

Both the ontology and MAS models developers based their work on the same specification. The ontology developers were graduate students of a course on knowledge representation technologies (focused on OWL 2). The MAS models developer was a research student working in AOSE. This provided an opportunity to evaluate whether the modelling experience of stakeholders has a major impact on the V&V process and the development of the process when feedback to fine-tune the ontology is provided regularly.

6.1.2 Task 1.1: Domain Ontology Acquisition – Iteration 1

The first task of the automated V&V process is the acquisition of an appropriate ontology. This can be a result of development from scratch or retrieval from a repository. A search in Swoogle found existing ontologies (such as the wine ontology (W3C 2003)) describing characteristics and types of wines that could be used at run-time by agents to make wine recommendations to clients. However, none of the existing ontologies model the domain of a wine broker. Thus an ontology describing the business model of the wine broker is to be developed from scratch. The wine broker ontology was developed based on the domain description detailed in Section 6.1.1. The open source ontology editor Protégé is compatible with the semantics of OWL 2 and was selected to support the ontology engineering. According to the description of the automatic mechanism in Chapter
the domain ontology is built as a set of undistinguished instances of the generic class \textit{Thing}. Five object properties are defined in the ontology to model the relations between individuals that represent various concepts in the domain: The property \textit{achieves} is used to model activities that achieve goals; \textit{needs} links activities with their required environment entities; \textit{precedes} establishes the execution order in a sequence of activities; \textit{participates} associates roles with the activities that require their participation; and finally, \textit{uses} describes the environment entities that roles require to perform their tasks. These object properties are used in conjunction with individuals representing important concepts in the domain problem to model the business model of the wine broker to comply with the provided conceptualisation (Section \ref{sec:vandv-process}). Figure \ref{fig:domain-ontology} shows an excerpt of the domain ontology where rounded rectangles represent individuals and labelled arrows represent relationships between them. The concepts \textit{Inform club member about special events}, \textit{Club member book special events}, \textit{Publish special events} and \textit{Public and client and corporate client book special events} have been duplicated to improve the readability of the figure.

Figure 6.2: Excerpt of the domain ontology for the wine broker problem

\section*{Task 1.2: Domain Ontology Augmentation – Iteration 1}

This is the first task of the ontology-based V&V process that was entirely automated. The support tool received the domain ontology produced by Task 1.1 and the AOSE metamodel described in Section \ref{sec:aose-metamodel} as input. The augmentation process was conducted automatically according to the approach described in Sections \ref{sec:ontology-augmentation} and \ref{sec:ontology-augmentation-algorithms},
and required human intervention only to solve inconsistencies in the domain ontology.

During the first inference step (Step 3 Section 5.4.1) after the object properties of the domain ontology were linked to those defined in the AOSE metamodel, the augmentation module detected that the domain ontology was inconsistent. This problem was due a mistake by the ontology engineer. The object property *needs* had been misused in several assertions. This object property was intended to link activities with their required environment entities. However, the ontology engineer had modelled *Special events manager needs Wine club member form* instead of the correct assertion *Special events manager uses Wine club member form*. This provoked the individual *Special events manager* to be inferred as belonging to classes *Role* and *Activity* simultaneously. These classes are defined as disjoint classes in the AOSE metamodel, therefore an inconsistency is triggered. Once the faulty assertions were manually amended, the augmentation module reinitialised and finished its processing without further human intervention. The process lasted 15 minutes and raised the 222 object properties assertions of the original ontology to 995 in the augmented ontology. The concepts defined in the original domain ontology were automatically classified in their appropriate classes as defined in the AOSE metamodel. Figure 6.3 illustrates how the individuals defined in the excerpt domain ontology of Figure 6.2 are classified. For each role, in this case *Special events manager*, *Club member* and *Non member*, an agent is generated.

Figure 6.3: Automatic classification of wine broker concepts
Figure 6.4 shows the new relations inferred in relation to the instances of concepts Goal, Activity and EnvironmentEntity: GoalAchievedByActivity, ActivityAchievesGoal, ActivityPrecedesActivity, ActivityFollowsActivity, ActivityNeedsEnvironmentEntity, EnvironmentEntityNeededInActivity, GoalNeedsEnvironmentEntity and EnvironmentEntityNeededForGoal. The goal Organise special event has been duplicated to improve the readability of the figure.

Figure 6.4: Augmented wine broker relations involving goals, activities and environment entities

Finally, Figure 6.6 shows the new relations inferred with regards to the instances of concepts Goal, Activity, Role and Agent: RolePursues-
Goal, GoalPursuedByRole, AgentPursuesGoal, GoalPursuedByAgent, RoleParticipatesInActivity, ActivityInvolvesRole, AgentParticipatesInActivity and ActivityInvolvesAgent. The goal Organise special event has been duplicated to improve the readability of the figure.

Figure 6.6: Augmented wine broker relations involving goals, activities, roles and agents

The knowledge modelled in the domain ontology complied with the axioms defined in the AOSE metamodel. The augmentation module detected no cardinality violations and the augmentation process concluded.

6.1.4 Task 1.3: Domain Ontology Validation

In this task the augmented domain ontology is validated with the client to fine tune it according to the client conceptualisation. The development case study described in Chapter 4 showed that this is an important task as the domain ontology underwent some adjustments to fully comply with the client initial problem conceptualisation. However, given that one of the important goals of this evaluation is to assess the ontology feedback mechanisms and the impact of inexperienced ontology engineers and MAS modellers in the V&V process, this task will be omitted for this case study.

6.1.5 Task 2.1: Ontology-Models Comparison – Iteration 1

The automatic tool applied the ontology-based comparison operators to the available models – goal, role, organisation and environment
models – automatically. The discrepancy report was instantaneously generated without human intervention.

The concepts and relationships used in the role model are a subset of the ones defined by the goal model (i.e. roles, goals and relations between them) and the definition of roles is also used in organisation and environment models, hence, the report generated for all models partially overlapped. The structure of the goal hierarchy was simpler in the ontology than in the goal model. The MAS models developer included many goals that had been considered activities in the ontology, for example, Premium ticket sale or Regular ticket sale as sub goals of the goal Organise special event. Other goals, such as Select wine, Make payment or Delivery as sub goals of the goal Sell wine online, had been only defined in the model. The ontology included two roles, Wine club manager and Special events manager that were not included in any model. The roles Club member and Special client in the ontology corresponded to only one in the models. The ontology defined many environment entities that represented forms containing information necessary for the wine broker activities (e.g. Wine club member form or Special event information form). The environment model did not include most of those environment entities. However, it defined the environment entity Courier to represent a wine delivery service.

In terms of relationships, the organisation model differed greatly from the ontology because no authority relations (RoleControlsRole and RoleIsPeerRole) had been modelled by the ontology engineer. The rest of the discrepancies involving the relations RolePursuesGoal and RoleUsesEnvironmentEntity between models and ontology were due to missing roles, goals or environment entities in artefacts (model or ontology) as described above.

6.1.6 Task 2.2: Discrepancy Removal – Iteration 1

This task involved examining each discrepancy to decide whether it would be solved by changing the ontology or changing the model.

It was determined that it was necessary to simplify the definition of goals in the model. Concepts such as Login or Process payment should not be considered as goals but as necessary activities to achieve the
goal of *Sell wine online* and, therefore, should be removed from the goal and role models. However, the definition in the ontology of that very same goal, *Sell wine online*, had to be extended in other aspects by adding as sub goals (defined in the models) *Select wine, Make payment* or *Delivery*. The roles *Wine club manager* and *Special events manager*, which were only defined in the ontology, were relevant and added to the model. However, the functionality associated with the ontology roles *Club member* and *Special client* overlapped and so were merged to comply with the model. The definition of environment entities in the ontology was conducted at a level that was too low, as the pieces of information to be input and exchanged in activities were considered a practical decision of the system designers and out of the scope of the domain analysis done in the ontology. All the information forms defined in the ontology and in the models were removed from both artefacts. The discrepancy between environment model and ontology with regards to the environment entity *Courier* being defined only in the model was solved by adding it to the ontology. Furthermore, this discrepancy inspired the addition of a new environment entity to both the ontology and environment model, *Bank*, to represent the financial services needed for the online shopping service.

The discrepancies detected between the ontology and models with regards to the relations *RolePursuesGoal* and *RoleUsesEnvironmentEntity* were solved according to the decision made for the missing concepts. In other words, if the relationship *A R B* was due to the absence of concept *A* in the ontology, and it was decided to add *A* to the ontology then *A R B* was also added to the ontology, whereas if *A* was removed from the model then *A R B* would also be removed from the model. The discrepancies involving the authority relations between roles originated a discussion. Although none of them had been defined in the ontology, they were not blindly added. It was decided that there was no real authority between roles in the wine broker, therefore the relation *RoleControlsRole* (for example between the *Cellar door manager* and the *Sale staff*) should not be used in the organisation model and thus not added to the ontology.
6.1 Validation of the Automatic V&V Process

Listing 6.1: Cardinality violations for concept Bank

<table>
<thead>
<tr>
<th>Description</th>
<th>Axiom</th>
<th>Participation Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>The EnvironmentEntity Bank violates the cardinality of the axiom:</td>
<td>EnvironmentEntity EnvironmentEntityNeededInActivity min 1 Activity</td>
<td>Currently, 0 relations</td>
</tr>
<tr>
<td>EnvironmentEntity Bank participates in 0 relations using the object property</td>
<td>EnvironmentEntityNeededInActivity.</td>
<td></td>
</tr>
<tr>
<td>The EnvironmentEntity Bank violates the cardinality of the axiom:</td>
<td>EnvironmentEntity EnvironmentEntityNeededForGoal min 1 Goal</td>
<td>Currently, 0 relations</td>
</tr>
<tr>
<td>EnvironmentEntity Bank participates in 0 relations using the object property</td>
<td>EnvironmentEntityNeededForGoal.</td>
<td></td>
</tr>
</tbody>
</table>

6.1.7 Task 1.1: Domain Ontology Acquisition – Iteration 2

The original domain ontology was modified in accordance with the decisions made in Task 2.2 during the first iteration. It was decided to modify the original domain ontology instead of the augmented ontology. There were many more relationships in the augmented ontology than in the original ontology, therefore changes were more complex to apply uniformly. In contrast, the augmentation module made it effortless to adequately propagate changes from the original ontology to a new version of the augmented ontology.

