

Acknowledgement

This is my master thesis, the final knot in the process of achieving my Master degree at the Institute of Informatics, University of Oslo. The work on this thesis was done at SINTEF, Department of Information and Communication Technology, Cooperative and Trusted Systems. I am very grateful for having the opportunity to use their facilities as well as being able to sit in on an actual research project.

I would specially like to thank my supervisor, Dr. Arne-Jørgen Berre for his accommodating attitude, as well as his guidance throughout the process. Furthermore, I would like to thank my fellow students, Asaf Mohammed Khan and Khudija Mahmood, for their motivation and practical advises. Many thanks also go to Ida Solheim for invaluable help during the final phase.

Finally, I want to thank my husband Ævar for encouraging me to start on this process in the first place as well as his support along the way. At the same time I want to thank my two daughters, María and Sigríður Freyja, for their patience and understanding.

Oslo, November 15th 2005
Kristrún Arnarsdóttir
Abstract

How to enable enterprises to interchange autonomously designed information addressing the same domain, is an active research area. Present technology allows enterprises to exchange data, but when it comes to interpreting the content or understanding the data, there is a problem. This problem has to do with semantic interoperability; enabling computers to understand the meaning of information being exchanged.

Two recent initiatives make promises regarding interoperability issues. Ontologies have been adopted by the field of Artificial Intelligence (AI), and play an important role when envisioning a computer-understandable and executable Web referred to as the Semantic Web. An ontology and related technology is to directly address semantic issues by providing an explicitly defined meaning of available services and the information to be exchanged. The Model Driven Architecture (MDA) on the other hand, addresses interoperability by addressing the conflicts introduced due to the different technological platforms chosen when implementing systems. The MDA framework enables developers to carry out their work at a higher abstraction level as well as applying model transformations to automatically generate technology-dependent parts of systems.

Semantic mapping denotes the task of capturing relationships between semantically similar terms in different data sources as well as supplying the techniques needed to convert between them. This thesis investigates how ontology is used in relation to semantic mapping with emphasis on what distinguishes the ontology-based approach from the MDA approach. An effort is made to explain what ontology denotes within the field of AI as well as to compare the ontology technology used to construct and apply ontologies, to the model transformation technology available within the MDA framework. This thesis is also meant to give the reader a good understanding of the challenges involved when trying to achieve semantic interoperability between enterprises.

The main contribution of this thesis is the account of similarities and dissimilarities between the ontology-based approach and the MDA approach to semantic mapping. Important questions are whether those two are alternative approaches, or whether they may be combined in a value-adding way.
# Table of content

ACKNOWLEDGEMENT ................................................................................................................... I

ABSTRACT .......................................................................................................................................... I

TABLE OF CONTENT ....................................................................................................................... I

LIST OF FIGURES.......................................................................................................................... V

LIST OF TABLES................................................................................................................................ VII

1. INTRODUCTION......................................................................................................................... 1
   1.1 PROBLEM .................................................................................................................................. 1
   1.2 SCOPE ...................................................................................................................................... 1
   1.3 GOAL ....................................................................................................................................... 2
   1.4 APPROACH ............................................................................................................................... 3
   1.5 STRUCTURE .............................................................................................................................. 4

2. INTEROPERABILITY IN DIFFERENT CONTEXTS...................................................................... 5
   2.1 THE ROOT OF THE PROBLEM ............................................................................................... 5
   2.2 INTEROPERABILITY – FUNCTIONAL ASPECTS ................................................................. 6
       2.2.1 Information as context ..................................................................................................... 7
       2.2.2 Information Systems as context ....................................................................................... 7
       2.2.3 Business processes as context – different levels ................................................................. 8
       2.2.4 Models as context – MDA (Model Driven Architecture) ................................................... 8
           Different model levels .............................................................................................................. 8
           MOF - Four layered model architecture ............................................................................... 9
           MOF model interchange - XMI ................................................................................................. 11
           Dynamic systems – specifications not captured by models ......................................................... 11
   2.3 INTEROPERABILITY - NON FUNCTIONAL ASPECTS ............................................................ 11
   2.4 INFORMATION – FUNCTIONAL AND NON-FUNCTIONAL ASPECTS .................................. 12
   2.5 MAPPING ARRANGEMENTS ................................................................................................. 13
       2.5.1 Mapping – many to many ................................................................................................... 13
       2.5.2 Standardisation – one to one mapping ................................................................................. 14

3. PROBLEM EXAMPLE – INFORMATION CONFLICTS......................................................... 15
   3.1 E-PROCUREMENT EXAMPLES SPECIFIED ........................................................................ 15
       3.1.1 About the standards ......................................................................................................... 16
       3.1.2 Business level – behavioural aspects ............................................................................... 17
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.3</td>
<td>Message content – structural aspects</td>
<td>18</td>
</tr>
<tr>
<td>3.2</td>
<td>IDENTIFYING GENERAL INFORMATION CONFLICTS</td>
<td>20</td>
</tr>
<tr>
<td>3.3</td>
<td>COMPARING EXAMPLES</td>
<td>21</td>
</tr>
<tr>
<td>3.4</td>
<td>REQUIRED FUNCTIONALITY FOR RESOLVING CONFLICTS</td>
<td>22</td>
</tr>
<tr>
<td>3.5</td>
<td>EXPERIENCE GAINED DURING EXAMPLE CONSTRUCTION AND ANALYSIS</td>
<td>23</td>
</tr>
<tr>
<td>4.</td>
<td>REQUIREMENTS – SEMANTIC MAPPING</td>
<td>25</td>
</tr>
<tr>
<td>4.1</td>
<td>FUNCTIONAL REQUIREMENTS</td>
<td>25</td>
</tr>
<tr>
<td>4.2</td>
<td>GENERAL REQUIREMENTS</td>
<td>26</td>
</tr>
<tr>
<td>5.</td>
<td>ONTOLOGY AND RELATED TECHNOLOGY</td>
<td>29</td>
</tr>
<tr>
<td>5.1</td>
<td>ONTOLOGY – WHAT IS IT?</td>
<td>29</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Ontology’s origin in philosophy – ontological categories</td>
<td>30</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Why did the AI-community borrow the concept?</td>
<td>31</td>
</tr>
<tr>
<td>5.1.3</td>
<td>A look at the problem to be solved – previous knowledge bases</td>
<td>32</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Ontology as explicit conceptualisation – the most common definition</td>
<td>33</td>
</tr>
<tr>
<td>5.2</td>
<td>COMPARING TO IT PERSPECTIVES – ONTOLOGICAL CATEGORIES</td>
<td>34</td>
</tr>
<tr>
<td>5.2.1</td>
<td>IT domain models – static ontologies</td>
<td>34</td>
</tr>
<tr>
<td>5.2.2</td>
<td>IT behavioural models – dynamic ontologies</td>
<td>36</td>
</tr>
<tr>
<td>5.3</td>
<td>HOW TO EXPRESS ONTOLOGY – REASONING</td>
<td>37</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Ontologies – semantic carriers and reasoning capabilities</td>
<td>38</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Languages for constructing ontologies – OWL and RDF</td>
<td>40</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Choosing a language</td>
<td>43</td>
</tr>
<tr>
<td>5.4</td>
<td>COMPARING TO IT PERSPECTIVES – OO MODELLING LANGUAGES</td>
<td>43</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Expressiveness OWL vs. UML</td>
<td>44</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Complementary advantages OWL vs. UML</td>
<td>46</td>
</tr>
<tr>
<td>5.5</td>
<td>SEMANTIC MAPPING – APPLYING ONTOLOGIES</td>
<td>47</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Architecture – different ways of mapping ontologies to information sources</td>
<td>47</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Approaches for establishing ontology-based mapping</td>
<td>48</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Expression mechanism for capturing mapping relations</td>
<td>50</td>
</tr>
<tr>
<td>5.6</td>
<td>SEMANTIC COORDINATION - AUTOMATIC MAPPING</td>
<td>51</td>
</tr>
<tr>
<td>5.7</td>
<td>CONCLUSIONS</td>
<td>51</td>
</tr>
<tr>
<td>6.</td>
<td>MDA TRANSFORMATIONS AND RELATED TECHNOLOGY</td>
<td>53</td>
</tr>
<tr>
<td>6.1</td>
<td>WHAT IS TRANSFORMATION?</td>
<td>53</td>
</tr>
<tr>
<td>6.2</td>
<td>THE ROLE OF MOF AND M2 LAYER</td>
<td>54</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Profiles in general</td>
<td>55</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Ontology UML profile - OUP</td>
<td>56</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Metamodels</td>
<td>57</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Ontology Definition Metamodel - ODM</td>
<td>58</td>
</tr>
</tbody>
</table>
6.3 SEMANTIC MAPPING – ALIGNED WITH THE MDA FRAMEWORK........................................59
6.4 QVT – MDA STANDARD FOR EXPRESSING TRANSFORMATIONS...........................................60
   6.4.1 Abstract syntax - graphical notation..................................................................................62
   6.4.2 Declarative syntax – transformation engines ...................................................................62
6.5 TRANSFORMATION LANGUAGES AND TOOLS .................................................................63
   6.5.1 Atlas Transformation Language (ATL)..............................................................................64
   6.5.2 MOF2.0 – QVT .................................................................................................................65
   6.5.3 UMLx ..............................................................................................................................66
   6.5.4 MOF2Text - MOFScript.....................................................................................................67
   6.5.5 XSLT – while QVT based languages lack tool support......................................................69
6.6 SEMANTIC MAPPING – OCL EVALUATED ..............................................................................69
6.7 CONCLUSIONS ...................................................................................................................70

7. EXISTING SOLUTION APPROACHES – CASE STUDIES ....................................................71
   7.1 ATHENA - INTRODUCTION .................................................................................................71
   7.2 A3 – KNOWLEDGE SUPPORT AND SEMANTIC MEDIATION SOLUTIONS (ONTOLOGY-DRIVEN) .................................................................................................................73
      7.2.1 Ontology Management - Athos ......................................................................................74
      7.2.2 Semantic Annotation – A* ............................................................................................75
      7.2.3 Reconciliation rule creation - Argos ...............................................................................76
   7.3 A6 - MODEL-DRIVEN AND ADAPTIVE INTEROPERABILITY ARCHITECTURES ...........78
      7.3.1 Mapping relations – Semantic Mapping Metamodel (SMM)...........................................79
      7.3.2 Semantic mapping tool – graphical notation ..................................................................80

8. COMPARING APPROACHES – ONTOLOGY-BASED VS. MODEL-BASED ..............82
   8.1 AS TWO SIMILAR ALTERNATIVES ......................................................................................82
      8.1.1 Semantic mapping method – ontology-based vs. model-based........................................83
      8.1.2 Transformation language examples................................................................................86
         Ontology-based ..................................................................................................................87
         Model-based ....................................................................................................................88
      8.1.3 The model-based approach as 2n mappings instead of 2^n .............................................89
   8.2 AS COMPLEMENTARY APPROACHES ...............................................................................92
      8.2.1 Complementary strengths ............................................................................................93
      8.2.2 Possible communication points ....................................................................................95
   8.3 EVALUATING APPROACHES ...............................................................................................97
      8.3.1 Functional requirements – expressiveness ......................................................................97
      8.3.2 General requirements ..................................................................................................98
      8.3.3 Table overview .............................................................................................................99

9. CONCLUSIONS AND FUTURE WORK.............................................................................100
   9.1 CONCLUSIONS ................................................................................................................100
   9.2 FUTURE WORK ...............................................................................................................102
9.2.1 Combining the ontology-based and the model-based approach ........................................ 102
   Exchanging data using ontology-based interchange format .............................................. 102
   Detecting similarities using ontology-based technology ................................................ 103
9.2.2 Data types .................................................................................................................. 104
9.2.3 Reasoning capabilities ............................................................................................. 104
9.2.4 Usability tests .......................................................................................................... 105

10. REFERENCE .................................................................................................................... 106

APPENDIXES ......................................................................................................................... 111

A. DIFFERENT ONTOLOGY DEFINITIONS ............................................................... 112
   A.1 Ontology explained using natural language ......................................................... 112
   A.2 Ontology explained using logic ........................................................................... 113

B. ARCHITECTURE OF ONTOLOGIES - DIFFERENT LEVELS ......................... 114

C. METHODS AND CONCEPTS ADDRESSING ONTOLOGY CONSTRUCTION .... 116
   C.1 Methodology ........................................................................................................ 116
   C.2 Ontology integration and merge ......................................................................... 117
      C.2.1 Integration ..................................................................................................... 117
      C.2.2 Merge ........................................................................................................... 118

D. MODEL-BASED MAPPING – MOFSCRIPT EXAMPLE ........................................ 119
List of figures

Figure 1. Questions addressed in relation to thesis structure .................................................4
Figure 2. Integration approach illustrated by using architectural perspectives ......................7
Figure 3. Interoperability in context of different model levels ...............................................9
Figure 4. Metalayers for defining UML models ...................................................................10
Figure 5. Different information aspects needed in an e-commerce setting .........................12
Figure 6. Star arrangement – many to many mapping ..........................................................13
Figure 7. Standardisation – one to one mapping .................................................................14
Figure 8. UBL and IDA - activity diagrams ........................................................................17
Figure 9. Conflicts caused by different naming conventions ..............................................18
Figure 10. UBL and Compiere – order diagram .................................................................19
Figure 11. Library of reusable knowledge-based software components ..............................31
Figure 12. Structure of expert systems ................................................................................32
Figure 13. Prolog code, an example of knowledge base ......................................................32
Figure 14. Order document and related conceptual model ..................................................35
Figure 15. Evaluation framework, expressiveness vs. reasoning capabilities .....................39
Figure 16. OWL as web-based architecture ........................................................................41
Figure 17. Example of OWL syntax - created by using Protégé with owl plug-in ...............42
Figure 18. Different ways to use ontologies - architecture ..................................................47
Figure 19. Example of C-OWL, OWL extension for defining explicit mappings ...............51
Figure 20. Transformation as an n-ary association ..............................................................54
Figure 21. Mapping as set of rules and techniques ..............................................................54
Figure 22. Illustration of the relationship between MOF architecture and MDA transformations .............................................................................................................55
Figure 23. Applied stereotypes with tagged values ..............................................................56
Figure 24. The Ontology UML profile class-oriented stereotypes .......................................56
Figure 25. Ontology modelling in the context of MDA and Semantic Web .............................58
Figure 26. Mapping between models - as MDA update transformation ..................................60
Figure 27. Basic pattern when transforming between models .............................................61
Figure 28. Relationship between different MOF2.0 QVT sub-languages .............................63
Figure 29. MOF2.0 QVT relation expressed using both textual and graphical notation .......66
Figure 30. UMLX rules expressed using graphical notation .................................................67
Figure 31. Transformations – Model to Text vs. Model to Model .......................................68
Figure 32. ATHENA, the scope of A3 and A6 illustrated ........................................73
Figure 33. A3 – Tools and process needed in order to create reconciliation rules ........74
Figure 34. A3 – Ontology chestnut ............................................................................75
Figure 35. Example of Jena rules, including a functor .............................................77
Figure 36. Semantic Mapping Metamodel (SMM) capturing A6 – semantic mapping....79
Figure 37. ATHENA Semantic Mapping Tool – Graphical notation .......................80
Figure 38. Semantic mapping in context – Model-based vs. Ontology-based..........83
Figure 39. Semantic mapping – ontology-based vs. model-based .........................84
Figure 40. Mapping between fields – graphical notation ........................................87
Figure 41. Model-based mapping – graphical notation ...........................................89
Figure 42. The model-based approach – current technology ...................................90
Figure 43. Alternative model-based approach – XMI for data interchange ..........91
Figure 44. The model-based approach as the ontology-based approach ...............92
Figure 45. The ontology-based approach and related technology .......................92
Figure 46. Possible communication points – ontology-based vs. model-based ........96
Figure 47. Combining approaches – Converting between transformation languages ...103
Figure 48. Combining approaches – utilising complementary strengths ...............103
Figure 49. Utilising complementary strengths illustrated with a use-case ..............104
Figure 50. Illustrating relationship between different systems of natural language ...113
Figure 51. Different ontology layers – Ontology architecture ..............................115
Figure 52. Illustrating ontology integration .........................................................117
Figure 53. Illustrating ontology merge and merge-alignment ...............................118
List of tables

Table 1. Overview of non-functional quality characteristics ..............................................11
Table 2. Different types of information conflicts. .................................................................20
Table 3. Concrete cases of information conflicts .................................................................21
Table 4. Functionality needed in order to solve various information conflicts ...............22
Table 6. Semantic mapping – general requirements .............................................................28
Table 7. Semantic carriers needed to capture static domain knowledge .........................38
Table 8. Inference mechanisms related to different semantic carriers ...............................39
Table 9. OWL stack of languages (XML, RDF, OWL) ...................................................41
Table 10. OWL features with no UML equivalent ..............................................................45
Table 11. Evaluation of OCL .............................................................................................70
Table 12. A3 – Semantic Annotation ..................................................................................76
Table 13. Different tasks involved during semantic mapping ..........................................84
Table 14. Example languages using the the ontology-based approach .........................87
Table 15. Example languages using the model-based approach .....................................88
Table 16. Elaborating on complementary strengths ........................................................93
Table 17. Functional requirements – required expressiveness of languages ................97
Table 18. General requirements – emphasis, model-based vs. ontology-based ............98
Table 19. Comparing approaches – table overview .........................................................99
1. Introduction

1.1 Problem

How to enable enterprises to interchange autonomously designed information addressing the same domain, is an active research area. Present technology allows enterprises to exchange data, but when it comes to interpreting the content or understanding the data, there is a problem. This problem has to do with semantic interoperability; enabling computers to understand the meaning of information being exchanged.

1.2 Scope

In order to exchange information between enterprises, a relationship between semantically similar but autonomously designed data needs to be established. In addition, techniques needed to convert between different ways of representing the data when using e.g. different data types, need to be identified. Capturing relationship between semantically similar terms as well as supplying the techniques needed to convert between them is here referred to as semantic mapping. Semantic mapping is needed in order to create executable code that automatically converts data from one way of representing it, to another. Below is an explanation of two approaches for obtaining semantic interoperability; the model-based approach and the ontology-based approach.