6.1.8 Task 1.2: Domain Ontology Augmentation – Iteration 2

The augmentation module processed the domain ontology without detecting inconsistencies. However, the module found 16 cardinality violations relating to some axioms. This was due to an over-simplistic addition of concepts to the ontology as a result of the decision making during first iteration. For example, during Task 2.2 – Iteration 1, it was decided that the environment entity Bank had to be added to the ontology. The ontology engineer introduced the concept Bank in the ontology and linked it to the corresponding roles, Sale staff, Club member, Non member and Winery sale staff, through the relation RoleUsesEnvironmentEntity. However, the environment entity was not modelled as related to any activity or goal through the relations EnvironmentEntityNeededInActivity and EnvironmentEntityNeededForGoal, respectively. This triggered the cardinality violations warnings depicted by Listing 6.1.
The ontology engineer modified the domain ontology to address these issues and used the amended domain ontology as input for the augmentation module. This time, the module finished processing the ontology without detecting further cardinality violations. The number of object property assertions (relationships) rose from 206 to 1122.

6.1.9 Task 2.1: Ontology-Models Comparison – Iteration 2

Once again, applying the ontology-based operators was automatically carried out by the comparison module to produce the discrepancy report. The MAS model developer provided an improved and extended set of models. Goal, role, organisation and environment models were refined from the first iteration. Additionally, the scenario model was supplied for the first time for the V&V process.

The comparison of goal, role, organisation and environment entity models with the ontology did not show discrepancies with the definition of goals, roles and environment. However, three relationships between roles and their pursued goals were detected as being defined in the models but not in the ontology. This discrepancy was caused by the addition of concepts to the ontology at the beginning of the iteration. For instance, in the case of the relationship Club member RolePursuesGoal Premium ticket sale, the ontology engineer added to the ontology the new goal Premium ticket sale, and the relationships Premium ticket sale GoalIsSubGoalOfGoal Organise special event, and Special events manager RolePursuesGoal Premium ticket sale. The ontology augmentation module did not detect the goal Premium ticket sale as a cardinality violation because the role Special events manager pursued the goal. Several other relationships appeared in the ontology but not in the goal model. These were concerned with the hierarchical relation of goals and the relation with roles. For example, as Club member pursues the goal Sell Wine Online and this is a sub goal of the general goal Wine Broker, the relationship Club member RolePursuesGoal Wine broker was automatically inferred by the augmentation process. Some discrepancies between the ontology and the organisation mode were detected with regard to the definition of
the authority relations between roles. Two of them were only defined in the model, and related the role Wine consultant with the roles Non member and the Sale staff. Others were defined only in the ontology, for example Cellar door manager RoleIsPeerRole Club member and Cellar door manager RoleIsPeerRole Non member. Only one discrepancy, Sale staff RoleUsesEnvironmentEntity Winery, was detected between the ontology and the environment model.

V&V had been completed on the instances of the goal, role and environment entity concepts in the previous iteration and thus the application of operators during Iteration 2 detected no further discrepancies in terms of those concepts. However, as activities were first introduced during this second iteration, a number of discrepancies in the definition of activities between scenario and ontology were detected. Examples of activities that had been defined in the model but not in the ontology were Make payment, Check payment, Receive application and Receive query. A significant number of activities was defined only in the ontology, for instance the activities Add selected wine to purchase cart or Club member books for special event.

Similarly to what happened in the first iteration, there were discrepancies in the relationships between the ontology and scenario because the activities involved had not been defined in either the ontology or the scenario. This was, for instance, the case of the relationships Non member RoleParticipatesInActivity Make payment and Non member RoleParticipatesInActivity Add selected wine to purchase cart, which were only defined in the scenario and ontology respectively, because the activities Add selected wine to purchase cart and Makes payment had not been defined in the scenario and ontology respectively.

6.1.10 Task 2.2: Discrepancy Removal – Iteration 2

The three discrepancies that were detected between the ontology on one hand, and the goal and role models on the other hand, were caused by new concepts not being properly used in the ontology (exemplified earlier by Club member RolePursuesGoal Premium ticket sale). These discrepancies were solved by modifying the ontology. The
relationships involving roles and goals in various levels of the goal hierarchy that were only defined in the ontology were deemed correct, but unnecessary for the purpose of the sought system. As these relationships were automatically generated by the augmentation module, it was decided that instead of removing them from the ontology (they would be automatically regenerated by the augmentation process) it was simpler to ignore them.

With regard to the discrepancies between the organisation model and ontology, the relationships defined only in the model were considered wrong and were therefore removed from the model. Two of the relationships defined only in the ontology, Cellar door manager RoleIsPeerRole Club member and Cellar door manager RoleIsPeerRole Non member were deemed correct and added to the model. The relationship RoleIsPeerRole is symmetric, i.e. if the relationship Wine consultant RoleIsPeerRole Club member is defined in the organisation model, then the relationship Club member RoleIsPeerRole Wine consultant must also be defined. The MAS modeller had only included one pair of each couple for each relationship, therefore it was accepted that the rest be added to the model. Other relationships that were not defined in the organisation model were caused by the definition of the object property RoleIsPeerRole as transitive in the AOSE metamodel. This provoked certain relationships, such as Non Member RoleIsPeerRole Wine Consultant, to be automatically added to the ontology by the augmentation module. While it is true that there is no authority relation between those two roles, it is also true that they do not interact with each other. It is, therefore, unnecessary to add the relationship to the model. The definition of the object property RoleIsPeerRole as a transitive property in the AOSE metamodel should be further discussed. The relationship Sale staff RoleUsesEnvironmentEntity Winery was defined in the environment model but not in the ontology; it was deemed correct and added to the ontology.

Of the activities defined only in the scenario, it was decided that only Make Payment was to be added to the ontology. The activities Check Payment, Receive Application and Receive Query, which were defined only in the scenario, were considered to be decisions to be made at design level and it was therefore necessary to remove them. Due to the lack of expertise of the ontology engineer, the ontology
included many activities that were also deemed to be design time
decisions. Only the ontology activities that represented the booking of
club members and non-members for special events were deemed
relevant and added to the model.

Decisions about discrepancies in the definition of participation of roles in activities were made in accordance with the decision of removing or adding the corresponding concept. For example, the relationship Non member RoleParticipatesInActivity Make payment was only defined in the scenario. As the activity Make payment was accepted for addition to the ontology, this relationship had to be added as well. Similarly, as the activity Add selected wine to purchase cart had to be removed from the ontology, the relationship Non member RoleParticipatesInActivity Add selected wine to purchase cart, which was only defined in the ontology, had to be removed as well. Examples of discrepancies that were solved by modifying the scenario because of the ontology were the participation relationships associating Member and Special events manager to Presale buy, Non member and Special events manager to Regular buy, Wine consultant to Contact for consultancy, and Sale staff to Make payment.

6.1.11 Task 1.1: Domain Ontology Acquisition – Iteration 3

This task revisited the domain ontology to materialise the decisions made in Task 2.2 during the second iteration. The definition of goals, roles and environment entities had been mostly stabilised in the previous iteration. This time, most changes were concerned with the removal of a large number of low level activities and their associated relationships.

6.1.12 Task 1.2: Domain Ontology Augmentation – Iteration 3

In this task, the modified domain ontology was used as input for the augmentation module. Although the ontology was consistent, the augmentation module uncovered three cardinality violations relating to the goal Organise home tasting (shown in Listing 6.2).
Similarly to what happened during the second iteration, the changes in the ontology were not properly propagated. When the unnecessary activities were removed, the goal Organise home tasting was disconnected from roles, agents and activities. The ontology engineer introduced new object property assertions in the ontology to associate the goal with appropriate roles and activities. The fixed domain ontology was used as input to the augmentation process, which concluded without detecting any other inconsistencies or cardinality violations. The number of object property assertions rose from 150 to 864. There were fewer object property assertions (c.f. Section 6.1.8) because of the removal of a significant number of irrelevant activities.

6.1.13  Task 2.1: Ontology-Models Comparison – Iteration 3

As no new discrepancies were detected in relation to the organisation and environment model, these models were considered validated. However, the modification of the definition of activities in the ontology provoked the appearance of new relationship discrepancies with the goal and role models. An example of these new discrepancies is Cellar door manager RolePursuesGoal Organise home tasting.