Semantic mapping needs to be carried out in some context, prior to information interchange. The model-based approach to semantic mapping emphasises the use of platform-independent models as context for identifying relations between semantically similar terms. Platform-independent models (PIMs) are central in Object Management Group’s (OMG)\(^1\) vision for system development, referred to as Model Driven Architecture (MDA). Models are to drive the development process where PIMs are system specifications independent of technical details related to a chosen implementation platform [33]. Transformation is a central concept in MDA addressing how to convert one model into another model of the same system and further into executable code. Today’s de facto standard for creating software models during object-oriented approach for system design, is the Unified Modelling Language (UML) [63].

Ontology is to directly address semantic issues by providing explicitly defined meaning of the information to be exchanged. The concept, ontology, has been adopted from philosophy by the field of Artificial Intelligence (AI) and is identified as one of the necessary elements in the Semantic Web [75]. The Semantic Web is a vision of an executable web, extending the capabilities of the World Wide Web, as we know it. Currently available information on the Web requires human interaction in order to interpret its content. The Semantic web is to

contain executable services as well as ontologies that explicitly describe the services provided, including their message content. This is to result in an executable web with clearly defined semantics. The notion of the Semantic web is lead by W3C and defined to be a “common framework allowing data to be shared and reused across application, enterprise and community boundaries” [75]. Web Ontology Language (OWL) [73] is the recommended standard for capturing ontologies in the context of the Semantic web.

1.3 Goal

The goal of the thesis is to investigate how the ontology-based approach can aid the model-based approach in obtaining semantic interoperability.

In order for me to suggest whether and how to employ ontologies, a number of questions need to be answered.

1. What is ontology?
2. How is ontology linked to information sources when explicitly defining their semantics?
3. Can MDA support semantic mapping as this indicates mapping between different systems and not between different models of the same system?
4. Do ontologies and PIMs overlap?

Frankel et al. [21], assert that there is an overlap between modelling languages used to realise the semantic web, and the UML languages used to capture PIMs. The difference lies in complementary strengths reflected by the different concerns and interests of the two communities. The one is a community that deals with developing industrial software applications, and the other is a community that deals with knowledge representation and how to precisely describe the meanings of terms. The strengths of MDA and related technology are model management, graphical notations of modelling languages and wide tool support, while the strengths of semantic-web-related technology is reasoning capabilities.

Given that the above is true, the following questions need to be answered:

5. What are reasoning capabilities, and are reasoning capabilities essential during semantic mapping?
6. Can MDA transformation technology support semantic mapping:
   a) through tool support?
   b) through graphical notation of languages involved?
7. How do ontology-based and MDA-based approaches to semantic mapping differ, and what are their respective strengths?

---

2 W3C: World Wide Consortium - public website: [http://www.w3.org/](http://www.w3.org/)
1.4 Approach

The approach taken to find answer to these questions is a combination of literature survey and case studies.

The case to be studied is related to a large European project named ATHENA³. ATHENA is a project involving number of subprojects and partners working together to achieve the following vision “By 2010, enterprises will be able to seamlessly interoperate with others” [32]. The case study will focus on two of ATHENA’s subprojects, led by different research organisations. The one is to provide an ontology-based architecture to facilitate interoperability and is led by LEKS⁴ in Italy. The ontology-based architecture is to support construction of ontologies as well as their application by means of semantic annotation. This subproject is named A3. The other subproject is led by SINTEF⁵ in Norway and its main objective is to facilitate interoperability by providing “communicating systems, a platform independent, model driven and adaptive architecture” [32]. This subproject is named A6. One of A6’s sub-goals is to provide semantic mapping and mediation technologies. Semantic mapping is to be supported by a mapping tool. A preliminary version of this mapping tool has been developed based on a specification of a model-based framework [34].

I will look into the two projects by studying project documentation in order to understand what semantic annotation denotes in A3. In addition, I will study the model-based approach to semantic mapping being implemented in A6, and its possible enhancement by using the ontology-based framework of A3. The ATHENA project is supposed to be a source of technical innovation guided by leading independent research centers. This implies that the approach to achieving interoperability in ATHENA is to reflect the current state of the art regarding available technology and methods for achieving interoperability.

The ATHENA project will focus on providing solutions for business processes crossing enterprise boundaries. ATHENA’s requirement definitions address various business scenarios, including Collaborative Product Design, e-Procurement, Supply Chain Management and Product Portfolio Management [31]. In this thesis I will analyse a problem example, based on one of the relevant scenarios identified by ATHENA or e-Procurement. This is done in order to identify requirements necessary to enable semantic mapping and thereby provide a basis for comparing the model-based approach and the ontology-based approach to semantic mapping.

Prior to the case study, a literature survey is necessary in order to understand the concepts and the technology applied in the two subprojects as well as to find an answer to the questions asked. Literature survey and case study in addition to analysing a problem example should provide me with the foundation necessary to determine whether or how the

³ ATHENA public website: http://www.athena-ip.org/
⁴ LEKS: Laboratory for Enterprise Knowledge and System: http://www.iasi.cnr.it/iasi/inglese/lab_leks.html
⁵ SINTEF public website: http://www.sintef.no
ontology-based approach can aid the model-based approach in obtaining semantic interoperability.

**1.5 Structure**

This thesis is organised as follows: Chapter two provides an overview of different ways for dealing with interoperability. In chapter 3, a problem example based on e-Procurement is presented and analysed. In chapter 4, set of requirements necessary to enable semantic mapping, are identified. Chapter 5 and 6 are the results of two literature studies addressing ontologies and MDA transformations respectively, and are to provide the answers to many of the questions asked in chapter 1.3. The following figure is to serve as a map illustrating where the different questions are being addressed. Chapter 7 introduces ATHENA and the different subprojects. Chapter 8 compares the model-based approach and the ontology-based approach with respect to semantic mapping, illustrating similarities as well as complementary strengths. The final chapter 9 contains conclusions and future work.

![Figure 1. Questions addressed in relation to thesis structure](image)
2. Interoperability in different contexts

2.1 The root of the problem

The challenge faced by enterprises, when integrating their business processes, is that their value chains are supported by a number of heterogeneous information systems. In other words, a single enterprise applies various technologies, heterogeneous components and different conceptual models in order to support its businesses, which in turn causes problems when systems and enterprises are to interact. There are a number of reasons for this colourful environment of information solutions within and across enterprises.

One reason is e.g. the fast pace of the IT evolvement. The information technology and the different ways in which IT can be applied, has been evolving extremely fast over the last decades. Enterprises started applying the technology in order to support their backend processing. With the advent of the personal computer the focus shifted from supporting the backend to supporting the employees at the front desk. The availability of the Internet and convergence of information and communication technology enabled distributed IT solutions. During this fast evolvement, enterprises have expanded their IT solutions by integrating already existing and new system components, building a network of heterogeneous components.

Another and important reason has to do with the fact that different systems are autonomously designed. When designing an information solution, it involves making an abstraction of something that exists in the real world. Capturing real-world entities in order to represent them in computer-based solutions has been referred to as conceptual modelling as in [43]. Designing the conceptual model can be an extremely subjective process, as we humans who design these models perceive things differently and are therefore presented to different alternatives as how to model real-world entities. In addition, concepts we capture are commonly labelled using natural languages, and natural languages are ambiguous, which in turn introduces different alternatives. This means that even though two IT solutions are addressing the same domain, their conceptual models can be vastly different.

Present IT solutions, are captured by terms such as B2B, addressing whole value chain and crossing enterprise boundaries. An increased number of requirements towards integrating various solutions is motivated by a need to respond more quickly to changed market situations. Although the computer industry has reached an agreement as to which standards to use when exchanging data, there is still a number of obstacles involved in integration of heterogeneous components. The challenge is e.g. involved in interpreting the content of the data package being exchanged or understanding the content. In that way, the information is the problem:

“If computer is the solution, than information is the problem” [12].

System or information integration has been a research area within the IT field for many years. The various factors needed in order to enable system interaction are commonly denoted by one single term: Interoperability.

2.2 Interoperability – functional aspects

Interoperability is one of the software quality characteristics adopted by ISO/IEC 9126 - 1991 [11]. Functionality is one of the main software quality characteristics, encompassing other sub characteristics, one of them being interoperability. Interoperability has been defined as follows:

“Attributes of software that bear on its ability to interact with specified systems.” [11]:

The definition above is very general but at the same time, clarifies that the term interoperability is used to denote all the different factors that enable systems to interact. Interoperability does thereby address information systems’ ability to interact despite the fact that they where not originally intended to do so. During interaction, different problems will appear precisely because the systems where not designed to interact in the first place. Even though two systems are apparently addressing the same domain, they are bound to vary in terms of programming languages, databases, applied protocols, and the way in which real-world entities are captured or modelled. Some of the reasons for this are mentioned in the previous chapter. In order to achieve interoperability, a variety of different problems caused by heterogeneous information systems, need to be addressed and solved. In order to enable interoperability, enterprises need to present their services and systems in similar context that is by using similar viewpoints.

At least two different dimensions that serve as context, are repeatedly used, [57], [19]. One dimension reflects the information systems and the different ways in which a system can be constructed. This includes e.g. different technical platforms and different representation languages at all levels, requiring the ability to translate information from one format to another. The other dimension is the information to be exchanged as information systems are used to process and exchange information. When discussing interoperability in terms of exchanged information, categories such as structure, syntax and semantics are commonly used, ref. [57].

Another dimension includes the business processes or the whole value chain. In today’s society, the notion of a system is not so clear anymore, partly because information is no longer created and processed within the same system. The convergence of communication technology and information technology now enables information to cross system boundaries as well as enterprise boundaries. This requires a level above the information systems when addressing interoperability.

In addition to this, models in general can be used to establish a context for addressing interoperability, viewing the system from a higher abstraction level. We use models to specify the behaviour of information systems and the way in which different system components interact. We also use models to describe business processes as well as to get an overview of how our data and information are structured. MDA (Model Driven
Architecture) [35] provides definitions for different levels of models as well as definition of the architecture of the information being captured by the models. These definitions can also serve as dimensions when addressing interoperability.

2.2.1 Information as context

When analysing autonomously designed information, different kinds of conflicts appear. These conflicts are basis for identifying different interoperability types.

Sheth [66], explains the different interoperability types as follows:

- **Syntax**: This includes different formats for data representation as well as different data types.
- **Structure**: This includes different languages for model representation, different ways to structure information as well as different languages for model interchange.
- **Semantics**: This addresses the different terms used to describe data. Semantic issues are extremely domain specific as they address the meaning of terms being used to capture and describe domain specific aspects.

Due to the fact that different interoperability issues have been an active research area for many years, a lot of progress has been made, and solutions identified as how to solve these different types of conflicts. Still there are many unanswered questions. Researchers seem to agree that the most difficult problem that still remains unsolved has to do with semantics and how to enable computers to understand the meaning of the terms used when describing information.

2.2.2 Information Systems as context

When interoperability is viewed in the context of the system, the architectural perspective is commonly used as in [82]. Interoperability is discussed in the context of different architectural layers, such as user-interface layer, application or service layer and database layer. Integration at database layer involves e.g. data-schema integration in order to provide uniform data access. The following figure illustrates these perspectives by presenting simple architectural view.

![Diagram](image_url)

**Figure 2.** Integration approach illustrated by using architectural perspectives.
2.2.3 Business processes as context – different levels

In [57], Sheth identifies system heterogeneity as one of the dimensions for viewing different heterogeneity issues that need to be addressed in order to enable interoperability. He identifies two levels of system heterogeneity, the information system and the platform. When taking into account the business processes, which are supported by different heterogeneous systems, the business level needs to be added as top level. Using different levels as dimensions results in the following:

♦ **The business level** addresses various business issues, such as description of business processes, trading partner agreements and legal aspects.

♦ **The application level** addresses the information systems and the different ways to specify and implement application logic and data structures.

♦ **The physical level** addresses the platform used to exchange information, such as different communication protocols, different operating systems etc.

2.2.4 Models as context – MDA (Model Driven Architecture)

Models are abstractions of something that exists in reality. By abstraction I mean that important aspects of the things being modelled are captured and the rest is omitted. Graphical modelling of software is commonly acknowledged as a good practice when specifying different aspects of information systems and business processes or when knowledge about a certain domain is captured. Models can be used in different ways, e.g. to serve as system specification, to help newcomers in gaining an overview of a system as well as to serve as discussion basis throughout the system design and evolution.

The Model Driven Architecture (MDA) is a framework for software development defined by the Object Management Group (OMG) [35]. MDA acknowledges the importance of models, and formal models are in fact the driving force of MDA’s software development life cycle.

**Different model levels**

One of the main ideas behind MDA is that business and application logic should be separated from the underlying architecture used to implement this. This also means that domain knowledge is to be explicitly specified and separated from implementation details. The different levels of models identified by MDA are:

♦ **CIM** (Computation-Independent Model), a software independent model used to describe business processes, stakeholders, departments, dependencies between processes etc.

♦ **PIM** (Platform-Independent Model), software dependent model, used to specify and capture requirements and domain knowledge. PIMs do no include implementation details.

♦ **PSM** (Platform-Specific Model), implicitly include the PIMs as well as platform-specific details added.
By using the term *underlying architecture*, MDA is not addressing the typical architectural perspectives of information systems, such as data, service and user-interface. The architecture here refers to various available standards or frameworks commonly used to implement systems in distributed environments, such as J2EE or .Net [35]. In MDA, the idea is that PIMs are to be automatically transformed to PSMs, where PSMs are refined PIMs. Because platforms like J2EE are in fact standards, it is possible to capture the knowledge needed to convert PIMs to PSMs, and by that creating what in [35], is referred to as *transformation definition*. Transformation definitions are a part of the transformation tool to be used to automatically transform between the different model abstractions or between PIMs and PSMs. The PSMs are in turn to be transformed using a similar approach in order to generate the code complying to the models being specified. Transformations are addressed in more detail in chapter 6.

Transformation between CIM and PIMs is not possible ([35]) because only parts of CIM models are supported by information systems and the choices regarding which parts to support, will always be made by humans.

The MDA approach, applying transformations to automatically construct platform-specific models as well as code, addresses some of the heterogeneity issues that cause interoperability conflicts. Transformation definitions capture information about how to convert between different model views. Therefore, the conflicts caused by different platforms and different programming languages can be ignored allowing us to focus on the conflicts caused by different ways of capturing domain knowledge.

If the different model levels defined by MDA are used to capture specifications of information systems or enterprise processes, these levels can serve as context when achieving interoperability in similar ways as the system dimension identified by Seth in [57], [66]. In order to establish seamless integration between enterprises, all different model levels need to be considered as illustrated in the following figure.

![Figure 3. Interoperability in context of different model levels](image)

**MOF - Four layered model architecture**

At the heart of MDA is a language that is used to define models, called MOF (Meta Object Facility). This language is to ensure the formal definition of other languages standardised by OMG and by that ensuring that tools are able to read these languages. Unified Modelling Language (UML) [63] is one of the languages standardised by OMG and therefore defined
using MOF. UML is also one of the de facto languages used for modelling different software perspectives and therefore commonly used to create software dependent models. The different models specified by MOF are defined using four layered metalevel hierarchy. The following illustration shows a small subset of the metamodel used to define UML.

![Figure 4. Metalayers for defining UML models](image)

The different meta-layers can also serve as context when exchanging information between enterprises. Different layers capture different kinds of information. The figure presented here is copied from [35] and shows a subset of model elements needed to capture static structure and their relevant meta-layers explained as following:

- **M0 layer** captures instances or what we commonly refer to as data.
- **M1 layer** captures the conceptual layer and in this case shows classes which are abstraction of real-world phenomena such as order and customer.
- **M2 layer** captures model constructs available for modelling the conceptual layer, containing a model of a model or the metamodel.
- **M3 layer** is a placeholder for different modelling languages, grouping the different metamodels within each language. In similar ways as M2 layer captures available constructs for modelling the layer below, M3 layer captures constructs for modelling the M2 layer. This could therefore go on forever as M3 layer is also a model, but OMG has decided that MOF language can be defined using its own constructs.

The term metadata is commonly explained as information about data. In [12] authors distinguish between data and information where the latter is the extra information you need in order to understand the former. In MDA context, metadata refers to the information
captured by the MOF information architecture. It should be noted that in the Artificial Intelligence (AI) community, metadata is used to refer to other available information in addition to what is captured by the different metalayers here, such as who registered the information or authorised it as explained in [21].

**MOF model interchange - XMI**

According to [26], MOF was born out of the need to manage model metadata in standardized way, in order to enable interchange between different software tools. The lack of standards for how to store software models caused tool vendors to store metadata in proprietary ways, causing lock-in situation for the users. Different tools from same vendor even used different ways of storing model information.

Together with the standard way of defining models, OMG has also defined set of standards for how to access and manage MOF compliant models. XML metadata interchange (XMI) [54], is an OMG standard defining how to share MOF model elements using XML. XMI is typically used for exchanging models between tools in addition to playing a central role during model transformation. According to [54], XMI can be used to exchange information at all layers in the MOF architecture.

**Dynamic systems – specifications not captured by models**

It should be noted that models do not always capture all information about an information system. The term dynamic systems, is here used to denote systems that are dynamically configured according to user’s preferences when installed at users site. Large enterprise systems such as SAP are examples of dynamic systems. The system provides support for various functions and different ways in how to structure the information captured and processed by the system. The general structure and behaviour of system components is described using models but the different implementations based in different user’s preferences are specified by different parameter values.

### 2.3 Interoperability - non functional aspects

As previously mentioned, different quality characteristics for software applications are defined by ISO/IEC 9126 – 1991, ref. [11]. Functionality is just one of few different aspects that information systems might need to address. These characteristics serve as a checklist when software requirements are specified. The criticality of the software determines the applicability of these quality aspects. Software supporting business critical or safety-critical processes has different requirements to quality, than software supporting less critical processes. The following table gives an overview of these different non-functional characteristics, as summarised in [37].