The scenario was also affected by the changes that the ontology underwent in the previous iteration. The application of the operators
uncovered new discrepancies in the definition of the relationships. The relationships Club member RoleParticipatesInActivity Make order and Apply ActivityPrecedesActivity Arrange date were defined in the scenario but not in the ontology. The relationships Club member RoleParticipatesInActivity Offer presale special event tickets, Non member RoleParticipatesInActivity Offer regular sale special event tickets, Special events manager RoleParticipatesInActivity Regular special event booking and Special events manager RoleParticipatesInActivity Presale special event booking were only defined in the ontology.

This iteration was the first time that the developer provided the agent model for V&V. This model presented a significant degree of overlap with the scenario, which underwent major changes during the second iteration. As the developer produced this model after addressing the issues detected for the scenario, most of the discrepancies that would have naturally occurred were prevented. The discrepancies detected for this model were the same as those discussed above for the scenario.

The interaction model was also newly provided by the developer in this iteration. Similarly to the agent model, its main components – roles, activities and relations between them – had been analysed within the context of the scenario in the previous iteration. Major discrepancies had been prevented and only the few differences described for the scenario were detected.

6.1.14 Task 2.2: Discrepancy Removal – Iteration 3

The new discrepancies detected with the goal and role model were a consequence of the modification of the activities and the definition of the AOSE metamodel. The metamodel defines that if a given role participates in a certain activity, then it can be inferred that the role pursues the goals achieved by that activity. Some of the activities defined in the scenario achieved several goals. For example, the generic activity Apply was used as part of the scenarios of applying for wine club membership and for the organisation of home wine tastings. Different roles participate in the different instances of the activity Apply, for example, the Wine club manager and the Cellar door
manager. The ontology reasoner inferred that all roles participating in the activity Apply pursued the goals that were achieved by the activity, in particular the Wine club manager RolePursuesGoal Organise home tasting. These discrepancies were, therefore, false positives in the comparison process that would not have occurred if the activities had been named differently. However, it could be argued that these discrepancies are caused by the literal comparison mechanism implemented by the comparison module and that a semantic comparison approach would overcome this issue. As the goal and role models presented no other discrepancies, they were considered validated.

The discrepancies detected for scenario, agent and interaction models were common (except for the relationship Apply ActivityPrecedesActivity Arrange date that was only involved with scenario and agent model) and were solved by modifying the models and the ontology depending on the case. The discrepancies involving the relationships Club member RoleParticipatesInActivity Make order and Apply ActivityPrecedesActivity Arrange date were solved by adding these assertions to the domain ontology. To solve the discrepancies involving the relationships Club member RoleParticipatesInActivity Of-fer presale special event tickets, Non member RoleParticipatesInActivity Offer regular sale special event tickets, Special events manager RoleParticipatesInActivity Regular special event booking and Special events manager RoleParticipatesInActivity Presale special event booking, scenario, agent and interaction models were modified to incorporate them.

6.1.15 Task 1.1: Domain Ontology Acquisition – Iteration 4

In this task the domain ontology was modified to include the new knowledge agreed upon in Task 2.2 during the third iteration. The ontology was very stable at this point; no concepts were added or removed and only two object property assertions were added.

6.1.16 Task 1.2: Domain Ontology Augmentation – Iteration 4

The augmentation module processed the domain ontology without detecting any inconsistencies or cardinality violations. As the onto-
logy changed minimally, the number of object property assertions was similar to the third iteration and rose from 152 to 870.

6.1.17 Task 2.1: Ontology-Models Comparison – Iteration 4

The application of operators showed no new discrepancies in any model. There was no need to modify either the ontology or the models as they had converged. All models were considered validated and the case study concluded.

6.2 Conclusions of the Wine Broker Case Study with Roadmap

The main goal of this case study was to assess the performance of the automatic V&V tool compared to the manual application of the V&V process as described in the case study in Chapter 4. The case study spanned four iterations. In the first iteration, only early models were processed. Progressively, more models were added until, in the third iteration, the set was complete. Some models, namely goal, role, organisation and environment model, concluded their validation in the third iteration. Other models that were incorporated in the process in later iterations were only considered validated in the last iteration of the case study.

To reduce the impact of validity threats on the experiment, the parameters of the development case study were set to differ as much as possible from the evaluation presented in Chapter 4. The ontology engineer and MAS modeller were chosen to be different from the ones involved in the first case study and neither of them had previous experience in developing ontologies or building MASs. This was intended to determine whether the V&V process was dependent on the developer choice and to what extent the quality of the starting artefacts could affect the outcome of the V&V process. To maximise the effect of this factor, initial validation of the ontology by the client was omitted. As a result, the V&V process commenced with a domain ontology that was relatively different to the client’s expectations. Despite the need for significant changes to both the ontology and the models, they finally converged in a consistent set
of models that fulfilled the client conceptualisation. Full convergence was achieved in four iterations, which is the same number of iterations as in Chapter 4 where the domain ontology was initially validated by the client. The completion of the models V&V in four iterations even without an initial ontology validation suggests that the feedback to adjust the domain ontology is a determining factor in achieving fast convergence. The case study presented in Section 6.4 will test whether higher quality initial artefacts can speed up the V&V process.

A different problem domain from the one used in the case study described in Chapter 4 was chosen to test domain independency of the V&V process. Both problems are typical examples of MAS developments. The literature survey in Chapter 2 discussed that typical areas of application of agent technologies is the development of simulators, e.g. the aircraft turnaround problem, and the development of complex systems that require agents with negotiation capabilities, e.g. the wine broker problem. The V&V process was conducted in a similar way, regardless of the problem domain. One factor that most definitely affected the execution of the case study was the use of the support tool.

In terms of efficiency, the application of the tool significantly reduced the processing time when compared with the manual approach adopted for the case study conducted in Chapter 4. Table 6.1 shows the execution times for the V&V tasks. No times have been detailed for the automated tasks as they required no human resources, or for the modification of the MAS models as they were considered part of the traditional AOSE process. Interpretation of these times should take into account that neither the ontology engineer nor the modeller had previous experience with their tasks and consequently the quality of the first versions of the produced artefacts suffered. Chapter 4 foreshadowed that by automating the ontology augmentation and operators application, the V&V time could be reduced by nearly half. This has been confirmed in this case study, as the time invested in the development was 14 hours; the development in Chapter 4, which was a similar size, took 26 hours.

The use of the fully automated ontology augmentation module to add AOSE structure to the domain ontology eliminates the burden of
Table 6.1: Estimated times for the second case study  

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Task Description</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Iteration 1</strong></td>
<td>T1.1: Ontology Acquisition</td>
<td>4 hrs.</td>
</tr>
<tr>
<td></td>
<td>T1.2: Ontology Augmentation</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>T2.1: Ontology-Models Comparison</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>T2.2: Discrepancy Removal</td>
<td>2 hrs.</td>
</tr>
<tr>
<td><strong>Iteration 2</strong></td>
<td>T1.1: Ontology Acquisition (Modification)</td>
<td>2 hrs.</td>
</tr>
<tr>
<td></td>
<td>T1.2: Ontology Augmentation</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>T2.1: Ontology-Models Comparison</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>T2.2: Discrepancy Removal</td>
<td>2 hrs.</td>
</tr>
<tr>
<td><strong>Iteration 3</strong></td>
<td>T1.1: Ontology Acquisition (Modification)</td>
<td>2 hrs.</td>
</tr>
<tr>
<td></td>
<td>T1.2: Ontology Augmentation</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>T2.1: Ontology-Models Comparison</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>T2.2: Discrepancy Removal</td>
<td>2 hrs.</td>
</tr>
<tr>
<td><strong>Iteration 4</strong></td>
<td>T1.1: Ontology Acquisition (Modification)</td>
<td>5 min.</td>
</tr>
<tr>
<td></td>
<td>T1.2: Ontology Augmentation</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>T2.1: Ontology-Models Comparison</td>
<td>–</td>
</tr>
</tbody>
</table>

**Total**                                                                 | 14 hrs.|

a costly manual augmentation. This is especially relevant because the refined version of the V&V process allows the ontology to change in each iteration to adapt to changes in the requirements specification. The manual modification of the domain ontology would have to take into account the inferred knowledge, as a number of relationships may be affected. The effort of adequately propagating the changes should not be neglected, as, for instance, in the third iteration of the case study the automatic augmentation mechanism raised the number of relations from 150 to 864. With the automatic augmentation mechanism, only the base domain ontology has to be modified to introduce the new requirements. Then, the ontology is again automatically augmented from scratch, adequately propagating the new changes. By automatically regenerating the full augmented ontology in each iteration, the manual modifications are kept to a minimum. The drastic augmentation in the number of asserted relationships (on average five times the initial figure) provides a rich benchmark to suggest improvements to the MAS models. It is expected that in the V&V of MAS models for critical applications, many of these addi-
tional relationships would be incorporated to the models to remove all traces of ambiguity and achieve a high quality final system.

An important benefit of the augmentation tool is the consistency and cardinality violation checking mechanisms. For large or complex domains, it is arduous to manually ensure the consistency of the ontology and that every axiom is fulfilled. Indeed, even for a medium size domain problem such as the one explored in the case study (with about 150 relationships), iterations 1 to 3 showed how inconsistencies and cardinality violations were inadvertently introduced in the domain ontology due to cognitive limitations of the ontology engineer when it was manually modified.

The application of the operators greatly benefits from the comparison module. The tool instantaneously produced the discrepancy report ready for the stakeholders to analyse it and make decisions based on it. The intervention of the stakeholders in the V&V process using the tool is reduced to the decision making process based on the report and the modification of the ontology and models accordingly.