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Usability</strong></td>
<td>Addresses human factors; aesthetics, ease of learning, and ease of use as well as consistency in the user interfaces, user documentation and training materials.</td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
<td>Addresses frequency and severity of failure,</td>
</tr>
</tbody>
</table>
recoverability, predictability, and accuracy.

| **Performance** | Imposes conditions on certain tasks or services, e.g. transaction rate, speed, availability, accuracy, response time, recovery time and memory usage. |
| **Supportability** | Addresses aspects needed in order to keep the system up to date after its releases, such as testability and maintainability. |

Even though interoperability has been characterised by [11] as a functional aspect, some other non-functional aspects need to be considered as well. When enterprises are to interact e.g. by using each other services, exchanging information about response time or availability can be very relevant. This introduces yet another dimension that needs to be considered when addressing interoperability.

### 2.4 Information – functional and non-functional aspects

In this section, I look into the different information aspects needed in order for enterprises to exchange information in a B2B or e-commerce setting.

In order to capture functional aspects, we need to specify both data and their structure as well as the processes addressed by enterprises and thereby by their information systems. Model languages used to specify systems, such as UML, reflect this as they distinguish between models addressing behavioural and static view of systems.

In terms of interoperability, exchanging information about non-functional aspects as well as functional aspects can be relevant as pointed out in previous section.

The following figure illustrates the different information aspects needed.

![Figure 5. Different information aspects needed in an e-commerce setting](image)

I have used the term *task* and *process* to reflect the behavioural view as processes are commonly composed of different tasks. I have used the term *metadata* in addition to *data* when reflecting the static view. Use of metadata is to indicate that we also need additional
information captured by the conceptual level in order to understand information exchanged. I then have used the abbreviation NFA to denote the non-functional aspects.

In this thesis, I will mainly focus on semantics of information structure, as opposed to messages (behaviour) and non-functional aspects.

2.5 Mapping arrangements

In order to deal with the heterogeneity involved in information interchange, whether it is different representation languages used for modelling or different ways of capturing real-world entities, some kind of mapping or translation is needed. By mapping I mean the methods, techniques and mechanisms necessary in order to enable transformation from one way of representing information to another.

2.5.1 Mapping – many to many

In a situation where different collaborating enterprises, all use proprietary ways of representing information within their systems, the number of mappings needed for them to exchange information will be exponential in terms of number of enterprises involved. Each pair of involved partners, needs translation mechanism to and from their information sources, resulting in a situation that is sometimes referred to as star arrangement. This situation is complex both because each time a new partner wants to join the network a large number of different mappings need to be created, but also because mappings and specially mappings addressing semantic issues are very complex and time consuming to create. Autonomy and distribution of information sources, in addition to heterogeneity, contributes to this complexity.

A star arrangement or situation where all pair of collaborating partners need point to point mapping in order to seamlessly interact is illustrated by the following figure.

![Figure 6. Star arrangement – many to many mapping](image)
2.5.2 Standardisation – one to one mapping

This complex situation described in previous section can be simplified by mutual agreement or standardisation efforts. If different partners can agree to adhere to certain way of representing the information needed, number of translations is reduced from exponential to linear. Each partner needs to provide two ways translation between his local or proprietary way of representing information and the format the different partners have agreed upon. This situation is explained by the following figure, here illustrating collaboration where all partners have agreed upon one single standard.

![Standardisation Diagram](image)

The figure above might indicate that one unified standard is or will be available addressing all aspects needed. That is of course not the situation I intend to illustrate. Standards evolve through different stages and early in the process and even later, there are often many standards that users need to relate to. Evolvement of standards is e.g. explained by Hanseth et. al. [27]. What standards do in terms of IT interchange is that they provide foundation needed in order to create mappings from some proprietary format to some standardised or shared format. When standards have stabilised, IT solutions can be designed to capture the knowledge contained in the standard and thereby provide solutions that aid in translating from and to the standards, but of course only addressing the aspects that have been standardised.
3. Problem example – information conflicts

In this chapter I will present a problem example. This problem example will serve as a context when identifying various information conflicts. Identifying and analysing different conflicts is also necessary in order to identify requirements that can serve as a basis for later comparing different approaches to semantic mapping. The reference problem is defined by specifying two different examples that both address the same scope but in different ways. This is done to provoke information conflicts and by that, creating a realistic situation that is commonly faced by enterprises wanting to interact.

The term *information conflicts*, is here used to denote the kinds of problem that emerge when different actors use different mechanism in terms of e.g. label names or data types when capturing the data needed. Information conflicts appear when the information being exchanged is analysed as explained in chapter 2.2.1.

The scope for the reference problem is e-Procurement. e-Procurement refers to business-to-business (B2B) purchase and sale of supplies and services where the “e” indicates that purchase and sale is to be done electronically, e.g. over the internet. Considerable efforts are being made in order to provide standards for e-Procurement, as this process involves general and common interaction between different enterprises and therefore becomes a source for information conflicts. The problem example is based on a subset from three different standards that all address e-Procurement. The context for this thesis is model-based development where models are created and maintained and are to drive the development process. The different standards are to represent already existing PIMs, illustrating different ways of addressing similar domain, where the challenge is to relate semantically similar terms in the different models.

I will first introduce the standards and then present the example, illustrating conflicts related to behavioural as well as structural aspects. The main focus in this thesis is on structural aspects. Then I will present some general information conflicts that can be identified in the literature and explain them. Further, I will identify the different information conflicts in the context of examples presented as well as identify the functions needed to resolve them. I round off by discussing my experience gained during the construction of the examples.

3.1 e-Procurement examples specified

The examples used to illustrate conflicts related to behavioural aspects are based on two standards. One is based on the Universal Business Language (UBL) standard [46] and the other example is based on Interchange of Data between Administrations (IDA) standard proposal [30]. The standards are detailed and comprehensive and in order to keep the example at surveyable level, the business processes as described in the standards have been simplified by omitting number of tasks.

The examples used to illustrate different structure of business documents are based on UBL and Compiere [55]. To state that one example is based on Compiere is partly done for
illustration purposes. The example is received from one of the participants in the ATHENA\(^6\) project and it has a strong resemblance to the Compiere interface for business documents although it has not been verified that this is the case. Compiere is an OpenSource enterprise product that is widely used by number of enterprises to support e.g. e-Procurement process, and can therefore be considered as a de facto standard in terms of its external interfaces. In the case of the message structure illustrated, some information elements and classes have been omitted as well in order to simplify.

What should be noted is that even though the models address similar domains and are agreed upon by domain experts, they vary in many ways and therefore cause information conflicts due to different choices regarding how to express real-world concepts.

### 3.1.1 About the standards

The UBL 1.0 standard was officially declared an OASIS standard after six years of continuous work in defining a standard XML-based vocabulary for business documents\(^7\). The working group, contributing to this standard, started as discussion group in 1999 but was formally established as OASIS technical committee in 2001. Membership in the technical committee as well as in OASIS is open to any business but the committee currently consists of 10 different business partners. OASIS is a non-profit organisation that is dedicated to open, public XML standards.

UBL is the first standard implementing the ebXML Core Components Technical Specification (ISO 15000-5), ref. [45]. Underlying data models are mapped to Core Component types where ebXML Core Components serve as a data dictionary.

Interchange of Data between Administrations (IDA) is a Community Programme managed by the European Commission’s Enterprise Directorate General\(^8\). The IDA programme has released the first version of the European Interoperability Framework (EIF) [29], which is to contain a set of standards and guidelines agreed upon by the different member states within European Union (EU). Interoperability is put on the agenda and EIF is recognized as a necessary condition in order to achieve this. One of IDA’s objectives is to connect public IT administration systems across EU in order to support the delivery of public services to enterprises and citizens.

IDA has delivered a working document addressing e-Procurement for the public sector. This document [30], proposes a process as well as a datamodel to be used for ordering and invoicing phases between the public sector as the buyer and an external supplier. The model is strongly influenced by work carried out by others in the same area, including UBL. According to [30], the UBL standard is more complete and more complex.

---

\(^6\) Refer to chaper 7.1

\(^7\) see [http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ubl](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ubl) for information on UBL

Compiere ref. [55], is an open source ERP software application that is widely used by number of enterprises. Compiere provides interfaces in order to import and export business documents such as orders and invoices to and from the system.

### 3.1.2 Business level – behavioural aspects

Figure 8 shows a subset of the activity diagrams defined by the different standards. Both standards address the following functions:

- **Ordering**: The buyer notifies the supplier of his intention to buy products.
- **Fulfilment**: The supplier organises the delivery of the products ordered by the buyer
- **Accounting**: The supplier sends the buyer an invoice corresponding to previously ordered and delivered products.

I have simplified the process such that the order is accepted as it is, without considering scenarios where order is rejected, changed or cancelled. The fulfilment process is simplified by only addressing the dispatching of items and omitting delivery notifications. The accounting process is also simplified by considering the invoice to be correct.

<table>
<thead>
<tr>
<th>UBL – activity diagram</th>
<th>IDA – activity diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="UBL_activity_diagram.png" alt="Activity Diagram" /></td>
<td><img src="IDA_activity_diagram.png" alt="Activity Diagram" /></td>
</tr>
</tbody>
</table>

By viewing the models above, we can see that the tasks used to initiate the information exchange have different names. In UBL, the task for initiating the placement of an order is called *Place Order* while in IDA it is called *Send Purchase Order*.

Assuming that the buyer has adhered to IDA when implementing services for supporting the e-Procurement process and the supplier has adhered to UBL when doing so, the following sequence diagrams better illustrates the conflicts that the two partners encounter when trying to interact. This could be a realistic situation if e.g. some European public sector adheres to
IDA when buying products from external suppliers. Messages and message contents that are named differently are emphasised using bold text. Message contents refer to business documents such as Order. The line in the middle is to illustrate interaction conflicts due to incompatible naming conventions.

![Diagram](image)

Figure 9. Conflicts caused by different naming conventions

### 3.1.3 Message content – structural aspects

Figure 10 on the following page contains a subset of class diagrams from other two standards. Those are UBL and Compiere. The reason for using different standards when looking at the structure of the message contents is that the IDA standard has not been implemented yet. The IDA documentation provides a good description of the intended process for e-Procurement but information about e.g. actual data types used in implementation is not available. On the other hand, Compiere is in use by many organisations and can therefore serve as basis when looking at a de facto standard for common business documents.

The figure serves as basis for comparing the two class diagrams, in order to identify differences as well as similarities. The diagrams presented, reflect the structure of the Order document in the different standards or a content associated with a `placeOrder` message. It should be noted that a large number of information elements have been omitted in order to simplify.
Figure 10. UBL and Compiere – order diagram
3.2 Identifying general information conflicts

Number of different types of conflicts that can occur when exchanging information, have been identified and specified in the literature, but a consensus regarding how to name these different types of conflicts has not been reached.

Khan and Mahmood [34] work with a set of what they refer to as data integration problems when defining a model-based framework for mapping between different models. Ram and Park [62] identify number of common information conflicts, based on analysis of literature in the field of schema integration and refer to these as semantic conflicts. This thesis will analyse the conflicts identified by Missikoff and Taglino [42]. These authors present different information conflicts that where identified when carefully analysing different local data schemas capturing concepts related to the tourism industry in Italy. These conflicts are in [42], referred to as interoperability clashes and are similar to the data integration problems identified in [34], although the labels used to identify them is different in some cases. This thesis will refer to these conflicts as information conflicts.

The conflicts identified in [42] are explained in the following table. It should be noted, that this list must always be considered as a basis to be extended, as analysing data and identifying types of conflicts is somewhat a constant learning process. Any solution addressing these conflicts, should be designed with focus on extensibility as pointed out by Khan and Mahmood [34].

<table>
<thead>
<tr>
<th>ID</th>
<th>Label / Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Encoding</td>
<td>Different units of measure used, typically different weight units.</td>
</tr>
<tr>
<td>2</td>
<td>Typing</td>
<td>Different data types to represent information e.g. string vs. integer.</td>
</tr>
<tr>
<td>3</td>
<td>Naming</td>
<td>Different label used for identical content, also referred to as synonym.</td>
</tr>
<tr>
<td>4</td>
<td>Granularity</td>
<td>Different decomposition of the same content, e.g. name vs. first name, last name.</td>
</tr>
<tr>
<td>5</td>
<td>Content</td>
<td>Different content denoted by the same concept or label, also referred to as homonym.</td>
</tr>
<tr>
<td>6</td>
<td>Coverage</td>
<td>Differences in presence/absence of information.</td>
</tr>
<tr>
<td>7</td>
<td>Precision</td>
<td>Difference in accuracy of information e.g. using different scales for capturing student grades (A,B,C vs. 1..10).</td>
</tr>
<tr>
<td>8</td>
<td>Abstraction</td>
<td>Level of specialisation refinement of the information, e.g. when one schema distinguishes between different types of swimming pools, (indoor, outdoor) and the other does not.</td>
</tr>
</tbody>
</table>
### 3.3 Comparing examples

A glimpse at the different data structures presented in figure 10, reveals that even though the standards are addressing the same context, there are number of differences between them. When having a graphical representation of the structures, it is possible to quickly identify both similarities and differences.

First of all, the label names used to identify the various classes and attributes differ in many cases. Both standards use the same label name to identify the top element, that is *Order*, as well as the *Address* class. In addition, we can identify similar attribute labels such as *description* and *quantity* in both examples. Further, it seems that in many cases, a label for attribute or class in one structure includes a subset of a label in the other structure. An example of this is *BuyerParty* vs. *BuyerInfo*, *Telephone* vs. *Phone* and *Address* vs. *AddressDetails*. In other cases, completely different labels have been chosen to denote somewhat similar content such as *ProductRecord* vs. *LineItem*. We can also see that some attributes that are part of one standard are not addressed at all in the other. An example of this is *BuildingNumber* included in the address information in the UBL standard but omitted from the Compiere standard.

The following table is based on the structural examples in figure 10, as well as the different information conflicts identified in table 2, and gives an example of each type of conflict, identified using the examples provided.

<table>
<thead>
<tr>
<th>Conflict</th>
<th>Description (UBL fields are prefixed with U and Compiere fields are prefixed with C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Encoding</td>
<td>An example of this could be the different units for measurements. The field <em>U.GrossWeightMeasure</em> in the UBL standard is a complex data type that contains decimal value together with a MeasureUnitCode and version id. Simplifying, the field in the example here is a numeric value. If this weight is captured as kilograms, while the value in <em>C.ProductAttributes.Weight</em> is captured using pounds, we have a conflict of type encoding.</td>
</tr>
<tr>
<td>2 Typing</td>
<td>An example of this conflict is e.g. the different date types. The figure does not indicate this clearly but in UBL, the date types are based on XSD date types, where date is on the form: YYYY-MM-DD, while in Compiere, date fields are structural types including three strings, labelled Year, Month and Day. Semantically similar fields of type date in the different standards are <em>U.IssueDate</em> and <em>C.OrderHeader.OrderDate</em>.</td>
</tr>
<tr>
<td>3 Naming</td>
<td>There are number of examples where semantically same data is labelled using different names. If we only look at attribute labels without considering the whole path, we have e.g. <em>C.phone</em> vs. <em>U.Telephone</em> and <em>C.street</em> vs. <em>U.StreetName</em>. In previous mentioned examples, some similarities can be identified as one string contains the other. An example with totally different naming is e.g. <em>C.productUnitaryCost</em> vs. <em>U.BasePrice</em>.</td>
</tr>
<tr>
<td>4 Granularity</td>
<td>An example of this conflict is the street name. In UBL the street name and number is included in one data element,</td>
</tr>
</tbody>
</table>
**5 Content**

The label names in the two standards are quite different and I have therefore not been able to identify concrete example here. This type of conflict typically occurs in combination with conflict of type precision, that is when different numbering system is used for indicating similar information where the different fields have same label name.

**6 Coverage**

A concrete example here is e.g. *U.Address.BuildingNumber*, which is not included in Compiere.

**7 Precision**

A concrete example of this is not contained. However, information about e.g. product colours could serve as an example where one enterprise uses red and green, while other uses more precise coding system for indicating colours of product to be ordered.

**8 Abstraction**

An example of this could be different classification systems used to identify products. The field *U.StandardIdentification* contains a code that refers to standard classification system while semantically similar field, *C.ProductRecord.model* contains a more general product description. A very general example is e.g. table vs. kitchen table or paper vs. recycled paper.

### 3.4 Required functionality for resolving conflicts

In order to enable interchange between enterprises that use different ways of capturing real-world concepts, the different information elements need to be converted from one way of representing, to another. This involves resolving the conflicts already identified. The different kinds of problems will require different functionality e.g. mathematical functions in order to convert between different measurements units used, string functions in order to manipulate with strings that are captured using different granularity, and querying functionality in order to traverse the models.

The following table elaborates on the different conflicts and can be viewed as a first draft, in identifying requirements that some transformation language needs to satisfy. The transformation language needs to express relationship between different model elements as well as to express how to convert from one way of representing data to another way. Transformations are addressed in chapter 6. The table is provided in order to gain a deeper understanding of the problem at hand and is based on the information supplied at this point.

<table>
<thead>
<tr>
<th>Conflict</th>
<th>Functionality needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Encoding</td>
<td>Converting between e.g. different units of measures requires at least common mathematical functions such as multiply, divide, plus and minus. The solution would either enable access to set of predefine functions to choose from where these functions can be initiated from the chosen transformation language or the ability to express such functionality using the transformation language itself.</td>
</tr>
</tbody>
</table>
Typing

Converting between different data types commonly requires some pointer mechanism or type casting. The transformation language needs therefore to either enable such manipulation directly or enable initiation of functions possibly written in other languages.

Naming

Converting between data using different label names for otherwise same data only requires simple data assignment.

Granularity

Converting between data where one structure uses e.g. two data strings to represents one string in the other structure, would require functionality to manipulate with strings. In order to enable transformation in both ways, both substring and merge are needed. In cases where this applies to e.g. integer values where one structure contains sum of values in the other structure, mathematical functions would be required.

Content

This type of conflict does not require special functionality in conjunction with converting between data. On the other hand, ability to identify such relations between data elements in different structures in order to prevent transformation would be required.