The case study reinforced the idea of the overlap between models. For example, in the third iteration, the discrepancies detected for scenario, agent model and interaction model were essentially the same. This high degree of coupling was highlighted by the discrepancy report. When a relation linking a role with an activity was selected for any of the three models, the other two models were suggested as possibly dependent. The dependencies between models were harnessed to avoid compound errors. When the scenario was first analysed in the second iteration, a large amount of discrepancies involving the definition of activities was detected. The MAS developer amended the scenario according to the decisions made during Task 2.2 and then developed agent and interaction models from scratch, avoiding modelling the incorrect activities.

The analysis of the discrepancy report had another interesting benefit. The discussion of certain discrepancies helped in uncovering relatively unrelated requirements. An example of this can be found in the second iteration. The ontology and model differed in that only the model included the concept of the Courier to represent an environment entity in charge of delivering a certain product. Not only was it decided to add it to the ontology, but by analogy
it was decided to add to a similar environment entity, the Bank, to both the ontology and model, to represent the provider of the financial services necessary for the system. Similarly, in the first iteration, the V&V process drew the attention of the developer to the authority relations between roles as they were discrepant with the ontology (they had not been modelled by the ontology engineer). The discussion that followed determined that there should be no control relationships between roles, so the organisation model and ontology could be adjusted accordingly.

6.3 Adjusting the ontology-based comparison operators for MOBMAS

In Chapter 4, a set of MAS models was chosen based on their popularity. However, this set of models is slightly different than the ones defined by MOBMAS. To facilitate the automatic application of the ontology-based operators in the third case study, an adapted version of the operators for task, role, organisational, resource, agent and goal models for MOBMAS is presented in this section. The technical modifications implemented for the ontology-based operators application module to support the adapted version of the ontology-based validation operators are then described.

6.3.1 Task Model and its Ontology-based Validation Operators

A task is a functionality or set of functionalities that the sought system should provide (Tran and Low 2008). Any particular task can be split into smaller sub-tasks. In some cases, all sub-tasks must be executed to consider that the super-task has also been executed (AND-decomposition). In other occasions, the execution of the super-task is achieved by the execution of any of the sub-tasks (OR-decomposition). For the purpose of the V&V process, the term Task in MOBMAS is equivalent to the term Activity as described in Chapter 4; both terms will be used interchangeably here. The Task Model captures the specification of the system tasks along with their
hierarchical decomposition. Figure 6.7 shows an example of a task model. The operators of the task model for MOBMAS are:

![Task Model Example](image)

Figure 6.7: A task model example using the MOBMAS notation

1. Every Activity included in the model has been defined in the ontology and vice versa; and

2. Every relation Activity is sub activity of Activity included in the model has been defined in the ontology and vice versa.

6.3.2 Role Model and its Ontology-based Validation Operators

Roles refer to a position within the MAS organisation (Tran and Low 2008). Each role is characterised by its tasks, which are the duties that the role must fulfil. Roles serve as the building blocks for defining agent classes, and describe the expected behaviour of the agent. The Role Model defines each role in the MAS organisation (its name and its associated tasks). It also defines the acquaintances with other roles and the authority relations with them, i.e. peer to peer relation or control relation. Figure 6.8 depicts an example of a role model. The operators for the role model for MOBMAS are:

1. Every Role included in the model has been defined in the ontology and vice versa.

2. Every Activity included in the model has been defined in the ontology and vice versa.

3. Every relation Role participates in Activity included in the model has been defined in the ontology and vice versa.
4. Every relation *Role is peer of Role* included in the model has been defined in the ontology and vice versa.

5. Every relation *Role controls Role* included in the model has been defined in the ontology and vice versa.

### 6.3.3 Organisational Model and its Ontology-based Validation Operators

The Organisational Model reflects the positions, individuals or departments that exist in the organisational context and the interaction channels between them (*Tran and Low 2008*). Two types of relations can be established between positions: *acquaintance*, if two organisational units interact with each other; and *membership*, if an organisational unit is part of another. For the purpose of the V&V process, it is assumed that organisational units or positions refer to roles or aggregated roles. Figure 6.9 depicts an example of an organisational model. The operators for the organisational model in MOBMAS are:

1. Every *Role* included in the model has been defined in the ontology and vice versa.

2. Every relation *Role is sub role of Role* included in the model has been defined in the ontology and vice versa.
Resource Model and its Ontology-based Validation Operators

A resource is a non-agent entity that provides application-specific information or services to agents in the MAS (Tran and Low 2008). They do not belong internally to the system and are available to agents in other systems. Resources and agents typically co-habit the same environment. The Resource Model includes the specification of the resources in the MAS, including their name and type. It also relates resources with the agents that wrap around them. For the purpose of the V&V process, the term Resource in MOBMAS is equivalent to the term Environment Entity as described in Chapter 4 and both terms will be used interchangeably. Figure 6.10 depicts an example of a resource model. The operators for the resource model for MOBMAS are:

1. Every Agent included in the model has been defined in the ontology and vice versa.

2. Every Role included in the model has been defined in the ontology and vice versa.

3. Every Environment Entity included in the model has been defined in the ontology and vice versa.

4. Every relation Agent uses Environment Entity included in the model has been defined in the ontology and vice versa.
5. Every relation *Agent plays Role* included in the model has been defined in the ontology and vice versa.

6.3.5 *Goal Model and its Ontology-based Validation Operators*

A goal is a state of the world that an agent wants to achieve, and is used to justify the existence of the agent (Tran and Low 2008). Goals are derived from tasks and can be similarly subdivided into sub-goals. If a goal is AND-decomposed into sub-goals, all the sub-goals must be achieved to consider the super-goal achieved. If a goal is OR-decomposed into sub-goals, then the achievement of only one of the sub-goals also achieves the super-goal. The *Goal Model* is associated with a particular agent. It shows the structure of goals pursued by the agent and the possible existing conflicts between goals. *Figure 6.11* depicts an example of a goal model. The operators for the goal model in MOBMAS are:

![Goal Model Diagram](image)

*Figure 6.11: A goal model example using the MOBMAS notation*

1. Every *Goal* included in the model has been defined in the ontology and vice versa.

2. Every *Agent* included in the model has been defined in the ontology and vice versa.

3. Every relation *Agent pursues Goal* included in the model has been defined in the ontology and vice versa.

4. Every relation *Goal is sub goal of Goal* included in the model has been defined in the ontology and vice versa.
5. Every relation *Goal conflicts with Goal* included in the model has been defined in the ontology and vice versa.

### 6.3.6 Agent Model and its Ontology-based Validation Operators

Agent classes describe the characteristics of agents. Each agent can play one or more roles, encompassing their associated functionalities. At run-time, agents can dynamically switch among their roles, demonstrating a dynamic behaviour and moving between different positions in the MAS organisation (Tran and Low 2008). The Agent Model describes the agents that compose the sought system. It defines the specification of each agent, including its name, its role(s), the goals that it pursues and the events that trigger its reactions. It also describes the acquaintances between agents and the resources used by each agent. Figure 6.12 depicts an example of an agent model. The operators for the agent model in MOBMAS are:

<table>
<thead>
<tr>
<th>agent class</th>
<th>agent class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crew / Crew Role</td>
<td>Airport Ground Staff / Airport Ground Staff Role</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>agent-goals</th>
<th>agent-goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1 Aircraft Turnaround</td>
<td>G1 Aircraft Turnaround</td>
</tr>
<tr>
<td>G2 Passenger Handling</td>
<td>G2 Arrival Preparation</td>
</tr>
<tr>
<td></td>
<td>G3 Baggage Handling</td>
</tr>
<tr>
<td></td>
<td>G4 Departure Preparation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>events</th>
<th>events</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1 Airbridge has been set</td>
<td>E1 Aircraft is landing</td>
</tr>
<tr>
<td>E2 Cleaning service has finished</td>
<td>E2 Airbridge has been set</td>
</tr>
<tr>
<td>E3 Catering service has finished</td>
<td>E3 Routine maintenance has concluded</td>
</tr>
<tr>
<td></td>
<td>E4 Non-routine maintenance has concluded</td>
</tr>
<tr>
<td></td>
<td>E5 Passengers have embarked</td>
</tr>
<tr>
<td></td>
<td>E6 Aircraft has been refuelled</td>
</tr>
<tr>
<td></td>
<td>E7 Baggage has been loaded</td>
</tr>
</tbody>
</table>

Figure 6.12: An agent model example using the MOBMAS notation

1. Every *Agent* included in the model has been defined in the ontology and vice versa.

2. Every *Role* included in the model has been defined in the ontology and vice versa.

3. Every *Goal* included in the model has been defined in the ontology and vice versa.
4. Every *Environment Entity* included in the model has been defined in the ontology and vice versa.

5. Every *Event* included in the model has been defined in the ontology and vice versa.

6. Every relation *Agent plays Role* included in the model has been defined in the ontology and vice versa.

7. Every relation *Agent pursues Goal* included in the model has been defined in the ontology and vice versa.

8. Every relation *Agent uses Environment Entity* included in the model has been defined in the ontology and vice versa.

9. Every relation *Agent reacts to Event* included in the model has been defined in the ontology and vice versa.

6.3.7 *Adjusting the Ontology-based Operators Application Mechanism*

The operators application mechanism is based on the selection ontology described in Chapter 5. This ontology defines the elements (concepts and relations) that should be validated for each MAS model. The same theoretical principle applies in this case: it must describe the elements defined in each MOBMAS model. For the purpose of the proof of concept represented by the operators application mechanism, the simplest possible approach has been adopted for the extension of the ontology selection.

The existing structure of the selection ontology has been extended with a class representing the software development methodologies supported by the ontology (shown by Figure 6.13). New individuals corresponding to the MOBMAS development methodology, their models and their required validation elements (as per operators definition in Section 6.3.1 to 6.3.6) are added.

Although the set of models is rooted in the literature survey, to simplify the notation in the selection ontology, the two alternative sets of MAS models have been named after the AOSE methodology used in the case studies presented in this thesis, namely, ROADMAP and MOBMAS. Most concepts and relationships are used by both
methodologies. However they are partitioned differently into models. In some cases, the elements defined by the same model in both methodologies are different. For instance, the role model in ROADMAP includes the concepts Goal and Role, and the relationship RolePursuesGoal. The role model in MOBMAS utilises the concepts Activity and Role, and the relationships RoleControlsRole, RoleIsPeerRole and RoleParticipatesInActivity (as depicted in Figure 6.13). Due to this, and to avoid homonym issues, all individuals representing types of MAS models in the selection ontology have been renamed (compare ROADMAP_RoleModel versus MOBMAS_RoleModel). Table 6.2 summarises the MOBMAS models added to the ontology, along with their associated concepts and relations.

The implementation of the ontology-models comparison module required minor changes to overcome the technical issues posed by
the extension of the ontology. There are two significant and relevant differences from the initial version of the tool introduced in Chapter 5:

- Two possible AOSE methodologies, ROADMAP and MOBMAS, can be specified for the set of MAS models to be validated and verified. Depending on this input, the selection ontology will be accessed to retrieve the list of models that should be processed and the concepts and relations that each of them defines.

- As the application of the operators is dependent on the model being validated and verified, new Java classes are implemented to design the application of the operators for the MOBMAS models.

Similarly as for the extension of the selection ontology, as the tool is a proof of concept, the implementation was driven by the development case study. Achieving a smart or efficient design was not the focus; the key outcome was but to ensure that the ontology-based operators could be automatically applied.

<table>
<thead>
<tr>
<th>Model</th>
<th>Concept</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal Model</td>
<td>Goal</td>
<td>AgentPursuesGoal</td>
</tr>
<tr>
<td></td>
<td>Agent</td>
<td>GoalIsSubGoalOfGoal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GoalConflictsWithGoal</td>
</tr>
<tr>
<td>Role Model</td>
<td>Role</td>
<td>RoleParticipatesInActivity</td>
</tr>
<tr>
<td></td>
<td>Activity</td>
<td>RoleControlsRole</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RoleIsPeerRole</td>
</tr>
<tr>
<td>Organisational Model</td>
<td>Role</td>
<td>RoleIsSubRoleOfRole</td>
</tr>
<tr>
<td>Resource Model</td>
<td>Agent</td>
<td>AgentUsesEnvironmentEntity</td>
</tr>
<tr>
<td></td>
<td>Environment Entity</td>
<td></td>
</tr>
<tr>
<td>Task Model</td>
<td>Activity</td>
<td>ActivityIsSubActivityOfActivity</td>
</tr>
<tr>
<td>Agent Model</td>
<td>Agent</td>
<td>AgentPlaysRole</td>
</tr>
<tr>
<td></td>
<td>Role</td>
<td>AgentReactsToEvent</td>
</tr>
<tr>
<td></td>
<td>Goal</td>
<td>AgentUsesEnvironmentEntity</td>
</tr>
<tr>
<td></td>
<td>Environment Entity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Event</td>
<td></td>
</tr>
</tbody>
</table>
6.4 EVALUATING THE METHODOLOGY INDEPENDENCE OF THE V&V PROCESS: AIRCRAFT TURNAROUND SIMULATOR WITH MOBMAS

The first case study, presented in Chapter 4, was the initial evaluation of the ontology-based operators and the V&V process itself. The second case study, presented in Section 6.1, was conducted with the support of the automatic V&V tool. This case study evaluated the efficiency of the automatic V&V tool, and changed the problem domain to test the domain independence of the V&V process. It also introduced inexperienced ontology engineers and MAS modellers to test the user independence and assess the impact of the initial quality of ontology and models. In the final case study conducted in this thesis, the aim is to apply the automatic V&V process to a set of MAS models developed with a different AOSE methodology, MOBMAS, to illustrate the mostly methodology independence of the process. Experienced stakeholders were in charge of the ontology development and the MAS models development. A comparison of execution times between this case study and the second case study was then possible to consider the impact of experience.

The problem domain chosen for the case study is the aircraft turnaround problem. As the domain independence of the V&V process had already been tested in Section 6.1, it is appropriate to revisit this problem. Moreover, a comparison between the manual process and the process with the support of the tool for the same problem could then be made. The MAS models are developed by the same developer who participated in the case study described in Section 6.1. The developer has experience working with AOSE methodologies. The developer did not work with the MOBMAS methodology previously or studied the aircraft turnaround problem prior to his involvement in this case study. The ontology engineer has extensive experience developing ontologies and is familiar with the aircraft turnaround problem. This case study configuration is designed to achieve an initial version of domain ontology and MAS models of higher quality than the ones used in the case study of Section 6.1. It is expected that this would affect the execution time of
the case study. As the problem domain is the same one as described in Chapter 4, its description will be omitted here (see Section 4.3.1).

6.4.1 Task 1.1: Domain Ontology Acquisition – Iteration 1

In Task 1.1, an ontology describing the problem domain is acquired. The development case study presented in Chapter 4 showed that no appropriate ontology describing the aircraft turnaround problem could be found through a search in Swoogle. However, the ontology produced in that case study perfectly described the domain and could, thus, be reused. The ontology engineer adapted that ontology for the needs of the current case study. In the first case study, the ontology-based V&V process was applied manually, as no support tool was available. In that case, the ontology was composed of a set of classes related with each other through object properties. As the last case study would be conducted using the automatic V&V process, the requirements of the ontology were different. The concepts describing the problem domain had to be represented in the ontology as individuals belonging to a generic class, Thing, as specified in Chapter 5.

The domain ontology defines nine object properties to establish the relations between the concepts of the domain. Achieves relates the system tasks or activities with the goals that each one accomplishes. Needs links the activities with the environment entities that they require. Participates associates roles with the activities in which they are involved. Plays defines which roles are played by each agent. Precedes establishes the execution order of activities. Reacts to links agents to the events that trigger their reactions. Finally, sub-activity of, sub-role of and sub-goal of, associate children of activities, roles and goals with their corresponding parents. Figure 6.14 depicts an excerpt of the domain ontology produced during this task.

6.4.2 Task 1.2: Domain Ontology Augmentation – Iteration 1

The support tool received the domain ontology produced by Task 1.1 and the AOSE metamodel described in Chapter 5 as input. The
190 Evaluation of the Automatic V&V Process

Figure 6.14: Excerpt of domain ontology for the aircraft turnaround problem

The augmentation of the ontology was performed automatically with no human intervention. No inconsistencies or cardinality violations were detected by the augmentation module in the domain ontology. Figure 6.15 illustrates how the concepts defined in the excerpt of domain ontology shown in Figure 6.14 are classified according to their AOSE metamodel class.

Figure 6.16 shows the new relations inferred with regards to the instances of concepts Goal and Activity: GoalIsSubGoalOfGoal, GoalIsSuperGoalOfGoal, ActivityIsSubActivityOfActivity, ActivityIsSuperActivityOfActivity, GoalAchievedByActivity, ActivityAchievesGoal, ActivityPrecedesActivity and ActivityFollowsActivity.

Figure 6.17 shows the new relations inferred with regards to the instances of concepts Goal, Activity and Environment entity: ActivityNeedsEnvironmentEntity, EnvironmentEntityNeededInActivity, GoalNeedsEnvironmentEntity and EnvironmentEntityNeededForGoal.

Figure 6.18 shows the new relations inferred with regards to the instances of concepts Role, Agent, Event and EnvironmentEntity: RoleIsSubRoleOfRole, RoleIsSuperRoleOfRole, AgentPlaysRole, RolePlayedByAgent, AgentReactsToEvent, EventStimulatesAgent, RoleUsesEnvironmentEntity, EnvironmentEntityUsedByRole, AgentUsesEnvironmentEntity and EnvironmentEntityUsedByAgent.

Figure 6.19 shows the new relations inferred with regards to the instances of concepts Goal, Activity, Role and Agent: RolePursuesGoal, GoalPursuedByRole, AgentPursuesGoal, GoalPursuedByAgent, RoleParti-
The augmentation module concluded the preparation of the domain ontology by raising the number of object property assertions (i.e. relationships between domain concepts) from 122 to 824.

6.4.3 Task 2.1: Ontology-Models Comparison – Iteration 1

In Task 2.1, the comparison module received the augmented domain ontology and the available MAS models – task, organisation and role model – as input, to automatically apply the ontology-based operators and produce the discrepancy report.