Coverage

This type of conflict does not require special functionality in the transformation languages but rather an agreement between different partners regarding how to resolve this. If information is contained in one structure, but omitted in the other, the different partners need to find out whether the data is necessary in order for them to interact. If so, the structure omitting the information might need to be changed.

Precision

This type of conflict requires som arbitrary mapping to be made, based on some agreement between different partners. If the target structure is a standard, the partner responsible for the source data needs to define a standard way of mapping from and to his way of representing the information, to the standard. This would typically require logical constructs such as IF-THEN-ELSE or CASE together with data assignment.

Abstraction

This conflicts also as an issue to agreement between the different partners, reaching a decision regarding whether the specialisation lacking on one side is necessary in order to conduct business.

The identified functionality is basically functionality that is provided by typical programming languages, that is data assignment, string manipulation, mathematical expressions and logical constructs. An exception is type casting needed to convert between different data types, which might be a less common functionality. The languages needed to solve the conflicts need either to have this expressiveness or enable initiation of external functions written in languages that do so.

3.5 Experience gained during example construction and analysis

The process of constructing the previous example, especially the data structure of the order documents has bee incredibly time consuming. I could have constructed examples that reflected my choice of organising relevant information but because I am looking into ontologies that can be viewed as kind of standards, I wanted to create an example based on
already existing standards. This because I wanted to illustrate the comprehensiveness of the standards as well as illustrating that different standards, developed autonomously, will vary in many ways despite of addressing the same or similar domain and despite of reflecting a consensus reached by some community.

Originally I worked with Khan and Mahmood, authors of [34], in order to construct two conceptually different examples to be used to illustrate information conflicts. First we analysed Compiere and UBL but as the information related to Compiere was not easily available, we decided to look into IDA and UBL instead. It required a lot of work to gain an understanding of how, specially the UBL standard, was documented and where to find descriptions of the different fields in order to understand their content. That is important in order to select the most relevant data elements from the standard. We ended up with a structure that was a subset of UBL and IDA, reflecting both the order message as well as the invoice message.

During work on my thesis, I decided to go back to Compiere instead of IDA in order to be in line with the work done in the ATHENA project. I had the opportunity, later in the process, to follow up on the work being done there, where they where using UBL and structure with strong resemblance to Compiere, to construct examples similar to the ones I was trying to accomplish. With the hope of being able to benefit from the work in the ATHENA project, as well as being able to contribute to that work, I decided to change my examples.

Anyway, through the effort of trying to understand the standards as well as trying to create examples, I have made the following observations:

♦ Standards are huge, as they include information elements that are to address all kinds of various information needs. In order to be able to select the relevant information, one needs to spend a lot of time on establishing an understanding of the standard.
♦ Based on my own experience of constructing the examples, I can conclude that trying to understand, and map between, different ways of representing the same or similar information is very challenging, time consuming and error-prone.
4. Requirements – semantic mapping

In this chapter I will elaborate on and make an effort to define a set of requirements that need to be fulfilled in order to establish semantic mapping. This is of course not to be considered an exhaustive list, but a draft that is to be extended as well as being subject to change. I will identify the requirements based on previous discussion as well as my experience of working with system development and integration. These requirements address semantic mapping or the tools, techniques, methods and expression mechanisms needed in order to capture semantic relationships between data sources as well as the techniques needed to convert between them.

The focus is the languages needed to express relationships between terms as well as some more general issues. I will include all requirements I consider relevant, thereby somewhat going beyond the scope of the thesis. Keep in mind that according to Stumptner et al. [70], “it is well accepted that, like database design, system integration is an engineering task which cannot be fully automated”.

4.1 Functional requirements

Functional requirements can be placed on language needed in the process of mapping between two models. The information that is going to be exchanged is captured by some expression mechanism, typically a modelling language. In addition, requirements can be identified towards a language that enables transforming from one way of presenting information to another. I will refer to the former as modelling language and the latter as transformation language.

The information architecture as defined by MOF provides important information about the semantics of the information being exchanged. The data instances are of course the central elements. In addition, the information about how the data is structured as well as the information about which modelling constructs are used to capture the different concepts, are apparently needed in order to resolve semantic conflicts. This information is referred to as metadata and needs to be available in the process.

Regarding a transformation language, table 4 is the source for identifying requirements towards such a language. The transformation language needs to be able to express relationship between the related elements in different models as well as to provide functionality needed to resolve the different information conflicts identified in chapter 3.2.

The functional requirements I am able to identify are summarised in the following table. I use FR as short for functional requirements, ML as short for modelling language and TL as short for transformation language.
Table 5. Semantic mapping – Functional requirements

<table>
<thead>
<tr>
<th>Requirement-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR1-ML-Metadata</td>
<td>Information about how the concepts are structured, needs to be available during semantic mapping.</td>
</tr>
<tr>
<td>FR2-TL-StringFunctions</td>
<td>String manipulation, such as substring and concat for resolving conflicts of type <em>Granularity</em>.</td>
</tr>
<tr>
<td>FR3-TL-Math</td>
<td>Mathematical functions for resolving conflicts of type <em>Encoding</em>.</td>
</tr>
<tr>
<td>FR4-TL-Logic</td>
<td>Logical constructs for resolving conflicts of type <em>Encoding</em>.</td>
</tr>
<tr>
<td>FR5-TL-TypeCasting</td>
<td>Type casting or direct memory addressing for resolving conflicts of type <em>Typing</em>.</td>
</tr>
<tr>
<td>FR6-TL-DataAssignment</td>
<td>Data assignment is generally needed, and the only functionality needed to resolve conflicts of type <em>Naming</em>.</td>
</tr>
<tr>
<td>FR7-TL-SemanticRelations</td>
<td>A transformation language needs to capture semantic relationships between model elements, in order to state whether attributes in different models are e.g. synonyms or homonyms addressing conflicts of type <em>Typing</em> and <em>Coverage</em>. Generally, the language needs to express that there is a semantic relationship between model elements, either 1-1, 1-many or many-many. In addition, the languages needs to express that attributes are not semantically related despite having similar name.</td>
</tr>
</tbody>
</table>

4.2 General requirements

The task of integrating semantically similar information with different origin is considered an engineering task involving human interaction, supported by tools, methods, techniques and automated steps. This indicates that human factors need to be considered when identifying requirements related to the integration task, addressing usability issues.

Graphical notation is better suited for humans e.g. in order to get an overview, as well as to serve as discussion basis when number of users need to come to mutual agreement. Akkøk [1] presents research results that support this common belief. Therefore, it will be important to provide graphical notation of the information to be exchanged or the modelling languages as well as when indicating relationships between elements in different models. It is equally important to provide formal textual notation in order for computers to process the information. Textual notation is also important for humans. Although graphical notation is often better suited for gaining an overview, textual notation might be better suited when exploring details. This results in the following requirement:

*Both textual and graphical notation of expression mechanisms involved should be supported.*

Other usability issues that should be considered have to do with ease of learning or utilising already invested knowledge. The usage of IT related technology varies from enterprise to enterprise. While one enterprise might supports its business processes using internally developed systems, other enterprise might support its processes using software from third
party. In today’s business, knowledge is considered an asset. Knowledge is typically an asset that is not quantified in accounting books. This includes e.g. tacit knowledge acquired by employees through their work as well as more explicit knowledge captured by e.g. software models. For enterprises already using certain technology and methods when developing their systems, it will be important to be able to utilise this knowledge when integrating the systems as well, if possible.

In other words, if an enterprise uses a model-based approach for developing their software, and has already acquired knowledge regarding the usage of e.g. MDA and UML, using similar technology for supporting system integration will be important. Also because information interchange can be equally important in inter-enterprise context as well as intra-enterprise context. Similar can be said about knowledge regarding other technologies. Stating a requirement that captures these issues is not straightforward, but it should be emphasised that familiar languages and technologies already used for system construction, should be applied for semantic mapping.

\[\textit{Enterprises should be able to carry out semantic mapping in familiar context and using similar technology as applied during system design and construction.}\]

Another important issue to consider is the ability to establish the right context when carrying out semantic mapping, thereby preparing the seamless interchange at run time. These issues were somewhat addressed in chapter 2. Addressing semantic issues is about addressing the different ways in how real-world concepts are captured. Providing a system view where real-world concepts are isolated from other e.g. technology dependent information is important. This has to do with human factors as platform independent view allows one to omit unrelated information. This results in the following requirement:

\[\textit{A platform independent view of the information being interchanged should be supported.}\]

Based on the experience gained when trying to construct examples in previous chapter, the task of identifying similarities and mapping between them is complex and will therefore require some methodology support. This results in the following:

\[\textit{The tasks involved when establishing semantic mappings between two ways of representing similar information, should be supported by documented method.}\]

The more general requirements I have identified in this chapter are summarised in the following table, using GR as short for General Requirement:
Table 6. Semantic mapping – general requirements

<table>
<thead>
<tr>
<th>Requirement-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR1-Graphical notation</td>
<td>Both textual and graphical notation of expression mechanisms involved should be supported.</td>
</tr>
<tr>
<td>GR2-Textual notation</td>
<td></td>
</tr>
<tr>
<td>GR3-Familiar context and technology</td>
<td>Enterprises should be able to carry out semantic mapping in similar context and using familiar technology as applied during system design and construction.</td>
</tr>
<tr>
<td>GR4-Platform independent view</td>
<td>A platform independent view of the information being interchanged should be supported.</td>
</tr>
<tr>
<td>GR5-Method support</td>
<td>The tasks involved when establishing semantic mappings between two ways of representing similar information, should be supported by a documented method.</td>
</tr>
</tbody>
</table>
5. Ontology and related technology

In this thesis, I investigate ontologies which according to number of sources, ref. [13], [82], [10] is an enabling factor when various integration conflicts including semantic conflicts, are to be addressed. I want to achieve and hopefully provide understanding for what an ontology denotes in the field of Artificial Intelligence (AI) as well as how an ontology is related to relevant concepts within the object-oriented paradigm, in the context of MDA. By MDA context, I mean models that are recognised as evolving artifacts where I focus on PIMs, which hide implementation specific details. I want to try to understand whether or why AI paradigms for expressing ontologies are better suited than e.g. object-oriented paradigms such as UML class diagrams.

First I will take a close look at how the concept is defined within the field of AI using somewhat historical approach. I will discuss the definitions of an ontology in relation to object-oriented modelling approaches, with focus on MDA.

Then I will address the representation paradigms available for building ontologies by looking at the available constructs as well as common languages used for capturing ontologies. I will focus on OWL as a language for modelling ontologies and compare OWL to UML, the language commonly used for capturing software models using object oriented approach.

Further I will survey the literature regarding how ontologies can facilitate interoperability during information interchange, or how ontologies are used when converting between different ways of representing information, addressing semantic mapping.

Finally I will look into mechanism available within the field of AI that facilitate automatic matching of semantically similar data elements.

5.1 Ontology – what is it?

The software industry has been evolving fast over the years, new programming languages appear, methods for designing software evolve, databases are constructed in different ways and so on. The changes that have taken place within the industry tend to be motivated by a need to solve a certain problem or reach new goals. For instance, the introduction of object oriented programming languages is motivated by the need to localise changes, making it easier to add functionality to existing applications.

When looking for solutions to conventional software challenges, the intellectuals within philosophy seem to be repeatedly consulted. That is quite natural, as software systems represent something that exists in the real-world and philosophy is about investigating principles of reality. I assume that this is not any different within the field of Artificial Intelligence. Today’s knowledge based applications are probably not designed in the same ways as they used to be. The reason for introducing ontologies must therefore be motivated by a need to reach a certain goal or solve a certain problem.
If I can identify and understand the problem being solved as well as locating the origin of the solution chosen, I might be able to better understand what an ontology really is.

### 5.1.1 Ontology’s origin in philosophy – ontological categories

The term ontology has its origin within philosophy and has to do with the subject of existence. In Webster’s dictionary, the following definition of the term ontology is found:

"a particular theory about the nature of being or the kinds of existents”.

The philosopher Aristoles, author of Metaphysics, studied different modes of being. One of the approaches he used was through what may be said or predicated about anything that exists in this world. According to [24], these studies might very well have been referred to as ontology.

These different categories related to what can be said about things are the basis for what is referred to as semantic structure. **Semantic structure** is the system that is used to describe relations between things that exists and the mechanism used to represent them, whether it is a natural languages or other **representation mechanism** used to capture real-world concepts such as object oriented modelling paradigms.

"Aristotle began his study of knowledge representation with an analysis of the semantic categories and relationships expressed in natural language. Those same categories correspond to the types or classes that are encoded in the latest AI languages and object-oriented systems. ” [69] (p.178)

According to [24], Aristole’s categories where provoked by philosopher Emmanuel Kant. He stated that things that exist are not only determined by the things themselves. The perceiver needs to be accounted for, as things can be perceived differently by different subjects.

Adhering to a world view where the world is as perceived according to the subject, is referred to as epistemological realism while adhering to a world view where one believes that the essence of things that exist can be defined in objective manner, is referred to as metaphysical realism [7].

Dahlbom and Mathiassen [12] also mention the relation between Aristole’s investigations and the constructs used when applying object-oriented approach to system design. They explain how Aristole’s ideas are the foundation for constructional approach to system design while Plato’s ideas are the foundation for more evolution or prototype approach. The philosopher Plato stated that things can’t be defined through categorization. One needs to have a prototype or instance of things used for comparison in order to determine if other things can be considered similar enough and therefore regarded as the same thing.

Another way to explain what is out there in the world, is to explain the different layers of reality, or three ontological strata of reality, as discussed by Poli [59]. These different layers are the material-, psychological- and social-layer, where the material layer is at the bottom and is the barrier for the psychological layer, meaning that concepts in the material layer need to be defined in order to express concepts or things that exist in the psychological layer.
In similar way, concepts in material and psychological layer are necessary in order to explain concepts in the social layer, which is the top layer.

5.1.2 Why did the AI-community borrow the concept?

The reason for the terms appearance within the field of Artificial Intelligence seems to be found in an article written in 1991, “Enabling technology for knowledge sharing” [44]. Being able to reuse and share previous design work is the motivation for borrowing the term from philosophy.

“If we could develop shared sets of explicitly defined terminology, sometimes called ontologies, we could begin to remove some of the arbitrary differences at the knowledge level. Furthermore, shared ontologies could provide a basis for packaging knowledge modules – describing the contents or services that are offered and their ontological commitments in a composable, reusable form.” [44]

Neches and associates envision a new way of designing knowledge based systems. Large portion of the time used in designing and implementing such systems previously has been put into designing and constructing new complex knowledge bases from scratch. Building an ontology is one of the initiatives needed to be able to reuse some of that work. The new systems are to be constructed from a library of the following reusable components:

- Ontologies – by selecting one or more
- Reasoning Modules – by selecting zero or more
- Component KR systems - by selecting one

These reusable components are not explained in detail in the article but one of the purposes of this chapter on ontology is to provide understanding for what these reusable components refer to. The following figure copied from [44], illustrating the envisioned library of reusable components, should give some indication to begin with.

![Figure 11. Library of reusable knowledge-based software components](image-url)
5.1.3 A look at the problem to be solved – previous knowledge bases

In order to better explain or understand the gap that an ontology is to fill, we can look at expert systems, systems that perform reasoning by means of formal logic. Those are also referred to as knowledge based systems as the knowledge base is the foundation for expert systems. They capture knowledge and experience of e.g. domain experts, in what is called the knowledge base. On top of the knowledge base is the inference engine, completing the structure of an expert system. The knowledge base consists of both rules and facts. This is presented by the following illustration from [68].

![Figure 12. Structure of expert systems](image)

Prolog, a popular language used for writing AI systems appeared around 1970. Prolog is referred to as a 5th generation language, and as such, specifies what to do rather than how to do it. Prolog stands for “programming in logic” and uses symbolic logic as programming language. The following illustration from [39], shows example of prolog code, and can be considered as knowledge base.

```
parent(X,Y):=-father(X,Y).
parent(X,Y):=-mother(X,Y).
grandparent(X,Z):-parent(X,Y),parent(Y,Z).
ancestor(X,Z):-parent(X,Z).
ancestor(X,Z):-parent(X,Y), ancestor(Y,Z).
sibling(X,Y):=-mother(M,X), mother(M,Y),
father(F,X), father(F,Y), X\=Y.
cousin(X,Y):=-parent(U,X),parent(V,Y),sibling(U,V).

father(albert, jeffrey).
mother(alice, jeffrey).
father(albert, george).
mother(alice, george).
father(john, mary).
mother(sue, mary).
father(george, cindy).
mother(mary, cindy).
father(george, victor).
mother(mary, victor).

:- ancestor(X, cindy), sibling(X, jeffrey).
```

![Figure 13. Prolog code, an example of knowledge base](image)

All the lines in the example, except the last one are involved in creating the foundation that later can be used to answer questions or solve problems. Those lines can be, according to MacLennan [39], thought of as the knowledge base. Asking questions in prolog is referred to as creating goal statements. The last statement in the example above is such a goal
statement and would be answered by using the inference engine. This statement asks if there is an individual who is an ancestor of Cindy and a sibling of Jeffrey.

Realizing the vision described by Neches et. al. [44], indicates that instead of constructing rules and facts from scratch as done in previous Prolog example, the process of building knowledge-based system will be done by assembling reusable components.

5.1.4 Ontology as explicit conceptualisation – the most common definition

Many different definitions have been used to capture meaning of ontologies. In appendix A, I address few of these different definitions. Here I will look at the most common one that is used in the AI literature, that is:

\emph{Ontology is a formal, explicit specification of shared conceptualisation.}

According to Góméz-Péres et. al. [24], this definition is a slightly revised version of Gruber’s definition appearing in [25] where he defined an ontology as following:

\emph{An ontology is an explicit specification of a conceptualisation.}

The most common definition can be explained by looking at the different terms used in the definition.

Being a \emph{conceptualisation} means that an ontology is a result of a process where real-world phenomena are abstracted creating a model that identifies relevant concepts related to the phenomena being modelled, that is the conceptual model. The models presented in chapter 3.1.3 are examples of such conceptualisations.

Being \emph{shared} requires that the conceptual model is to reflect a consensus reach by members of given community. In other words, different members agree to adhere to certain specification or view of the world being modelled. In chapter 3.1.3, two different models (subset) of similar domain are presented. Both models presented are shared, as they are agreed upon by members of given community, one by the users of Compiere and the other by business partners that are members of the technical committee defining the UBL standard.