The concepts defined for task model and the ontology were mostly overlapping. However, two concepts defined in the model Deboard and Board crew and crew check had been defined in the ontology as four concepts, namely, Disembark crew, Disembark passengers, Embark crew and Crew check. Consequently, the relations involving the above concepts were also detected as discrepancies. E.g. Deboard Activ-
ityIsSubActivityOfActivity Handle passengers} had not been defined in the ontology whereas the relationships {Disembark passenger ActivityIsSubActivityOfActivity Handle passengers} and {Disembark crew ActivityIsSubActivityOfActivity Handle passengers} had not been defined in the model. The remaining relationship discrepancies were due to a higher level of detail being provided in the ontology. The augmentation mechanism exploited the transitive character of the relations ActivityIsSubActivityOfActivity to generate all possible relations. For example, if the activity Load Baggage is a sub activity of Handle Baggage, and this is in turn a sub activity of Aircraft Turnaround, then the reasoner can establish that Load Baggage is a sub activity of Aircraft Turnaround.
A number of discrepancies in the definition of concepts were detected between the organisation model and the ontology. The concept Ground Staff defined in the model was split into Airport Ground Staff and Airline Ground Staff in the ontology. The concept Pilot was only defined in the model and the concept Passenger was only defined in the ontology. The composite role Airport Staff had been defined in the ontology to encompass the functionalities of the personnel associated with the airport. The functionality of the role Aircraft Maintenance Technician / Engineer defined in the model was covered in the ontology by the separate roles Fueller and Engineer. As above, discrepant relationships were detected corresponding to the missing concepts.

The discrepancy report for the role model showed that most discrepancies in the definition of roles overlapped with the ones detected for the organisation model. However, unlike the organisation model, in the role model, the role Passenger had been defined. This illustrates how the operators application can uncover inconsistencies between models. All authority relations between roles were detected as discrepancies as they had not been originally included in the ontology.
To solve the discrepancies in the definition of the activities, the task model was modified to refine the activity *Deboard* as *Disembark crew* and *Disembark passengers*, and the activity *Board crew and crew check* as *Embark crew* and *Crew check*, as defined in the ontology. The discrepancies in the relationships involving the above concepts were solved accordingly. In other words, the relations involving the concepts *Deboard* and *Board crew and crew check* were removed from the task model and the corresponding relations that used the refined activities were added to the model.

The discrepancies in relationships caused by the transitive character of the object property *ActivityIsSubActivityOfActivity* were not deemed relevant and were ignored. They were not removed from the ontology because in essence they are correct and because the augmentation module would automatically regenerate them in the following iteration.

The definition of the roles in the organisational and role models was modified to comply with the ontology to a great extent. It was decided to split the model concept *Ground Staff* into *Airport Ground Staff* and *Airline Ground Staff*. The concept *Pilot* was not considered to be completely relevant during the turnaround activities.
and it was removed. The concept *Passenger* that had been defined in the role model and the ontology was added to the organisational model, thus removing the inter-model inconsistency. The composite role *Airport Staff* encompassing the functionalities of the personnel associated with the airport was added to the organisational model. It was necessary to further refine the concept *Aircraft Maintenance Technician / Engineer* defined in the models according to the roles *Fueller* and *Engineer* as defined in the ontology. However, based on this discrepancy it was decided that an additional role had to be added to the ontology and models, the *Technician*. The discrepancies in the relationships involving any of the concepts modified above were solved accordingly.

The modification of a number of roles and activities in the model is reflected in the relations of participation (*RoleParticipatesInActivity*) between roles and activities. Relations involving deleted roles, for example *Pilot*, are removed, whereas other involving new roles are added, for example *Technician RoleParticipatesInActivity Routine maintenance* or *Airline ground staff RoleParticipatesInActivity Embark passengers*. As some activities have been modified, their associated relations change accordingly. For example, the activity *Deboard* is divided into *Disembark passengers* and *Disembark crew*. The role *Crew* becomes associated with both activities and *Passenger* only to *Disembark passengers*. The discrepancies in participation relationships involving composite activities, for example *Airport ground staff RoleParticipatesInActivity Baggage handling* were considered to add an unnecessary level of detail and were ignored.

The authority relationships between roles only defined in the role model were added to the ontology with some modifications, as some of them were not considered to accurately represent the relation between roles. For example in the role model it was stated that *Crew RoleControlsRole Ground staff*. However, as the definition of the role *Ground Staff* had changed, the updated relationships should be refined to express that *Crew RoleControlsRole Airline ground staff* and *Crew RoleIsPeerRole Airport ground staff*. Note that the authority relation only linked the *Crew* with the *Airline ground staff*, not with the *Airport ground staff*. This is another example of how the review of de-
196 EVALUATION OF THE AUTOMATIC V&V PROCESS

tected discrepancies can encourage discussions leading to modelling
aspects not previously included in the model or ontology.

6.4.5 Task 1.1: Domain Ontology Acquisition – Iteration 2

In Task 1.1 of the second iteration the original domain ontology
is modified to comply with the decisions made in Task 2.2 of the
first iteration. The modifications were not too numerous, and most
of them involved authority relations between roles, such as the
addition of the relationships Engineer RoleControlsRole Fueller or Crew
RoleIsPeerRole Airport ground staff. The only additions to the ontology
not related to role authority relations were the addition of the concept
Technician and the relationship Technician RoleIsSubRoleOfRole Airport
staff.

6.4.6 Task 1.2: Domain Ontology Augmentation – Iteration 2

The augmentation module processed the refined domain ontology
without detecting any inconsistency or cardinality violations. The
resulting augmented ontology had 901 object property assertions
(relationships) while the original domain ontology had 136 object
property assertions.

6.4.7 Task 2.1: Ontology-Models Comparison – Iteration 2

The operators application module automatically compared the aug-
mented domain ontology produced during by Task 1.2 with the
MAS models. The developer provided a refined version of the task,
organisational and role models, and new versions of the resource,
agent and goal models.

The discrepancy report showed no new discrepancies between
the task and organisational models and the ontology. The task and
organisational models were considered validated. The role model
presented two relevant discrepancies with the domain ontology.
The relations Crew RoleControlsRole Passenger and Airline ground staff
RoleControlsRole Passenger had been defined in the role model but not in the ontology.

The discrepancy report highlighted three concepts defined in the resource model but not in the ontology. These were the environment entities Service facilities, Maintenance facilities and Fuel. In contrast, the environment entities defined in the ontology but not in the resource model were Baggage train, Cargo loader, Cleaning truck, Galley truck, Tug and Wheel chocks. Other concepts only defined in the ontology corresponded to the agents AgentAirlineStaff and AgentAirportStaff. Discrepancies in the relationships involving the above concepts were consequently detected.

The goal model presented a significant number of relevant discrepancies with the ontology, as the modeller had included a goal corresponding to every task to be performed by the system, such as Remove wheel chocks or Crew check. Accordingly, the relationships defined in the model as using these goals were also detected as discrepant.

The agent model overlapped with resource and goal model in the use of goals, agents and resources. The model introduced the use of Events and their relation with Agents. There were four events (and their corresponding relationships) defined only in the model: Technician reports problem, Embarking has commenced, Aircraft is landing and Aircraft is ready for take-off. The events (and associated relationships) defined by the ontology that were not included in the model were Baggage has been loaded, Non routine operations have concluded, Routine maintenance operations have concluded and Passengers have embarked. Other discrepancies in relationships that involved concepts defined in both the agent model and ontology were Passenger AgentReactsToEvent Airbridge is set and Passenger AgentReactsToEvent Embarking has commenced and had only been defined in the model. Conversely, the relations that had been defined only in the ontology were Airport ground staff AgentReactsToEvent Aircraft has been refuelled, Airport ground staff AgentReactsToEvent Airbridge is set and Engineer AgentReactsToEvent Airbridge is set.
6.4.8 Task 2.2: Discrepancy Removal – Iteration 2

The two discrepancies in the authority relationships between the ontology and the role model, Crew RoleControlsRole Passenger and Airline ground staff RoleControlsRole Passenger, were resolved by adding those two relationships to the ontology.

For resolving discrepancies in the definition of environment entities, resources that cannot be used for multiple tasks at the same time are considered relevant to model. Fuel would not be such a relevant resource as it is assumed that the airport provides a sufficient amount of fuel to service all its aircrafts. The environment entity Maintenance facilities refers to the equipment used by maintenance personnel to carry out their duties. It is assumed that maintenance equipment is assigned to individuals, therefore if an engineer is available to perform non-routine maintenance tasks, the required equipment will be also available. The environment entity Service facilities was deemed too general as the ontology defines specific resources for the different activities. Therefore, the three environment entities not defined in the ontology, Fuel, Maintenance facilities and Service facilities, were removed from the models. The environment entities defined only in the ontology, Baggage train, Cargo loader, Cleaning truck, Galley truck, Tug and Wheel chocks had to be added to the model together with the corresponding relationships with agents.

The two agents defined only in the ontology, AgentAirlineStaff and AgentAirportStaff, correspond to aggregated roles with no real functionalities and should not be ignored. These two agents had been generated by the automatic augmentation mechanism based on the existence of the aforementioned roles.

All goals defined in the model but not in the ontology that corresponded to system tasks were deemed unnecessary and therefore were removed from the models. The associated relationships linking the goals to agents were also removed.

Of the four events defined only in the model, three of them, Technician reports problem, Embarking has commenced and Aircraft is landing, and their associated relationships were considered relevant and were added to the ontology. However, the event Aircraft is ready for take-off and its corresponding relations were removed from the model as it
was considered that the turnaround process concludes before the actual aircraft take-off. It was decided that all the events defined by the ontology that were not included in the model, i.e. *Baggage has been loaded, Non routine operations have concluded, Routine maintenance operations have concluded* and *Passengers have embarked*, should be added to the model together with the relationships that involve them. The relationships included only in the agent model, *Passenger AgentReactsToEvent Airbridge is set* and *Passenger AgentReactsToEvent Embarking has commenced* were considered incorrect and, hence, removed from the model. The relationships *Airport ground staff AgentReactsToEvent Aircraft has been refuelled* and *Airport ground staff AgentReactsToEvent Airbridge is set* defined only in the ontology were correct and had to be added to the agent model. However, the relationship *Engineer AgentReactsToEvent Airbridge is set* was incorrect and had to be removed from the ontology. In the previous iteration, it was decided to add the *Technician* as a new role. The ontology engineer did not correctly propagate this modification. Because of this discrepancy, the error was detected and the relationship in the ontology was modified to *Technician AgentReactsToEvent Airbridge is set*.

### 6.4.9 Task 1.1: Domain Ontology Acquisition – Iteration 3

Based on the decisions made in Task 2.2, the domain ontology was manually modified to become closer to convergence with the MAS models. The ontology largely remained stable requiring very minor modifications. Most modifications involved the definition of events and their associated relationships. For example, the events *Technician reports problem, Embarking has commenced* and *Aircraft is landing* and their associated relationships were added.

### 6.4.10 Task 1.2: Domain Ontology Augmentation – Iteration 3

The augmentation module received the refined domain ontology produced during Task 1.1 as input. The augmentation process concluded without any inconsistency or cardinality violation being detected. The number of object property assertions increased from 141 to 916.
6.4.11 Task 2.1: Ontology-Models Comparison – Iteration 3

The application of the ontology-based comparison operators by the operators application module produced the discrepancy report. The discrepancy report showed no relevant differences between the augmented domain ontology and the MAS models. All models were as such considered validated and the V&V process concluded.

6.5 Conclusions of the V&V Process Evaluation

The main goal of this case study was to test the methodology independence of the ontology-based V&V process and its associated support tool. This case study applies the V&V process to another AOSE methodology, MOBMAS. The transition to MOBMAS, as an AOSE methodology, was largely seamless. The domain ontology was acquired in a similar way as for the previous case study. The ontology did not make any assumptions about any AOSE methodology. The augmentation of the domain ontology was conducted automatically and knowledge was generated as described by the AOSE metamodel. The application of the operators was also done without human intervention. The support tool automatically generated reports for the correct set of models and detected discrepancies for the adequate concepts and relationships. For instance, the definition of the goal model in MOBMAS differs substantially from the one of ROADMAP. The former includes hierarchical relations between roles, whereas the latter does not (ROADMAP includes them in the organisation model). The comparison module successfully identified missing hierarchy relations only for the goal model, not for the organisation model. Indeed, the execution times of the case study were no worse than in the previous case study (see Section 6.1) where the original set of operators was used by the automatic V&V process. Rather, the times were shorter (see discussion below), which seems to indicate that the application of a different AOSE methodology did not hinder the normal application of V&V process.

The comparison of the overhead (not including model development and improvement) introduced by the ontology-based V&V
The reduction in the case study times from the second to the third case study can be understood when comparing the individual task times for both case studies (Table 6.4). The third case study concluded in three iterations whereas the second case study required four iterations. The faster execution of the development case study can be attributed, amongst other factors, to the degree of participant expertise. Compared to the case study presented in Section 6.1, both the ontology engineer and MAS modeller were more experienced,
which in turn determined a higher initial quality of ontology, and models that required fewer modifications and converged faster. As shown in Table 6.5, the domain ontology required fewer modifications in the third case study (1.5 hours of work) than in the second case study (4 hours of work).

In fact, the modification of the ontology in the third case study required 1.5 hours while the models required 4 hours worth of work. In comparison, in the second case study where the initial ontology was of a lower quality, the models remained more stable than the ontology, requiring only 3 hours worth of work versus the 4 hours taken for the modifications to the ontology. This indicates that although the initial quality of the ontology has an impact on the V&V process by introducing some delay (one additional iteration in this case), it does not severely affect the efficiency of the V&V process, as the models and the ontology finally converged to comply with the client’s conceptualisation. For both case studies, the models were modified to some extent in each iteration to solve errors that,
potentially, could have critically affected the functionality of the final application.

Part of the reduction in time between the second and the third case study cannot, however, be directly attributed to the experience of the ontology engineer. The execution of Task 1.1: Ontology Acquisition required half of the time for the third case study compared with the second case study (Table 6.4). Whilst in the first and second case study there was no initial domain ontology available (and hence a new one had to be developed from scratch), in the third case study it was possible to re-use a domain ontology. As the domain was the same as already explored in the first case study, the domain ontology from the first case study was adapted to produce a higher quality domain ontology for the third case study.

6.6 CHAPTER SUMMARY

The evaluation of the ontology-based V&V process in Chapters 4 assessed its manual application to V&V ROADMAP models. The focus was the semantic adequacy and the formulation of the process. This chapter tested the user, domain and AOSE methodology independence of the V&V process and the support of the automatic V&V tool. The development case studies presented in this chapter focused on removing these validity threats.

The chapter commenced with the description of the first development case study in Section 6.1 and the discussion of its outcome in Section 6.2. The case study showed that the automatic V&V process greatly alleviated the augmentation of the ontology and the applica-
tion of the comparison operators. The tool successfully performed both tasks automatically. The augmentation module extracted a high number of implicit relations from the asserted knowledge and automated the ontology consistency and cardinality violation checking. The comparison module produced a discrepancy report for each model that was used to improve the ontology and the models. By changing the configuration from the previous case study, i.e. a different problem domain (wine broker), and different participants (ontology engineer and model developer), the domain and user independency of the V&V process were tested. The literature survey showed a set of commonly used concepts and relationships across most extant AOSE methodologies. MOBMAS is no exception and also uses many of these common concepts and relationships. However, the distribution of concepts and relationships among the models defined by MOBMAS does not fully match the one used by many common AOSE methodologies, which was also adopted for the ontology-based validation operators defined in Chapter 5. Section 6.3 adjusted the definition of the ontology-based validation operators to comply with the particular description of models defined by MOBMAS. The adjustment of the ontology-based validation operators described in Section 6.3 had to be reflected by the automatic V&V process. Technical modifications in the operators application module to support the new definition of operators were also described in Section 6.3. The AOSE methodology independency of the V&V process was tested in the final case study presented in Section 6.4 and its conclusions discussed in Section 6.5. The aircraft turnaround problem used in the case study in Chapter 4 was revisited to develop the MAS models for MOBMAS. The models were validated and verified faster than in the second case study (also automated), indicating that use a different AOSE methodology did not create a challenge for the V&V process. The faster execution of the case study could be attributed to a greater degree of expertise of the ontology engineer and modeller, and, hence, higher quality of the initial artefacts. The next chapter will conclude this thesis and will outline its future work.
CONCLUSION

This chapter concludes the thesis and recapitulates its main points. The chapter is structured as follows: Section 7.1 summarises the undertaken research; Section 7.2 recapitulates the contribution of this thesis; Section 7.3 describes the research limitations of this thesis and future research lines; Section 7.4 concludes the chapter and the thesis.

7.1 RESEARCH SUMMARY

This thesis explored the usage of ontologies, as knowledge representation tools with underlying formal semantics, to validate and verify (V&V) multi agent systems (MAS) analysis models. The outcome of this research is a new ontology-based MAS models V&V process add-on. Using the automatic reasoning capabilities of ontologies, an automatic support of the process was developed to facilitate its application. The process and its automatic support tool were iteratively refined and validated through the planning and execution of three software development case studies. The research method leading to this outcome is described in what follows.

A design science research approach consisting of five research phases (discussed in Chapter 3) was adopted to achieve the goals of the thesis (c.f. Chapter 1):

- To first study how ontologies are currently used along the software development lifecycle, in particular in the validation of requirements models in AOSE.

- To design a process to use ontologies for the validation of AOSE software models, that can be adapted for any AOSE development methodology.
• To provide automatic support for validation process, harnessing the formal semantics of ontologies.

• To validate the process and the tool in several case studies.

Phase 1, Research Problem Definition, was the subject of Chapter 2 which presented the literature review. This review highlighted that ontologies have been often used during software development to support various activities and generally improve the value of the produced systems. However, their use during requirements engineering activities to validate that the product being developed complies with the client’s conceptualisation remained mostly unexplored. Furthermore, in Agent Oriented Software Engineering (AOSE), few of the extant methodologies contemplated the use of ontologies and this was only to model the domain problem to support agent communication. Whilst the review also highlighted the lack of consensus in the definition of models amongst extant AOSE methodologies, it was still possible to compile a set of common concepts and relationships representing key aspects of AOSE analysis defined by most methodologies. This formed a set of common models that covered them adequately.

Phase 2, Initial V&V Process Definition, was described in Chapter 4. This defined an initial process filling the research gap identified in the literature review. The process was designed as an add-on applicable to the model development activity assumed defined by an extant AOSE methodology. This process is based on a domain ontology conceptualising the client’s specification. The domain knowledge in the ontology was augmented and structured using AOSE key concepts and relationships to allow an easy comparison with the MAS analysis models. The comparison used validation operators that focus on relevant features of the models based on key elements identified in the literature review. The augmented ontology was used to detect discrepancies. These were compiled into a recommendation report used as a basis to propose improvement to the MAS models. The comparison-improvement cycle was iterated until no further discrepancies between ontology and models are detected.