The specification needs to be \emph{formal} which means that its constructs are formally defined and are therefore readable by computers. The examples presented in chapter 3.1.3 are explained using UML. Some critics have been expressed towards UML stating that UML does not have formal semantics as its semantics are defined using natural languages. The OMG initiative, introducing MOF as a formal language for defining models, addresses this. I can therefore say that if the domain models are created using MOF defined languages, they are indeed formal.

An \emph{explicit} specification as opposed to implicit or tacit, means that the conceptual model is to be precisely and clearly expressed. As mentioned previously, the different phenomena such as person have been implicitly expressed in earlier knowledge bases as being the subject of the facts. The models presented in chapter 3.1.3 are examples of such explicit specification as they clearly and precisely express the real-world phenomena being modelled and do not contain irrelevant information such as implementation details.
5.2 Comparing to IT perspectives – ontological categories

Having looked into what ontologies represent within the field of AI, I will now discuss related concepts within the conventional information science field.

Today’s information science already provides many of the foundation for knowledge management, as it has long tradition for organising and presenting human knowledge, according to Jurisicia et. al. [33]. Authors classify the concepts used for knowledge representation in IT systems into four ontological categories:

♦ **Static ontologies** – addressing things that exist as well as their attributes and relations between them.

♦ **Dynamic ontologies** – addressing behavioural aspects of the world captured in terms of states, state transitions and tasks and processes.

♦ **Intentional ontologies** - addresses things that agents believe in, what they want prove, disprove or reason about, captured by terms such as issue, goal, supports, denies, subgoalOf, agent etc.

♦ **Social ontologies** - cover social settings, organizational structures or shifting networks of alliances and interdependencies, captured by terms such as actor, position, role, authority, commitment etc.

I will here mainly address the first category as static knowledge is the main concern of the theses. I will include some discussion regarding the second one as well. The other two categories are also very relevant in terms of interoperability but will need to be saved for later research. According to [33], social ontologies are relevant when capturing business processes in terms of modelling roles and authorities. Intentional ontologies are also relevant in terms of enterprise interoperability, as concepts related to this category have been used for modelling non-functional aspects.

5.2.1 IT domain models – static ontologies

If I focus on Gruber’s definition where an ontology is defined as conceptualisation [25], the similarities between ontologies and object-oriented models, suddenly become obvious. An ontology as well as IT models capturing static structure are conceptualisations.

Conceptualisation or conceptual models, where real-world concepts are abstracted in order to represent them within information systems, can include all kinds of concepts but it will at least address what has been referred to as entity modelling or static knowledge. Entity modelling is about capturing concrete or abstract things that exist as well as attributes and relations among them. The following figure is to illustrate how the concepts related to an order document, are typically captured in terms of classes and their relations using a class diagram.
Mylopoulos and Levesque [43] confirm this when they refer to the activity of capturing real-world concepts as conceptual modelling, which according to them is a common goal that needs to be addressed by the AI field as well as the conventional software development field. Designers of expert systems are involved in conceptual modelling when building knowledge bases that represent knowledge about an enterprise. Today’s database design methodologies require modelling enterprises using conceptual models, before addressing detailed logical and physical design of databases.

This analogy is also acknowledged by Gómez-Pérez et al. [24] who state that software developers actually create ontologies when they design their domain models. But in [24], authors distinguish between IT domain models and so called heavy weight ontologies built by the AI community. The difference between these is the extract use of axioms in heavy weight ontologies. UML models accompanied with Object Constraint Language (OCL)\textsuperscript{9} can be, according to [24], considered as heavy weight ontologies if the domain models are agreed upon by a community and not created in the context of single enterprise or system. Still, they express their concerns regarding that OCL semantics are expressed using natural languages and therefore not formal as well as not being widely accepted. Latest version of OCL [53] now included as part of UML 2.0 and sharing the same metamodel somewhat addresses this informality.

Jsowa [69] also states in his writing that both programmers and philosophers build ontologies although using different approach, that is programmers use bottom up, while philosophers use top down. Programmers build the concepts needed for certain application but in order to share knowledge, the top-down approach is needed and that "philosophy provides the framework: it’s guidelines and top-level categories form the superstructure that can relate the details of the lower-level projects.” [69] (p.53).

At this point I can conclude that explicitly defined IT domain models representing static knowledge can be considered as ontology. Then I am referring to PIMs as defined in the context of MDA where models are used and maintained throughout the whole lifecycle of an IT product and implementation details are abstracted away. The main difference is that ontologies are to be agreed upon by a community in order to be considered as shared. IT systems used within single enterprise do therefore not satisfy the most common ontology definition used (recall 5.1.4). Still I might want to distinguish between IT solutions that are

\textsuperscript{9} OCL is explained in chapter 6.6
designed and used within single enterprise and IT solutions that are designed to be used by many different enterprises. Many large IT applications are sold to and used by different enterprises and I want to state that the conceptual model represented within these solutions can be considered as shared as they are implicitly agreed upon by all the different users that actually decide to use these solutions. An example of such solution is Compiere (recall 3.1.1).

Some literature addressing ontologies seems to accept that conceptual models related to single enterprise can be referred to as ontology, even though the conceptualisation can not be considered as shared. Wache et. al. [79] distinguish between the two by using local ontologies vs. global ontologies as where local is to indicate that the ontology reflects the view of single party rather than a community. I will therefore use the term local ontology to refer to conceptual models that are defined in the context of single enterprise or single application.

In many previous as well as current IT solutions, the conceptual model is implicitly contained as part of the system. Database schema does not directly reflect the conceptual model as explained by Koren [36]. The conceptual model is the framework used for structuring classes in terms of tables and records in the database, without being explicitly maintained as such. Object-oriented languages provide mapping between classes that are represented in databases but the conceptual model is implicitly contained within the programming code or somewhat hidden within the implementation details. An application specified using MDA (recall 2.2.4) recognises the importance of models. According to MDA, models are to be maintained throughout the lifecycle of an IT product, and by doing so we are at the same time maintaining explicitly defined conceptual model in terms of the PIMs.

One of the motivations behind MDA is to relief the developer from concentrating too much on implementation details. The PSMs are to be automatically generated based on the PIMs and this is enabled by more or less standard platforms available for implementing today’s distributed systems. The same argument, that is higher abstraction level where implementation details are abstracted away is also used to encourage construction of ontologies.

“Ontologies enable the developer to concentrate on the structure on the domain and the task at hand and protects him from being bothered too much by implementation details.” [24] (p. 2)

5.2.2 IT behavioural models – dynamic ontologies

I have now concluded that IT domain models representing static knowledge can be considered as ontology, provided that models have been agreed upon by a community. What is not so clear is whether ontologies as adopted by the AI community are to address dynamic behaviour of a system as well as the entities and inter-relations of domain concepts. Although this is not the main concern of the thesis, I want to include some discussion regarding these aspects.
As discussed in chapter 2.4, in order to achieve seamless interoperability, information about tasks and processes needs to be understood as well as the structure and semantics of the data being exchanged. In the examples provided in chapter 3.1.2 I illustrate how the dynamic behaviour of a system, or its supported processes and tasks, can cause information conflicts. In the e-Procurement example, the task needed to place an order is referred to as “Place order” on one side and “Send Purchase Order” on the other side. The dynamic behaviour needs therefore also to be addressed in order to establish semantic agreement.

According to [24], ontologies do not address dynamic behaviour, only static domain knowledge. Gómez-Pérez et al. explain that PSMs (Problem Solving Methods) capture the processes. PSM defines a way to achieve a task where tasks may have inputs and outputs and can be decomposed into subtasks. PSM also specifies the information flow between subtasks. An important aspect of the PSM is its method ontology. According to [24], the problem of information could be solved by providing mappings between domain ontologies and its method ontology. Even though I have not been able to analyse further what PSM is all about, it is obvious that ontologies are to play important role in terms of modelling behavioural aspects e.g. by defining the method ontology or the task ontology.

Even though the IT literature used as reference for this thesis seems to agree that conceptual modelling does at least address modelling of entities and their relationships, there are exceptions to this. Avison and Fitzgerald [4] mention that unfortunately, the term conceptual modelling has been used about entity modelling. They state that conceptual models have activities as elements, not entities.

In spite of previous paragraph, the conclusion I can draw, based on the surveyed literature, is that ontologies are intended to address both static and dynamic aspects of system knowledge. Figure 11 also indicates this when including “generic tasks” as one of the reusable ontology components (recall chapter 5.1.2). Apparently, the constructs and the languages needed to capture static knowledge have received attention prior to other aspects. Just as physical layer of things that exists, is the foundation for the social layer according to the philosophers, the static structure of the knowledge captured by AI or IT systems, is the foundation for being able to capture the dynamic knowledge. The discussion in chapter 5.4, addressing the modelling languages clarifies this better, showing that languages capturing static domain knowledge are more established with regards to web-based ontology languages, than languages capturing dynamic knowledge.

5.3 How to express ontology – reasoning

One of the components in the vision described by Neches et al. [44], are KR components. KR stands for knowledge representation and refers to the languages used for writing ontologies or using AI-terms, the knowledge representation used for capturing knowledge about ontologies.

10 Refer to Appendix B for discussion of different types of ontologies
Languages used for capturing conceptual models such as ontologies, have semantic structures just as natural languages. The semantic structure is complex system and refers to the ways in which the different building blocks of the language can be structured to form a meaning. The building blocks of natural language are the different word forms, such as nouns, verbs and adverbs. I will refer to these building blocks as semantic carriers. The semantic structure encompasses the rules for how these semantic carriers can be combined to express something that has a meaning. In modelling languages we have e.g. classes, attributes and relations as semantic carriers and we can express meaning by adhering to rules of how these can be combined.

In this section I will identify and explain the semantic carriers needed to capture ontologies according to the literature as well as discussing how the semantic carriers decide which conclusions can be drawn from the information captured. I will explain the different language families available and provide an example of how different constructs are implemented in OWL, the ontology language commonly used to describe semantics for web-based interaction. I will also include some discussion regarding how to choose appropriate ontology language according to the literature.

### 5.3.1 Ontologies – semantic carriers and reasoning capabilities

A framework for assessing the expressiveness of suitable languages for building ontologies is presented by Corcho and Gómez Pérez [9]. Here, authors identify the semantic carriers needed in order to express static knowledge about a domain in an ontology as well identifying a set of features needed for each different carrier in order to determine the expressiveness. These features are formulated as questions and serve as basis for the framework presented in the document.

The semantic carriers needed to express static knowledge about a domain according to [9] are listed in the following table:

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concepts</td>
<td>Concepts, also known as classes, are used in broad sense. A concept can be concrete or abstract, capturing real phenomena such as person, task, process etc. Different types of attributes are to be associated with each concept.</td>
</tr>
<tr>
<td>Taxonomies</td>
<td>Taxonomies are used to organise ontological knowledge using different kinds of decomposition such as generalisation/specialisation.</td>
</tr>
<tr>
<td>Relations</td>
<td>Relations to represent interaction between concepts including unary and binary relations.</td>
</tr>
<tr>
<td>Functions</td>
<td>Functions express special kinds of relations such as “Mother of” and “Priced of a used car”. Applying functions, the n-th element of the relationship is unique for the n-1 preceding elements.</td>
</tr>
<tr>
<td>Axioms</td>
<td>Axioms are used to model sentences that are always true. Axioms are included to e.g. define meaning of ontology component, define constraints on values of attributes, specify arguments of relations etc.</td>
</tr>
<tr>
<td>Instances</td>
<td>Instances are used to represent specific elements. Other terms or labels</td>
</tr>
</tbody>
</table>
also used to present information about elements are individuals, facts and claims.

| Production rules | Production rules have an If…Then… structure and are used to express sets of axioms and heuristics that can be used independently from the way they will be used. |

The expressiveness of the language determines its reasoning capabilities. Apparently there is a trade off between what can be said and what can be inferred. Being able to infer is being able to draw conclusion from the information contained in some closed space such as knowledge base. The Prolog example presented in figure 13, contains a small set of facts and rules enough to be able to conclude e.g. whether there is an individual who is an ancestor of Cindy and a sibling of Jeffrey (recall chapter 5.1.3). Inference mechanism is therefore similar to what in conventional IT is referred to as querying. The questions that can be answered obviously depend on what has previously been said.

The following figure from [9] shows the relations between the different semantic carries and the inference mechanism that can be used for reasoning upon the information supplied.

![Evaluation framework](image)

**Figure 15.** Evaluation framework, expressiveness vs. reasoning capabilities

As illustrated with the figure, there is a relation between the semantic carries and the questions that can be asked. The following table explains the different inference mechanisms associated with the available constructs for capturing ontologies.

<table>
<thead>
<tr>
<th>Inference mechanism</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exceptions</td>
<td>As explained in next chapter 5.4.1, AI languages such as OWL can have instances that do not belong to any classes. AI languages also allow instances to be associated to class even if they do not fully satisfy all class properties. Identifying exceptions is identifying instances that have some properties exceptional to the properties of the associated class.</td>
</tr>
<tr>
<td>Automatic</td>
<td>According to [65], classification is about trying to establish correct</td>
</tr>
</tbody>
</table>
**Classifications**

Class for some instance. This is again due to the fact that instances don’t have to belong to any class. The ability to automatically classify is therefore a mechanism that automatically establishes correct class for some instance by comparing instance properties and class properties.

<table>
<thead>
<tr>
<th>Inheritance</th>
<th>Classifying supporting inheritance enables asking questions regarding inherited properties.</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Monotonic, Non monotonic</td>
<td>Functions are monotonic if ( x \leq y ) implies ( f(x) \leq f(y) ). Non monotonic is the other way around, that is ( x \leq y ) implies ( f(x) \geq f(y) ), for all ( x ) and ( y ) in its domain.</td>
</tr>
<tr>
<td>- Simple, Multiple</td>
<td>Multiple inheritance is about allowing a single class to inherit properties from more than one super-class. Inferring, based on simple or multiple inheritance is therefore about asking questions regarding inherited properties of some class, including inherited properties from multiple super-classes, given that the language used to capture ontologies supports such mechanisms.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Execution of Procedures</th>
<th>The ability to execute procedures or functions, receiving an output based on provided input.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint Checking</td>
<td>This inference mechanisms is about asking questions or verifying whether the classified instances, adhere to the constraints placed upon their associated classes. If some relation is restricted to e.g. cardinality 1..4, the inference mechanism can detect whether this constraint is satisfied.</td>
</tr>
</tbody>
</table>

| Reasoning with rules | The prolog example in figure13, contains a set of rules. One example is the following:  
\[
\text{grandparent}(X,Z) :- \text{parent}(X,Y), \text{parent}(Y,Z).
\]  
This rule can be read as follows: IF \( Y \) is parent to \( X \) AND \( Z \) is parent to \( Y \), THEN \( Z \) is grandparent to \( X \). Backward chaining and forward chaining are two opposite mechanisms for how to draw conclusions from set of such rules. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Backward chaining</td>
<td>This mechanism starts looking at the THEN clause of the rule, when drawing conclusions. When THEN clause has been matched, the IF clause becomes the new goal or question that needs answer. In the example above, the inference mechanism would first match ( Z ) as grandparent of ( X ) and the try to determine whether ( Y ) is parent to ( X ) as well as whether ( Z ) is parent to ( Y ).</td>
</tr>
<tr>
<td>- Forward chaining</td>
<td>This mechanism first establishes a true IF clause, and then adds the expressions belonging to THEN clause to list of goals or questions that need to be answered.</td>
</tr>
</tbody>
</table>

### 5.3.2 Languages for constructing ontologies – OWL and RDF

The first AI-based languages specially designed to capture ontologies appeared in the beginning of 1990 [24]. Languages are grouped into language families based on the similarities of their primitives. The underlying KR paradigms for these first languages were first order logic (e.g. KIF), frames combined with first order logic (e.g., CycL, Ontolingua,
OCML and Flogic), and description logic (e.g., LOOM). These different paradigms are what Neeches et al. [44] referred to as Component KR systems in figure 11.

Increasingly available information on the Internet, led to design of ontology languages based on Internet technology. Their syntax is based on existing markup languages, such as HTML and XML, the first ones appearing in 1996. Languages using this paradigms are referred to as ontology markup languages or web-based languages according to [24]. Examples of such languages are SHOE, XOL and OWL.

A survey of the most known ontology languages that have been used is provided by Góméz-Pérez [24], together with detailed explanations and examples. I will here explain OWL, which is referred to as web ontology language. OWL [73] is intended for sharing and publishing ontologies on the web, and thereby enabling machine understandable data to be shared.

OWL has three increasingly expressive sub-languages, OWL Lite, OWL DL and OWL Full, where OWL Full is the most expressive one. OWL Lite supports classification or taxonomy structures and simple constraints. OWL DL makes use of descriptive logic, enabling more advanced constraints to be expressed. According to [73], OWL DL has maximum expressiveness while retaining computational completeness, meaning that all features in OWL DL are computable in finite time. OWL Full has features such as the ability to define class as an instance of another class. With OWL Full, users have more freedom in terms of expressiveness, but in return, no guarantee’s regarding computational completeness.

The architecture of the OWL language can be illustrated by the following figure from [15]:

![Figure 16. OWL as web-based architecture](image)

The family of languages involved when expressing OWL ontologies, are all included in the stack. These are all recommendations of W3C (World Wide Web Consortium). W3C is an international consortium that develops standards and guidelines available for everyone to use in order to facilitate interoperability. Standards and guidelines are published as W3C recommendations.

The languages constituting OWL are explained in following table:

<table>
<thead>
<tr>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>XML (Xtendible Markup Language) is actually a language for describing other languages or so called metalanguage. XML does not have fixed set of primitives like HTML but enables authors to freely define their own primitives. XML imposes certain tree-like structure on XML documents, as well as on documents expressed by languages defined using XML. Documents are well formed when adhering to this structure. OWL is e.g. a language defined using XML and so is XMI</td>
</tr>
</tbody>
</table>
XML Schema provides means for defining the structure and vocabulary of associated XML documents. XML documents that are written according to certain schema, can only contain the elements defined in the related schema document and the elements need to be structured according to the schema. Documents are valid when their structure is according to schema definitions.

RDF (Resource Description Framework) is a general purpose language for representing information on the web [74]. RDF introduces a system of simple triple of the form, object-predicate-subject in order to make statements about web resources. Associating triples results in a graph structure. Resource is anything that has a unique identity. The subject is always a resource explicitly identified by URI (Uniform Resource Identifier) and refers to the thing being described. RDF can be thought of as capturing attributes of resources or relation between resources without having vocabulary for describing the resources. That is done in the RDF schema.

RDF Schema is RDF’s vocabulary description language used to define e.g. classes and properties. RDF Schema is similar to the type system in object-oriented programming languages[74]. RDF Schema has primitives such as subject, namespace, comment and value.

OWL is built on top of RDF and provides the additional primitives necessary in order to express ontologies. OWL has primitives such as class, unionOf, maxCardinality and intersectionOf.

Below is a small example, just to illustrate what OWL syntax can look like as it includes constructs from more than one language.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:owl:Class rdf:ID="PurchaseOrderLine">  
    <owl:Class rdf:ID="PricedLine"/> 
    <owl:subClassOf> 
        <owl:Class rdf:ID="PricedLine"/> 
    </owl:subClassOf> 
    <rdfs:comment rdf:resource="#Amount"> 
        Inherits from Line and contains information related to delivery. 
    </rdfs:comment> 
</owl:Class> 
<owl:ObjectProperty rdf:ID="changesLineValue"> 
    <rdfs:range rdf:resource="#Amount"/> 
    <rdfs:domain rdf:resource="#PricedLine"/> 
</owl:ObjectProperty> 
</rdf:RDF>
```

**Figure 17.** Example of OWL syntax - created by using Protégé\(^\text{11}\) with owl plug-in

\(^{11}\) Protégé can be downloaded from [http://protege.stanford.edu/](http://protege.stanford.edu/)
5.3.3 Choosing a language

Gómez-Pérez et al. [24] strongly recommend using the framework presented in the document: “A Roadmap to Ontology Specification Languages” [9], to help deciding on a suitable ontology language. This is needed because ontology languages vary regarding their expressiveness and reasoning capabilities. The framework includes relevant questions regarding different primitives in order to help in the decision making. An example of such a question is: “Is it possible to define integrity constraints in order to check the correctness of the arguments’ value?” [9] (p.3). This question addresses the arguments of a function or a relation. The recommended framework implies that one needs to have an idea regarding which questions the ontology should be able to answer, or which requirements the inference mechanism needs to satisfy when choosing a language.

Some ontology languages include inference engines as a part of the language. Web-based languages such as OWL do not according to [24]. The language requires that externally constructed engine is used in order to draw conclusions from the information captured. The Jena framework [56], includes an inference engine that can reason with ontologies constructed using OWL. According to [56], the Jena OWL reasoner generates a prototype or instance of some class, and using this prototype, is able to deduce that this instance can be considered as a member of some other class and therefore to be considered as subclass thereof. This inference mechanism would be characterised as automatic classification (recall chapter 5.3.1).

Regarding e-commerce, Gómez-Pérez et al. [24] recommend using languages that provide inference engines. Further they imply that only simple constructs are needed when capturing domain specific knowledge in order to describe the semantics of some domain in an e-commerce setting. The ontology will typically need to address product descriptions as well as related services. On the other hand, they state that the need for reasoning capabilities is usually higher. If number of services or products in some setting are high, the support for e.g. automatic classification of services is very useful.

5.4 Comparing to IT perspectives – OO modelling languages

Based on the discussion in chapter 5.2, I have already concluded that Ontologies and PIMs are to express similar things, that is how real-world concepts are captured and expressed in the world of computers, or what Gruber refers to as conceptualisation in [25]. As UML is the de facto language for capturing such model within the IT community, I use UML in my discussion but it should be noted that other modelling languages might be used to capture domain specific knowledge. Further I assume that the reader is familiar with the model constructs available in UML.

In this section I will make an effort to answer the following questions, drawing conclusions based on previous discussions as well as some additional references.

♦ Is the expressiveness of UML similar to the expressiveness of OWL?
♦ What are the strengths and weaknesses of each paradigm?
5.4.1 Expressiveness OWL vs. UML

Even though I have concluded that the information captured by Ontologies and PIMs is somewhat the same, there is still an important issue that should be noted and I will start this discussion by addressing this issue.

As Sowa [69] points out, programmers do construct ontologies but they use more bottom up approach in the process while philosopher’s use more top down approach. This indicates that even though the goal is similar, that is conceptualisation of certain domain, the scope might be somewhat different. In order to really understand the semantics of some conceptualisation, we might need a bigger picture then the one typically captured by PIMs addressing single domain. However, the concepts used in PIMs are somewhat top level grounded as they are based on the available constructs in the UML language such as role, activity, class etc. These concepts are similar to what is to be expressed in so-called Top Level or Upper Level ontologies\(^\text{12}\). Despite this, following should be noted; in order to capture semantics of some domain, the scope of a PIMs is probably more narrow than the scope of some ontology. The main conclusion is that the UML 2.0 language used to express PIMs is a language that also can be used to express Ontologies.

Back to the expressiveness of OWL and UML. Gómez-Pérez et. al. [24] include discussion regarding whether other paradigms than typical ontology languages from the AI community could be used to express ontologies. They state that UML can only be used to express light-weight ontologies, similar to what can be accomplished using OWL Light. In order to achieve the expressiveness of OWL-DL, UML language needs to be accompanied by OCL [53]. OCL is a predication definition language that can be used together with the UML language in order to define constraints in forms of e.g. axioms [28]. UML does not mandate the use of OCL but provides mechanism for expressing additional constraints on all UML modelling through the use of UML profiles\(^\text{13}\). This mechanism can be applied using any predicate language of choice but OCL is now included as a standard part of UML 2.0, sharing the same metamodel.

Another important source when considering the expressiveness of the different languages, is the document “OWL Full and UML 2.0 Compared”, ref. [28]. In the article, Hart et. al. point out the fundamental differences between the two languages. These are summarised in table 10. In addition to the fundamental difference identified in the table, there are a number of constructs available in OWL-DL and not in UML and vice versa. An example of this is the ability in OWL to state that two classes are equivalent, using the equivalentClass property. It is pointed out by Hart et. al. [28], that these additional constructs can be modelled using the UML language.

\(^{12}\) Different kinds of Ontologies are addressed in appendix B

\(^{13}\) UML profiles are addressed in chapter 6.2
In the following table, the left column is a direct copy of table 11 in the document and the right column includes my supplied explanation that is based on discussions provided in the document.

**Table 10.** OWL features with no UML equivalent

<table>
<thead>
<tr>
<th>In OWL and not in UML</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Thing, global properties, autonomous individual</strong></td>
<td>In OWL, instances as well as some relations (in owl, relations are called properties), can exist without being attached to certain class. This is due to the fact that OWL is based on sets while UML is based on types. Instances and relations in OWL can be a subset of the universal class <em>Thing</em> or binary relation between two <em>Things</em>.</td>
</tr>
<tr>
<td><strong>Class-specific cardinality redefinition</strong></td>
<td>As OWL properties can be declared independent of classes, they can have different cardinality definitions when applied to different classes.</td>
</tr>
<tr>
<td><strong>allValuesFrom, some ValuesFrom</strong></td>
<td>“In OWL, property can have its range restricted when applied to particular class, either that the range is limited to a class (subclass of range if declared) (<strong>allValuesFrom</strong>) or that range must intersect a class (<strong>someValuesFrom</strong>).” [28]</td>
</tr>
<tr>
<td><strong>SymmetricProperty, TransitiveProperty</strong></td>
<td>OWL allows properties to be declared symmetric or transitive. In both cases the domain and range must be type compatible.</td>
</tr>
<tr>
<td><strong>Classes as instances</strong></td>
<td>In UML or MOF defined languages, there is a strict separation of metalevels so that population of M1 classes is distinct from the population of M2 classes. In OWL full, one class can be an instance of another class on the same level, a characteristic inherited form RDF. In OWL DL, this usage is restricted.</td>
</tr>
</tbody>
</table>

As can be determined from previous table, there are fundamental differences in the expressiveness of the two languages e.g. the different ways of declaring relations between classes and the ability for instance to exist without being associated to certain class.

The constructs being compared in the referred article are primarily constructs needed to capture static domain knowledge. According to Corcho et. al. [9], the semantic carriers identified in table 7 are the ones needed to capture static domain knowledge. Other literature surveyed here such as [24] and [25] discuss the semantic carriers, that is classes, relations, functions, taxonomies and axioms, as the constructs necessary to create ontologies, indicating that this is all you need. Apparently, constructs needed in order to capture static knowledge, is what has received attention first when in comes to building ontologies.

The question that needs to be answered is whether the expression mechanisms lacking in UML but included in OWL-Full are necessary, in order to explicitly define the semantics in the area of e-commerce. In addition, it needs to be verified that the additional expression mechanisms available in OWL-DL, can be added to the UML language. The UML language includes extension mechanisms called UML profiles. The usage of UML profiles to support ontology modelling is addressed in chapter 6.2.2.

Despite the main focus on static domain knowledge when constructing ontologies, ontology languages addressing dynamic behaviour within a domain are also available. OWL-S [40], is
a language that is layered and actually built on top of OWL. OWL-S is a knowledge representation paradigm for capturing knowledge needed for describing web services. OWL-S has three main parts; a service profile providing mechanisms needed for advertising and discovering web services, a process model providing constructs for describing service operations in details and the grounding needed for describing how to interoperate with the service via messages [40]. OWL-S was submitted as W3C recommendation in November 2004. After spending a couple of days on comparing the constructs available in OWL-S using [40], to the constructs available in UML 2.0 using [63], I can conclude that there are many similarities between the constructs in OWL-S process model and the constructs used for creating UML-2.0 sequence diagrams.

5.4.2 Complementary advantages OWL vs. UML

In order to understand the strengths and weaknesses of the different languages, without an experience of applying the different languages in practical setting, the literature needs to be consulted.

In an article, “A Model-Driven Semantic Web, Reinforcing Complementary Strengths”, Frankel et. al. indicate that the strength of UML based modelling languages is their wide tool support, including support for graphical representation of the information captured as well as. The strength of the ontology language paradigms on the other hand, is their reasoning capabilities [21].

In a current working document of the ATHENA project\(^{14}\), authors of [8] acknowledge OWL and UML as two alternatives for representing conceptual models or ontologies. They identify the strength and weaknesses of each language and choose OWL as the paradigm to be used, mainly because of the advanced reasoning and query services available as well as being a recent W3C proposal for modelling ontologies. Some of the main positive features of using UML according to [8], is the fact that UML is a widely accepted standard at industrial level for capturing and representing conceptual models. UML is also acknowledged as a language that is very well suited for communication between people while OWL is considered a better choice for machine interaction.

The ontology language that will be used in the ATHENA project to capture their ontology or shared conceptual model is referred to as OPAL. OPAL is an OWL extension and is explained in chapter 7.2.1. It will consist of a general ontology that serves as a top level grounding, including constructs such as Object, Process and Actor taken from UML in order to tailor the language towards domain experts, currently familiar with UML.

Both previous references acknowledge reasoning capabilities as the main strength of ontology languages provided by the AI community. Gómez-Pérez et. al. in [24], also recommend choosing languages that provides good reasoning capabilities as pointed out in chapter 5.3.3. In addition, according to [24], the web-based languages such as OWL do not include inference engines. These need to be built on top of the language. An inference

\(^{14}\) The ATHENA project is discussed in detail in chapter 7.2
engine capable of reasoning with OWL is part of the Jena Framework, ref. [56]. According to [56], the reasoning capabilities provided by the Jena Inference engine are automatic classification where the inference engine analysis instance and tries to deduce, based on their attributes, which class they might belong to.

5.5 Semantic mapping – applying ontologies

In previous sections I have discussed the definition of ontologies as well as discussing how to represent or capture ontologies addressing common language constructs. In this section I am going to look into how to apply or use ontologies.

In order to make use of an ontology when integrating information sources, the ontology needs to already exist. The ontology can be acquired in various ways, e.g. built from scratch, by extending an already existing ontology or by assembling a set of already existing ones into one unified ontology. Despite the approach being used, the process of constructing ontologies is complex and needs to be supported by appropriate methods and tools [24].

In this section I will survey the literature regarding how ontologies can be linked to information systems in order to describe their semantics.

5.5.1 Architecture – different ways of mapping ontologies to information sources

In [79], a survey of existing approaches of ontology-based integration of information, Wache et. al state they found a striking lack of adequate methods for supporting the process of both developing as well as using ontologies or mapping ontologies to existing information sources.

The different approaches for linking ontologies to information sources are illustrated in the following figure from [79]:

![Figure 18. Different ways to use ontologies - architecture](image)

- **Single Ontology approaches**: This approach relates all information sources to one global ontology which provides the shared vocabulary needed for the specification of
semantics. The objects of each information source are related to the global domain model. This approach is appropriate for information sources that all address the same domain with similar granularity. The disadvantage of this approach is the potential complexity involved in making changes to local information sources as this might impose changes to the global ontology as well as its mapping between other information sources. This approach assumes that an explicit conceptual model for the different information sources is not available.

♦ **Multiple Ontology approaches:** Here, all the information sources are described by their own ontologies that presumably use different vocabularies. The advantage is that minimal or no ontology commitment is required, reducing the complexity involved in making changes. In practice this approach is nearly impossible because of heterogeneous vocabularies but have been aided by applying special “mediation agents” that are customized to translate between different ontologies and even different representation languages. Despite the aid from “mediation agents”, this approach is complex because of the different kinds of information conflicts that need to be accounted for.\(^\text{15}\)

♦ **Hybrid Approaches:** In this case, semantics of each data source are described using a local model but the terms of a local ontology are described using primitives of a shared vocabulary that might also be an ontology. The primitives are the basic terms of a certain domain. The way in which this description is done, is what is interesting according to Wache et. al. [79].

### 5.5.2 Approaches for establishing ontology-based mapping

Ontologies are used to explicitly specify the meaning of concepts or using Gruber’s common definition “explicit specification of conceptualisation” [25]. In terms of interoperability, ontologies are connected to information sources in order to provide explicit meaning or to provide a shared agreement regarding how some data shall be represented. How exactly this connection is made varies from approach to approach but the term I use to encompass this connection is *mapping*.

The examples presented by Góméz-Péres [24] do not cover how an ontology can be mapped onto information sources. It is stated that the process of mapping is not covered in current literature, and that considerable research efforts is needed in order to define this process and find adequate relations to capture semantic similarities between models. Similar concerns are expressed by Pinto et. al. [58] and Wache. et. al. [79].

Pinto et. al. [58] claim that the process of linking an ontology to information sources is completely different from the process of merging different ontologies. I would like to state that these approaches have some similarities in cases where explicit conceptual model of the local information source is available. In both cases, relation between semantically similar concepts needs to be established. The difference might be that in case of an ontology-merge, transforming data between the different ways of representing it, is not necessary.

\(^{15}\) Refer to chapter 3.2 for overview of different information conflicts.
During the survey of existing approaches, Wache et al. distinguish between four different ways of connecting ontologies to information sources; structure resemblance, definition of terms, structure enrichment and meta annotation [79]. The approaches they describe seem to assume that a local ontology, or explicit conceptualisation of local sources, is unavailable.

- **Structure Resemblance**: Approach where one to one copy of the structure of the database scheme is made and encoded in a language that makes automatic reasoning possible. The integration is done on the copy of the database structure.

- **Definition of Terms**: This approach attempts to explain concepts in information sources by linking them to the corresponding concepts in the ontology. In this case, the structure of database does not necessarily correspond to the way concepts in the ontology are structured.

- **Structure Enrichment** is the most common approach according to [79]. Here, an effort is made to combine both previous approaches. A conceptual model is built, that represents the structure of the local information source and is then enriched with information that links local concepts to concepts define in some global ontology.

- **Meta Annotation** is used for semi-structured information, where information source is defined using markup languages based on e.g. HTML or XML. Information about semantic structure is inserted into information sources, where local concepts are further defined in terms of web-based ontology languages.

Wache et al. [79] also include a survey of approaches for what they call, an inter-ontology mapping, that is how to establish relations between terms in different ontologies addressing a similar domain. As I have already pointed out, I consider the process of establishing inter-ontology mapping between terms to be similar to the process of linking terms in local ontologies to terms in some global ontology, and therefore it is very relevant to investigate how this is achieved. The different approaches for this, identified in [79], are; Defined Mapping, Lexical Relations, Top Level Grounding and Semantic Correspondence.

- **Defined Mappings**: A common approach, providing users with possibility to define mappings between ontologies. The translation is thereafter done based on the mapping definitions defined, using what the AI community refers to as “special mediator agents”. This approach is flexible according to [79], supporting different kinds of mappings as well as translation between different representation languages. The drawback is that it does not prevent the user from defining arbitrary mappings that even don’t make sense or do not cause conflicts.

- **Lexical Relations**: Instead of the ability to apply user-defined relations or mappings, a set of somewhat semantically grounded relations can be applied. These relations extend common description logic and are borrowed from linguistics. Examples of such relationships are synonyms and overlap. Even though these relations have similar constructs as in description logic, they are not formally grounded resulting in somewhat heuristic sets of relations.

- **Top Level Grounding**: An approach where one has to stay within formal representation language when defining mappings. One way of doing that is to relate concepts from

---

16 Ref. chapter 5.3.2
different ontologies by extending a common superclass in a single top-level ontology. This approach does not establish correct correspondence, and is therefore not sufficient when exact matches are required.

♦ **Semantic correspondences**: An approach that attempts to overcome the drawbacks of the previous approach. In order to avoid arbitrary mapping between concepts, this approach has to rely on common vocabulary.

The latest approach is very interesting as this indicates that a shared vocabulary, which can according to [79] also be a full blown ontology, is needed in order to describe two different ontologies addressing same domain. Here the problem has been escalated up one level. Instead of using common vocabulary to assist in mapping between local ontologies, shared vocabulary is needed to map between two ontologies.

### 5.5.3 Expression mechanism for capturing mapping relations

As can be deduced from the information in previous section, the challenge involved in how to establish mapping between two conceptual models or data sources and ontologies, is an ongoing research process. This indicates that the problem of establishing semantic relationships between similar concepts in two models is not fully solved by agreeing on a shared ontology. The information sources still need to be mapped on to the related terms in the ontology in order to define their semantics.