Phase 3, V&V Process Evaluation, initiated in Chapter 4 focused on evaluating the process and its comparison operators. A development
A case study was conducted where the V&V process was applied to the ROADMAP models developed as part of an aircraft turnaround simulator. The application of the ontology-based operators resulted in the early detection of errors and prevention of rework. This showed that the V&V process was effective in early detection of modelling errors. However, it also confirmed the researcher’s expectation during the initial description of the V&V process, that the client’s conceptualisation can evolve with the development process. In other words, the domain ontology was expected to remain changeable to adapt any new requirements. Furthermore, the V&V process needs to be automated to the maximum extent to enable an efficient application. These extensions to the process were developed in Phase 4 of the research.

Phase 4, Automatic V&V Process Definition, was presented in Chapter 5. This phase started with the refinement of the original V&V process to allow the modification of the ontology to adapt to the changes in the requirements specification. However, the main focus of the extension of the process was automating much of it. In particular, two main aspects of the process were automated: the domain ontology augmentation process and the application of the validation operators. The AOSE key concepts and relationships identified by the literature review were first formalised as an AOSE metamodel. This metamodel was automatically linked to the domain ontology. Then, the automatic reasoning mechanisms of OWL 2 ontologies were exploited to check the consistency of the domain knowledge and to infer new knowledge based on the asserted relationships in the metamodel. The resulting augmented domain ontology was successfully produced without human intervention, except to remove detected inconsistencies in the domain knowledge, and became the input for the automatic operators application mechanism. The ontology-based validation operators were also formalised in a selection ontology to identify only aspects of the augmented domain ontology that should be compared with the relevant elements of the MAS models. As a result, a discrepancy report that also highlighted dependencies between the models was automatically generated to assist the model-ontology convergence decision making.
Phase 5, *Automatic V&V Process Evaluation*, was described in Chapter 6 and assessed the adequacy of the automatic V&V process in two more development case studies. In the first, ROADMAP models were developed for the problem domain of a wine broker. In the second, the aircraft turnaround problem was revisited but to develop the corresponding MOBMAS models. By varying the parameters of the case studies, different validity threats were mitigated. The main aspects of the process that were tested were the following: its user independence, its problem domain independence and finally its development methodology independence. The evaluation of the process also highlighted as expected the significant reduction in the time required to apply the V&V process.

### 7.2 Research Contribution

The literature review presented in Chapter 2 showed that there is an important research gap regarding to approaches to validate requirements specifications taking into account the problem domain. As the development process proceeds fixing errors becomes costlier. Indeed, early validation of requirements models can prevent rework or building a system non-compliant with client’s specification. The situation for AOSE is similar: Although there is a great deal of research on formal techniques to verify the structure and properties of MAS models, little work was done to validate the models taking into account the problem domain. The V&V process produced in thesis fills in this gap by ensuring that the developed analysis MAS models are consistent with each other and with the client conceptualisation of the problem.

The literature review highlighted that there is no standard de facto amongst extant AOSE methodologies and none of them support the domain-dependant validation of models. Having a variety of methodologies can be beneficial in early stages of development of a new paradigm. However, the AOSE community needs to establish standard approaches to allow the AOSE paradigm to become widely adopted. Semantic commonalities between models of extant methodologies were identified in this thesis to ensure interoperability of
the V&V process. This contributes to the standardisation efforts in the AOSE field. The V&V process is usable as an add-on that can be applied to V&V the MAS models developed as part of any extant AOSE methodology.

Ontologies have been widely used in the past to improve several activities of the software development life cycle or their work-products (see Chapter 2). However, few of those usages have harnessed the automatic reasoning capabilities of OWL 2 ontologies. More specifically in AOSE, methodologies that include the construction of ontologies as part of their development activities confine their use as vocabularies to only facilitating the communication between agents. This thesis harnesses the automatic reasoning capabilities of OWL 2 ontologies in the automatic domain ontology augmentation process. This process defined in this thesis produces an ontology to support the completion and ensure correctness of the MAS models. The automatic augmentation process drastically increments the number of explicit knowledge assertions available in the V&V ontology, which in turns raises the possibility of the operators application to uncover hidden requirements or inconsistencies.

The V&V process has been defined to introduce little overhead in creating the MAS models. The process is conducted iteratively accommodating a typical AOSE methodology lifecycle. It incrementally and concurrently validates the MAS models produced. However, the ontology augmentation and operators application tasks can be computationally demanding and this added time constraint may need to be accounted for in planning of some projects. This automation is nevertheless critical to reduce the human intervention to simply deciding on best way to remove modelling errors. The development case studies described in Chapter 6 illustrated how the automatic V&V process dramatically improves the V&V times when compared with the manual application of the process in the case study in Chapter 4. These case studies also showed that the expertise of the ontology engineer and AOSE developer (hence, the initial quality of the artefacts) had acceptably limited impact on the V&V times. Even when the stakeholders were unexperienced, the domain ontology and the MAS models converged relatively fast (only one iteration more than the case study conducted by experienced participants) and
the final outcome was eventually the same, a set of MAS models fully compliant with the client’s specification. This makes the automatic V&V process particularly suitable in projects where the domain is complex or unknown, and where the developers are relatively new to AOSE (even if they are experienced in other technologies) to guide their work and avoid errors. However, the V&V process does not bypass the need for the agent modeller and ontology engineer to acquire certain familiarity with the problem domain, as this is an inherent requirement of any development process.

7.3 RESEARCH LIMITATIONS AND FUTURE RESEARCH DIRECTIONS

The research plan in Chapter 3 anticipated that some tasks of the V&V process could be too arduous to be conducted manually. This was confirmed in the first development case study (Chapter 4) where in a medium size domain the augmentation of the ontology and the application of the operators had significant time requirements. Later case studies (Chapter 6), where the V&V of models was supported by the automatic domain ontology augmentation and operators application modules, required half the manual application times (considering the same number of iterations). In other words, the manual V&V of models (unassisted by the support tool) for large problem domains is too costly and best avoided.

The semantics underpinning the automatic augmentation is an AOSE metamodel. This metamodel currently does not exhaustively cover all concepts and relationships defined by every AOSE methodology. It is currently only a subset of the FAML metamodel (Beydoun et al. 2006a, 2009; Low et al. 2009). FAML has been shown to cover elements described by most extant AOSE methodologies at analysis and design level, including agent planning and internal mental states. The full FAML was not used to avoid complexity beyond the scope of the current work. The FAML subset described in Chapter 5 was considered to adequately represent the most common elements used across many extant AOSE methodologies. This fulfilled the goal of the thesis to provide a V&V process and a proof of concept tool to effectively improve the quality of MAS models. The next step of
this research would be to replace the AOSE metamodel by the full FAML metamodel. This would better support adding the V&V of the models to any of the extant AOSE methodologies.

Amongst the other issues avoided, as to not obscure the results of the thesis, is automatic resolution of synonyms. Comparison of ontology elements with model elements deployed a simple text comparison. For instance, *Plane* and *Aircraft* often refer to the same concept interchangeably. This was solved by manually parsing the domain ontology and the MAS models to unify the names of shared elements. The better automated solution would be the development of a semantic comparison module. This should also be considered in future work.

The three development case studies conducted have been configured to mitigate several validity threats. To assess the user independence of the process the participants, i.e. ontology engineers and AOSE modellers were different. To test the domain independency, two different domain problems were used, the aircraft turnaround problem and the wine broker problem. The AOSE methodology independence was also evaluated by using two different AOSE methodologies, ROADMAP and MOBMAS. The outcome of the case studies indicates that the ontology-based V&V process is, indeed, independent of the participants, problem domain and AOSE methodology. However, to further mitigate against the validity threats, further case studies could be conducted.

Finally, the ontology V&V process has been designed for the AOSE paradigm. The AOSE metamodel developed to support the augmentation process represents a significant modelling effort. MAS models are less commonly understood, complex and prone to modelling errors. This easily justifies the overhead of applying the V&V process. However, it is theoretically possible to adapt the process to other development paradigms (as long as they support iterative modelling), for example object oriented methodologies. In fact, the adaptation of the automatic V&V process may be relatively simple, as the new paradigm metamodel could be used as input together with the domain ontology. This could also be an objective of future research.
7.4 CONCLUDING REMARKS

This thesis has successfully introduced and validated a novel approach for an ontology-based validation and verification of MAS requirement models. An add-on process was developed to conduct an early validation of requirements models to prevent compounding errors and to reduce further related development costs. This takes into account the client’s problem conceptualisation. The process validates the MAS models for completeness against the client’s conceptualisation and verifies their structure for consistency. This is a novel AOSE contribution. It is the more significant due to automating the bulk of the complex tasks in the validation process. This harnessed the automatic reasoning opportunities offered by the use of formal ontologies. Towards interoperability of the process, an AOSE metamodel is used to describe common features of agent requirement models.

The process supports rectifying any problems detected in the models by suggesting to the developers changes to the requirements that may have remained undetected. The process effectiveness is validated using different cases studies. Validity threats are mitigated against by ensuring that different ontology engineers and AOSE modellers are used, that different problem domain of the applications being developed as well as different AOSE methodologies are changed throughout varying case studies.


References

5333 of Lecture Notes in Computer Science, pages 584–594. Springer Berlin / Heidelberg. (Cited on page 23.)