In [5], Bouquet et. al. propose an extension to OWL, defining a new language, C-OWL for capturing relations between different models. According to Bouquet et.al, the OWL language does not have constructs for relating semantically similar concepts in two different ontologies.

They use *explicit mapping* to refer to when local ontologies are kept local, and mapping is defined against a global ontology by explicitly defining the relationship between semantically similar terms in the two different models. This approach corresponds to what is referred to as **Semantic correspondence** in previous section, given that the shared vocabulary is also an ontology. Authors define four simple constructs to use for expressing one-to-one relationships between two terms. Explicit mapping allows for controlling which terms in the local model are made globally visible. According to [5], “this is only a the first step and a lot of research that remains to be done.” The explicit mapping addresses information hiding, where the core issue is to share only what needs to be shared.

The following is an example of the syntax, where equivalence relationship is expressed between two terms in different ontologies:
5.6 Semantic coordination - automatic mapping

Recent literature addressing ontology acquisition acknowledges that creating global ontologies for a single domain is a very challenging task [17], [41]. The UBL standard for e-Procurement introduced in chapter 2 might be considered a global ontology, based on Gruber’s definition [25] as long as it is expressed in formal language. Constructing such a domain model that is to be agreed upon by a large and distributed community has, in the case of UBL, taken six years as mentioned in chapter 3.1.1. Therefore the focus is changing to also include research regarding how to better support the task of mapping between different local ontologies addressing the same domain as well as how to construct widely accepted domain ontologies. Evolvement of ontologies is therefore travelling a similar path as heterogeneous information systems. Due to distribution, ontologies addressing the same domain are being built by separate partners as mentioned in [24].

Elst and Kiesel [17] present an approach where they, instead of formally inferring relationship, gather evidence for such relationships and present a merging framework for what they call evidence-based ontology merging. As explain earlier, inferring means drawing conclusion from already available information. Instead of acting upon the conclusions being drawn as if it where the truth, users are involved in order to approve or disapprove the suggested relationships. Users merging different ontologies are presented with a list of mapping evidences that have been detected or inferred by using three different mechanisms.

- **Term-based Matching**: In order to determine similarities between strings, a so called n-gram approach is used. Similarities of two match candidates are computed by comparing all substrings of length-n. Degree of fitness is determined by number of similar substrings found.
- **Structure-Based Matching**: An approach where models structures are viewed as two graphs that are compared in order to identify similarities. Initial matching points are established based on user’s input. Similarity of two nodes in the two graphs is determined by the similarity of the adjacent nodes.
- **Instance-Based Matching**: This mechanism exploits similarities between instances of classes or class objects. If instances in different ontologies are similar it is concluded that the class might be similar even if it has a different name.

5.7 Conclusions

The main discussions in this chapter and related conclusions are sketchily summarised in the following points:
Global ontologies reflects a shared view of some conceptualisation while a local ontology reflects how real-world concepts are captured and represented within a single enterprise or single application.

The two languages, OWL and UML, can be viewed as two alternatives when choosing a language paradigm for expressing an ontology.

The future questions that need to be answered, or queries that need to be made, should govern the choice of ontology language.

Mapping between conceptual models that address similar or same domain, is needed in order to achieve interoperability. Such mapping is similar and therefore independent of whether one is mapping between two proprietary conceptual models (local ontologies), local ontology and a global ontology or two global ontologies addressing same domain.

Semantic constructs for expressing conceptual models are available while semantic constructs for capturing mapping or expressing semantic relationships between conceptually different but semantically similar models are less available or lacking.

Methods addressing how to explicitly establish semantic relationships between conceptually different but semantically similar models are lacking.

Reasoning capabilities seem to be the main argument for choosing ontology related technologies. What strikes as odd is that querying mechanism for reasoning with these language seem to be rather limited as web-based ontology languages such as OWL do not include reasoning capabilities. On the other hand, this is a very active research field and the knowledge of different techniques for doing reasoning, is certainly the strength of AI related technology.

---

17 Referring to UML2.0 that includes the OCL language and is defined in MOF compliant way.

18 In other words; different structures and label names for same data.
6. MDA Transformations and related technology

The Model Driven Architecture (MDA) is a framework for software development where models are maintained throughout the life cycle of a system. In MDA, the application logic is to be separated from the underlying architecture. PIMs, in MDA context, can be viewed as conceptual models, capturing domain concepts in a language that is formally defined using MOF (recall chapter 2.2.4). UML is the de facto language used for capturing conceptual models within the software industry and UML 2.0, including OCL as part of the language, is now defined in MOF compliant way. The UML modelling language used can therefore be considered as technology suitable for capturing or representing ontologies, where ontology is defined as "explicit specification of a conceptualisation" [25]. This is one of the conclusions I came to through the discussions in previous chapter. Another conclusion I came to is that; methods, techniques and technologies regarding how to map between two conceptual models or ontologies, is still very much at research stadium.

In this chapter I am going to look further into the MDA framework with focus on transformations. Transformations play a central role in MDA’s software development life cycle and are carried out when transforming PIMs to PSMs. I want to analyse the principles related to transformations in the context of MDA in order to understand and identify mechanisms that could be used when transforming between semantically similar but conceptually different models. In my discussion, I assume that I am to convert between a local model which is to be considered as the Compiere example from figure 10, to some global model or ontology which is to be considered as the UBL example from figure 10.

I will first discuss what transformation is in more detail. Then I will explain the role of MOF and how metadata, or UML profiles and metamodels are used when converting between different abstraction levels or models of the same system, with examples related to ontologies. I will then discuss QVT, the OMG standard for transforming between MOF compliant languages as well as how the task of converting between different ways of structuring the same information, can be viewed in the context of the MDA framework. Finally I will address current tool support available for carrying out MDA transformations.

6.1 What is transformation?

I will explain transformation using the terminology presented in the document: “What is Model Driven Architecture?” In [64], Runde and Stølen describe MDA as a framework for constructing methodologies addressing model driven development as well as presenting a terminology for addressing the concepts involved.

One of the central concepts addressed is model transformation, which captures how one model is converted into another model of the same system. Transformation is illustrated as a relation between input and output models, mapping and mark assignment where marks are additional information that can be assigned to original model elements in order to aid in the transformation process. The transformation might also be done step-wise, mixing automatic and manual steps with an in between results in form of some marked model. The MDA
approach does not explicitly account for this, but marking models are viewed as part of the transformation process as every model in MDA context is either a PIM or PSM. The transformation as n-ary association is illustrated by the following figure from [64]:

![Figure 20. Transformation as an n-ary association](image)

Mapping denotes the rules and techniques to be used in the transformation process as illustrated in the following figure:

![Figure 21. Mapping as set of rules and techniques](image)

Transformation rules basically capture which or under which condition, elements are related while transformation techniques capture how to transform from related elements in input model to related elements in output model.

### 6.2 The role of MOF and M2 layer

Transformations in MDA bridge the gap between different levels of abstractions, that is between platform independent and platform dependent models, where the MOF architecture plays an important role. If a PIMs is an UML model, the related PSMs is either an UML model or instance of some metamodel different from UML. When both PIM and PSM are UML models, the PSMs has been supplemented with implementation specific details, using UML profiles. Converting between different abstraction levels, either by using UML profiles or by using different metamodels, requires that all constructs used, are defined in a MOF compliant way. Recall that M2 layer or metalayer in the MOF architecture, captures the available constructs to use when creating models.
The relation between the MOF architecture and MDA transformations can be illustrated using the following figure. The figure is to show how PIMs are transformed into related PSMs, using a) related profile and metamodel as input or b) related metamodels as input.

![Figure 22. Illustration of the relationship between MOF architecture and MDA transformations](image)

### 6.2.1 Profiles in general

UML profile constitutes the different extension mechanism available for adding constructs to UML without modifying the underlying language.

Profiles can be used in different ways e.g. to tailor the modelling language to specific purpose such as specific domain or specific technical platform, or simply to add marks during model transformation. UML can be extended using three different features [63]:

- **Stereotype** is a model element with same structure and information content as existing elements but with extended meaning. Stereotypes extend existing UML model elements and may be supplemented with additional constraints as well as separate icon to use for their presentation. Stereotype may also define set of tagged values to store properties that are not supported by the base element. When stereotype is applied to model element, the model element gains the additional properties associated with the stereotype.

- **Tagged values**: A user defined attribute that can indicate e.g. project management information or code generation guidance.

Application of stereotype and its associated tagged values is illustrated in the following figure from [63].
**Constraints:** A well-formed condition expressed as a text string in some language, such as programming language, natural language or constraint language such as OCL.

Defining package of additional model elements, which extend existing UML metamodel elements, is actually a mechanism for creating a totally new language based on UML.

### 6.2.2 Ontology UML profile - OUP

In [23], Gasevic et. al. explain how they have tailored UML to use for modelling Ontologies, by adding additional model elements to the language in terms of UML extensions. The extensions they define are organised in a profile they call Ontology UML Profile (OUP). By applying this profile, they can use a graphical UML tool to model ontology. The following is an example from [23], illustrating their approach.

Defining an ontology profile in order to achieve the same expression capability as OWL DL, enables the use of UML modelling tools for constructing ontologies. Note that only OWL DL is supported and not OWL Full which allows classes to be defined as instances of other classes. MOF compliant languages such as UML are not allowed to cross meta-layers as classes belong to M1 level and their instances belong to M0 level (recall chapter 2.2.4). This makes it impossible to express classes with double role, that is both an instance as well as a class, at the same metalayer.
The advantage of supporting OWL DL with an UML profile is that ontologies can be made available in the same modelling languages as the data source that are to be mapped to the ontology, given that data sources are represented with PIMs. In addition, it can be assumed that business experts involved in constructing conceptual models that describe static aspects of information systems, already have knowledge of how to use those tools as well as the modelling language. If some shared conceptualisation is to be agreed upon in order to reduce number of mappings between enterprises, where the local conceptualisations are currently captured using UML/PIMs, constructing the shared conceptualisation using the same modelling language and tools might obviously be an advantage.

On the other hand, if the reasoning capabilities, available for ontologies captured using e.g. OWL are necessary in order to answer future questions, constructing them using UML/PIMs might cause disadvantages. Removing the ability to cross meta-layers, expressing classes as instances of other classes, might also cause disadvantages.

### 6.2.3 Metamodels

Every model in the MDA framework uses constructs that are either defined by its metamodel or by it’s a profile, which is a package of metamodel extensions. This means that e.g. every UML model is an instance of its UML metamodel, where the UML metamodel defines all available constructs as well as the rules for how these can be combined. OMG has standardised two metamodels in addition to the language for defining metamodels or MOF. These metamodels are UML and CWM. UML is the de facto language for modelling software applications and the latest version UML 2.0, is now defined in MOF compliant way and includes OCL as part of the language. CWM is a metamodel directed towards modelling warehousing applications. As both UML and CWM models are defined in MOF compliant way, these models can be both source and target for MDA transformations.

OMG has issued Request for Proposal (RFP) for several other metamodels. OMG standardisation process evolves around RFP where the RFP document serves as a requirement specification for the metamodel to be proposed. Two of current RFP’s are important for context of this thesis. The first one is Ontology Definition Metamodel RFP issued in March 2003, ref. [50] and the second one is MOF 2.0 Query/View/Transformation RFP, issued in October 2002, ref. [51].

The Ontology Definition Metamodel (ODM) is to consist of set of metamodels and profiles that together enable modelling ontologies using MDA related technology. The following is to be supported according to [6]:

- Development of ontologies using UML modelling tools.
- Implementation of ontologies in the W3C Web Ontology Language (OWL).
- Forward and reverse engineering of ontologies.

The MOF 2.0 Query/View/Transformation RFP (QVT) states requirements toward a model-based transformation language together with languages for creating views and writing queries. This means that the constructs available when writing transformations are defined using a metamodel.
ODM is further addressed in next section and QVT is addressed in more detail in chapter 6.4.

### 6.2.4 Ontology Definition Metamodel - ODM

In [16], Djuric, Gasevic and Devedzic, describe an architecture enabling transformation between OWL and UML by means of Ontology Definition Metamodel (ODM). Authors have defined a metamodel that is in accordance with OMG’s RFP for such a language and the name ODM, is therefore used accordingly. Their approach requires set of several specifications, and is illustrated by the following figure:

![Figure 25. Ontology modelling in the context of MDA and Semantic Web](image)

ODM is about defining the modelling constructs available in e.g. OWL as a separate MOF compliant modelling language. Previously described approach, using UML profile, is an alternative to defining separate ontology language as a metamodel independent of UML. Defining ODM simplifies transformations between ontologies captured using OWL into ontologies captured using MOF compliant models, as the ODM modelling constructs are directly targeted towards representing certain language e.g. OWL. In similar way as PSMs typically provides one to one mapping to constructs in e.g. java language, the ODM contains model elements that have one to one mapping to constructs provided in OWL.

As pointed out by [16], current tool support for UML models has concentrated on the graphical representation of the M1 layers or the models. This situation is changing rapidly as large tool suppliers are paying more attention to supporting MOF compliant modelling languages, ref. [26]. Various tools, supporting conventional software development disciplines, such as change and configuration management, can become accessible for ontology evolution through the application of the MDA technology and MOF compliant repositories. As pointed out in [24], current methodologies supporting ontological
engineering, focuses mainly on development activities and pay less attention to activities related to management and evolution.

In order to benefit from graphical notation using UML tools, the UML profile is needed as well as transformation between ODM and OUP as current tools primarily support graphical notation of UML. According to [16], the approach described enables using ontologies in a way closer to software engineering.

6.3 Semantic mapping – aligned with the MDA framework

How to view transformation between two models capturing similar real-world concepts in different ways in the context of MDA, is not so clear.

One issue is that in MDA framework, the main focus is on transforming between different abstraction levels of the same system or what can be considered as vertical transformation. When mapping between two conceptually different models, the transformation is horizontal meaning that mapping is to be carried out between two models at the same abstraction level.

Another issue is that relationship between models in the MDA framework, e.g. between PIM and PSM, is refinement. Refinement is opposite to abstraction and is a more detailed model that conforms to its related model. In other words, PSMs can be considered as more detailed description of its related PIMs where the PSMs is derived from the PIMs. The relationship between the two different models, addressing same domain, can hardly be described as refinement. The models are autonomously defined, and their relationship lies in the fact that both models describe same or similar data, but in different ways.

In addition to this, the main focus within the MDA framework seems to be on transformation between models using its metamodels as input. In terms of interoperability, the focus needs to be on model instances, together with the model itself, as data fields structured according to one model are to be populated at execution with data fields structured according to another model.

Further, the MDA framework captures how one model is converted into another model of the same system according to [64]. Transforming between two different models, is more about transforming between different systems.

Despite these differences, the model transformation between the two ways of capturing real-world entities, can be viewed in a way that fits the MDA framework. A model transformation in MDA can be carried out, where the target model is the same as the source model as pointed out by [64]. If the target model is viewed as part of the system, representing its external interfaces, the transformation can capture how to update the target model with relevant data elements at execution time. Object models are UML models that include data. In that way, the relationship between the input model and the output model, which happens to be the same model, can be about supplying additional information or data.

This could be illustrated in the following figure, where MDA based transformation defines e.g. how to update an instance of the UBL object model from figure 10, using instance of the Compiere model as an input together with related models:
6.4 QVT – MDA standard for expressing transformations

As already mentioned, OMG issued a RFP for MOF 2.0 Query, Views, Transformations (QVT) in October 2002. QVT is to be part of the MDA framework, a standard that addresses how transformations are achieved between models defined using MOF.

The language is to have three subsets or sub-languages as indicated by its name:

- **View – A language for creating views on model**
  According to [51], the view is to be derived from a model, revealing certain aspects of the modelled systems. According to [22], view can be partial, is often read only and does typically not exist independently of the source model. View might contain similar or same information as its related source model, but maybe organised differently in order to support special task or usage. Transformations generate views according to [22].

- **Query – A language for querying model**
  Querying a MOF model is similar to querying a database. The query is to return set of model elements or a boolean value based on some criteria defined or question asked. An example of queries over UML models, ref. [22] are “Return all packages that do not include any child packages” and “Does a particular attribute in the source have public visibility”, where the former query returns a set of model elements and the latter returns a boolean value.

- **Transformation – A language for writing transformation definitions**
  Transformations generate target model from source model. According to [22], the derived model can be either dependent or independent. Being dependent indicates that participating models are still related after transformation and all later changes can therefore affect involved models. Being independent indicates that model can evolve independent of each other after transformation. Transformations, where target model replaces the source model, are called update transformation.

My main concern in context of this thesis is the language for writing transformation definitions but all the languages are interrelated as transformation uses queries in order to...
navigate in models and views are generated by transformations. As explained in chapter 6.1, transformation is the n-ary relationships between input and output models as well as the mapping between them. If viewing transformation between conceptually different models as detailing of the target model with source model as input, the transformation to be carried out can be characterised as update transformation.

One of the requirements in the QVT RFP states that the all languages shall have abstract syntax that is defined as MOF compliant model. This indicates that a transformation definition is a model instance in similar way as the models participating in the transformation. The available constructs for writing transformations are therefore formally defined and captured by a metamodel. The following figure is to illustrate this, using what can be considered as basic pattern for transforming e.g. between PIM (Model A) and PSM (Model B).

Another requirements states that the transformation definition language shall have declarative syntax. Declarative syntax expresses relationships between elements in terms of functions and inference rules rather then in terms of sequence of steps which is the case with imperative languages [22]. An example of declarative language is Prolog, a language that expresses what to do rather then how to do it (recall chapter 5.1.3).

These two requirements, requiring abstract as well as declarative syntax, deserve further attention in the context of this thesis. Having abstract syntax is closely related to providing graphical notation of the model elements. Having declarative syntax needs to be discussed in terms of transformation execution. These issues are further elaborated on in next two sections.
6.4.1 Abstract syntax - graphical notation

As illustrated in figure 27, the QVT language is to be MOF compliant, having an abstract syntax implying that the language is in itself a model. Models are abstraction and the notation for a capturing them can very well be textual. Despite that, modelling languages are commonly associated with having notation for graphically representing the model elements. The advantages of using e.g. UML as modelling language for capturing conceptual models, is in addition to its wide tool support, the graphical notation of model elements. The graphical notation is considered to be very well suited for communication between humans.

In [80], Willink clearly states his view where he believes that model driven technologies should be modelled using graphical notation, including the transformation language. He comments on different submissions to the QVT RFP and is surprised regarding the fact that few of the earlier submissions have complete graphical syntax for the transformation languages suggested. In [80], Willink presents a transformation language that has complete graphical syntax based on UML, called UMLX. Different QVT based languages are addressed in next chapter but the intention here is to point out that, as the QVT language is a model, defined in MOF compliant way, a graphical notation of its modelling constructs should be considered as an advantage.

Addressing graphical notation of the QVT languages brings the discussion back to tool support for modelling languages. Recall from chapter 6.2 that UML profiles are considered important when addressing ontology modelling, as current tools primarily support the graphical notation of UML model elements. Usage of UML profiles enables using UML modelling tools for creating models.

6.4.2 Declarative syntax – transformation engines

To explain in detail the technology needed to enable execution of declarative transformation definitions, is far beyond the scope of this thesis. Thesis intention is to give an overview of the current status of the MDA technology with focus on a model-based mapping. What I want to point out here is that even though a language for capturing transformations is defined, the mechanism for executing the language also needs to be addressed. In similar way as the OWL web-based languages need inference engines to be written on top of them, transformation languages need inference engine or transformation engine to be implemented, in order to execute the transformation definition.

I can explain to certain degree what this implies in the context of the most recent submission to the QVT RFP ref. [60]. This is a revised submission from QVT Merge-Group where different partners previously submitting QVT proposals work together in order to agree on single language. In this submission, the transformation definition language is defined as set of three related languages: Relations, Operational Mappings and Core. The language proposed is in fact a hybrid language, including both declarative and imperative language constructs.

The Relation is a high level language that is able to express information needed in order to map between different models, using their metamodels as input. The constructs in the Relation language have graphical notation. The language called Operational Mappings is an
imperative language, which can be invoked from *Relation* as well as *Core*. In addition to Operational Mappings, the proposal allows for functions written in other languages to be invoked at execution time, in a Black Box or plug-in manner. The role of the *Core* language is explained using Java virtual machine as analogy. If the *Relation* language is viewed as Java language, the *Core* language can be viewed as Java Byte Code, where transformation from *Relation* to *Core* is needed in order to execute the transformation definition. Transforming from *Relation* to *Core* is therefore analogous to the Java compiler, where the input is *Relation* and the output is *Core* or executable transformation.

Based on this I conclude that transformation engine can be seen as analogous to the Java Virtual Machine, capable of executing the *Core*.

The following figure from [60] illustrates the relationship between the different languages.

![Figure 28. Relationship between different MOF2.0 QVT sub-languages](image)

### 6.5 Transformation languages and tools

The MDA framework and specially the technology for writing and executing transformation in the context of MDA framework is very much in its childhood, in similar way as the ontology technology discussed in chapter 5. The latest QVT proposal previously referred to, is submitted on 1\textsuperscript{st} of July 2005 and this is written in October 2005.

Several languages have been used for implementing transformations with MOF compliant models as input and I will briefly discuss few of those languages as well as their currently available tool support.

When referring to tool support, I am referring to editors available for creating transformation definitions preferably supporting a graphical notation, as well as transformation engines capable of executing transformations.
6.5.1 Atlas Transformation Language (ATL)

ATL is a QVT based transformation language developed by the French national research institute INRIA\(^\text{19}\), in co-operation with the University of Nantes. ATL is intended to be a proposal answering to OMG’s RFP for a QVT standard according to \cite{2}. It is a model transformation language defined using abstract syntax in terms of its metamodel. ATL is a hybrid language, with both declarative and imperative constructs. Declarative constructs support simple transformations but imperative constructs are needed to support more complex transformations.

Although defined using a metamodel, ATL currently supports only textual notations for writing transformations. Transformation definition consists of rules and helpers. Rules are central elements in ATL transformation language. Rules have \textit{from} and \textit{to} parts, defining the relationships between the source model elements (from part) and target model elements (to part). Model parts are identified by means of pattern matching, using OCL. Helpers can be viewed as kind of functions that can be written in order to aid in the transformation process. Helpers contain OCL expressions \cite{3}.

A virtual machine for executing ATL transformation has been defined and implemented. In addition, ATL Development Tools (ADT) provide text editor with various support in order to create and execute transformations.

ADT are still under development but are available as Eclipse plug-in, published under the Generative Model Transformer (GMT)\(^\text{20}\) project. ADT depend on Eclipse Modelling Framework (EMF) in order to navigate, modify and serialize models. Serializing models is about converting them to XMI format.

According to ref. \cite{18}, “EMF can be thought of as a highly efficient Java implementation of a core subset of the MOF API”. EMF is also an Eclipse plug-in. The core model in EMF is called Ecore and is similar to a core elements in MOF 2.0, referred to as EMOF (Essential MOF). The EMF framework does therefore provide the available constructs used to create the ATL metamodel, as well as set of services used to work with MOF compliant models.

Eclipse\(^\text{21}\) is referred to as an extendible open source Integrated Development Environment (IDE). The Eclipse IDE workbench is a platform for hosting tools developed as Eclipse plug-ins. Simplifying, it can be said that the Eclipse platform provides general functionality needed by teams developing software artifacts, independent of the programming or modelling languages being used. This includes e.g. text editor framework, help system, configuration tool integration, support for debugging and building etc. In addition, the platform provides services or API’s for extending the existing functionality, in terms of software plug-ins. Tool providers can therefore make use of all the services already provided by the Eclipse platform, instead of having to develop their own. When providing

\(^\text{19}\) Inria public website: \url{http://www.inria.fr/}

\(^\text{20}\) GMT project website: \url{http://www.eclipse.org/gmt/}

\(^\text{21}\) Eclipse public website: \url{http://www.eclipse.org/}
functionality that extends the platform, tool providers can do so by making use of other additional plug-ins already developed, as done in the case with ATL when making use of EMF.

### 6.5.2 MOF2.0 – QVT

The most recent submission to the QVT RFP, is referred to as MOF2.0-QVT [60]. This is, as already mentioned, a revised submission where different partners previously issuing proposals, come to an agreement. One of the partners supporting this proposal, is INRIA, the research organisation participating in developing ATL transformation language.

Current tool support is limited, as this proposal has just recently been issued. The MOF 2.0-QVT language has both textual as well as graphical notation and is defined with an abstract syntax captured by a metamodel, an instance of MOF. The different sub languages; Relation, Operational Mappings and Core are already addressed in chapter 6.4.2. Despite current lack of support, it is reasonable to assume that tool support will be available soon. The MDA framework is currently getting attention with large tool providers and some of the submitters of MOF2.0-QVT are in fact large tool providers.

Making an effort to summarise, the following points should describe current status of MOF2.0-QVT:

- The language has been defined, including both graphical and textual notation.
- Transformation from Relation to Core has been defined, indicating that it has been defined how to transform from high-level transformation definition to executable code.
- Currently, transformation engines for executing MOF2.0-QVT transformation definitions are not available.
- Currently, tool support for creating MOF2.0-QVT transformation definitions using graphical notation is not available.

The following figure illustrates how QVT Relation (rules and techniques) are defined in MOF2.0-QVT. The example is taken from [60] and includes both graphical and textual notation for expressing the same relation named “PrimitiveAttributeToColumn”. The relation expresses how to convert all instances of primitive UML attributes into columns in an RDBMS model. The transformation uses information captured in the relevant metamodels to express the relationships. I deliberately chose this example as it shows how the body of a function “PrimitiveTypeToSQLType” in the MOF2.0 QVT is expressed using the imperative language, Operational Mappings and does therefore not have a fully graphical notation.
6.5.3 UMLx

UMLx is a proposal for model-to-model transformation language with graphical notation, based on graph theory [81]. This is an independent work of Edward D. Willink. The graphical notation is defined using minor UML extensions, and Willink states that stronger integration with UML is one of its main contributions.

The language has a declarative graphical syntax as well as an underlying metamodel, but textual syntax is not defined. Transformations have been developed using XSLT [78]. Editor for graphical notation has been supported using GME [38], a generic tool for defining domain specific metamodels as well as defining graphical notation of modelling constructs. These approaches have limitations according to [81] and further development of the language is awaiting approval of QVT standard for writing transformations. Then “UMLX will re-appear as a transformation of its compact graphical notations into the QVT abstract syntax” [81].

The following figure is copied from [80] and addresses same transformation rules as figure 29, but using UMLx notation instead of MOF2.0–QVT.
6.5.4 MOF2Text - MOFScript

OMG issued in April 2004, a proposal for a standard referred to as MOF Model to text transformation [52]. This language, referred to as MOF2Text, is to fulfil the need for transforming models into textual representation. Current QVT proposal does not address this as it’s primarily concerns are model-to-model transformation, e.g. for transforming between PIMs and PSMs.

PIMs is in itself an abstract concept depending on the definition of a platform. Although MOF2Text language is intended to address transformation from e.g. PSM to code, author’s of [47] argue that MOF2Text could also be applied for transforming PIMs directly to code. The following figure from [47] illustrates possible applications of model to text transformations in relation to model-to-model transformations.
Several proposals have been submitted as response to the OMG request. I will here briefly introduce one of these languages which is referred to as MOFScript [49], a revised submission submitted by Softeam[^22^], supported by Sintef.

MOFScript is an imperative language. Although intended for writing model to text transformations, MOFScript can also be used for writing model-to-model transformations. Transformations are defined by means of rules that are executed explicitly in the same sequence as defined by the script. The language has an underlying metamodel, which is defined as an extension to one of the metamodels belonging to latest MOF2.0-QVT, more specifically the sub language OperationalMappings (recall chapter 6.4).

MOFScript has number of additional functionality needed in order to support writing text as opposed to models. Some of these additional functions are organised into libraries, providing a set of built in functionality available for writing transformations. The following function libraries are identified:

- **String library**: containing set of string manipulation functions in addition to the functions available through use of OCL. This includes functions such as `subStringBefore`, `subStringAfter`, `replace`, `toUpper` etc.
- **Collection library**: providing abstract datatypes in terms of `list` and `hashtable` as well as functions to manipulate with these.
- **Utility library**: Including set of utility functions such as data, time, indent, newline, tab, etc.
- **XML library**: providing small set of built in element in order to support creation of XML documents as output. This includes functions such as `addAttribute`, `addElement`, `replaceNamespace` etc.

A tool prototype, developed as Eclipse plug-in, is available for download[^23^]. The tool supports parsing, checking and executing MOFScript.


[^23^]: Downloads available at: [http://www.modelbased.net/mofscript/](http://www.modelbased.net/mofscript/)
6.5.5 XSLT – while QVT based languages lack tool support

Many of current approaches for implementing and executing MDA transformations have relied on XSLT [78] as transformation language. This is the case with the approach described by Djuric et. al. [16] where authors implement executable transformations between OWL and UML profiles using XSLT. XSLT is an imperative transformation language that, together with XPath [76] used for querying and pattern matching, can be used to parse any XML-based languages and produce a desirable output. As both OWL and UML have XML-based representation format, XSLT can be used to parse OWL and UML/XMI models.

Both XSLT and XPath are part of what is referred to as XSL (eXtendesible Stylesheet Language) family. XSLT stands for XSL Transformation and XPath is short XML Path Language. XSLT and XPath are W3C recommendations.

XSLT lacks graphical notation but transformation engines for executing XSL transformations are available. It is pointed out in [52], that XSL transformations are good for transforming XML documents but when applied to XMI documents, the result tends to be complex and ill-maintainable code.

6.6 Semantic mapping – OCL evaluated

Mapping as pointed out in chapter 6.1 denotes the rules and techniques needed to express which elements are related or under what condition as well as how to convert between them.

The QVT-based transformation languages, MOF2.0-QVT, UMLx and ATL, all rely on OCL when expressing relationship between models as well as expressing how to convert between model elements. OCL [53] is a declarative language now included as part of UML2.0. OCL can be used as a query language over model elements. In addition OCL expressions are used for stating constraints over model elements in terms of e.g. invariants on classes and types, guards, pre and post conditions.

Although the intended usage of MOF2.0-QVT, UMLx and ATL is mainly model-to-model transformations, these languages can also be used to transform between instances using object models as would be the case when addressing interoperability (recall chapter 6.3). Anyhow, as these languages all rely on OCL for querying and writing expressions. OCL is the language that will be used to relate elements as well as expressing the techniques for converting from one way of representing data to another.

As an example, the when clause of the graphical notation in figure 29 is composed of OCL expressions. Number of examples are also found in the textual notation. The function body contains OCL expressions between {}. The relation also contains number of OCL expressions where e.g. “uml c:Class {attribute=a:Attribute {name=an, type=p:PrimitiveDataType {name=pn}}}” expresses the same as the left hand diagram in the graphical notation in figure 29.

The following table gives an overview of how OCL supports the necessary constructs identified in chapter 4.1, addressing the requirements stated towards a transformation language.
Table 11. Evaluation of OCL

<table>
<thead>
<tr>
<th>Requirement-ID</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR2-TL-StringFunctions</td>
<td>Available functions to manipulate with strings are:</td>
</tr>
<tr>
<td></td>
<td>- substring(lower:Integer,upper:Integer):String</td>
</tr>
<tr>
<td></td>
<td>- concat(s:String):String</td>
</tr>
<tr>
<td>FR3-TL-Math</td>
<td>OCL supports common mathematical functions for its integer and real types.</td>
</tr>
<tr>
<td>FR4-TL-Logic</td>
<td>OCL IF expression supports this.</td>
</tr>
<tr>
<td>FR5-TL-TypeCasting</td>
<td>OCL has four primitive basic datatypes; boolean, integer, real and string.</td>
</tr>
<tr>
<td></td>
<td>OCL supports casting string to integer and real through string functions</td>
</tr>
<tr>
<td></td>
<td>toInteger and toReal.</td>
</tr>
<tr>
<td>FR6-TL-DataAssignment</td>
<td>OCL Let expression supports this.</td>
</tr>
<tr>
<td>FR7-TL-SemanticRelations</td>
<td>OCL is used to express relationships between model elements by identifying</td>
</tr>
<tr>
<td></td>
<td>model elements being source and target for transformation. Model elements</td>
</tr>
<tr>
<td></td>
<td>are identified by means of pattern matching using OCL collections and</td>
</tr>
<tr>
<td></td>
<td>constraints upon them. Ability to express non-existing relationships between</td>
</tr>
<tr>
<td></td>
<td>concepts, addressing coverage or homonyms, is not clearly available.</td>
</tr>
</tbody>
</table>

6.7 Conclusions

Based on previous discussions in this chapter on transformations, the following conclusions can be drawn.

- The focus for the MDA framework is transformation between different abstraction levels or between different modelling languages.
- Mapping between semantically similar models can be aligned with the MDA framework.
- The transformation languages being addressed in the context of MDA framework, seem to provide functionality needed to map between semantically similar models.
- MDA transformation technology together with a standard metamodel for defining ontologies is about to provide functionality for converting ontologies between different expression mechanisms, OWL and UML respectively.
- MDA based transformation is getting lot of attention in the commercial tool market, indicating that tool support will be available soon.
- The support for graphical notation of modelling languages seems to primarily apply to models captured using UML. Graphical editors for supporting graphical notations for transformation languages are not commonly available.
- The functions needed to convert between different ways of representing data elements are supported, either through the use of OCL or the ability to initiate functions written in external languages.
7. Existing solution approaches – case studies

In this chapter, I will explain two related approaches for facilitating interoperability. The overall context is a large European research project called ATHENA. The approaches being explained are ATHENA subprojects referred to as A3 and A6. A3 is to facilitate interoperability using an ontology-based approach for explicitly defining semantics of information to be interchanged. A6 on the other hand, facilitates interoperability using a model-based approach.

I will first give an overview of the ATHENA project and then explain the two subprojects involved. When this is written, the ATHENA project is still in its initial phase. Some of the documents being referred to, are currently internal working documents but will apparently become publicly available in due time.

During my survey of the ontology-related literature, I have managed to identify some additional approaches where an ontology is being applied for addressing e-commerce related interoperability. I only mentioned these approaches here as they are relevant for the context but I will not address them any further. These approaches are:

♦ **HARMONISE** an approach described in the article: “HARMONISE: a Solution for Data Interoperability” [14]. A mapping is done against a global domain ontology where this ontology is a result of wide co-operation between large and small actors within the business of travel and tourism in Italy, which again are members of THN (Tourism Harmonise Network). All members exchange information using an interchange format called Harmonise Interoperability Representation (HIR). Each partner then interacts using, custom semantic gateway (CSG) to convert from his local way of representing information to the standard representation format. CSG denotes sets of reconciliation or mapping rules, as well as well as transformation functions and engine necessary to process them.

♦ **ARTEMIS** is an approach described in the article “Ontology-based Interoperability for Interorganizational Applications” [13]. ARTEMIS is the name of the tool environment that is used to support an ontology construction and deployment. The ontology used is constructed by analysing inter-organisational order documents with focus on product description. This is done in order to design a shared conceptualisation of the product descriptions, currently varying from department to department. The result of this analysis is an integration of the different descriptions, which again becomes one of the layers in the ontology architecture. The goal is to present the enterprise as single actor when ordering products from external suppliers in order to achieve commercial advantage or simply put, lower prices.

7.1 ATHENA - introduction

ATHENA is the name of a large integrated project that started 1st of February 2004. The project is sponsored by the European commission and is scheduled to last 36 months. It
involves 19 independent partners, bringing together research teams, industry expertise and ICT suppliers, all working together in order to enable the following vision:

“By 2010, enterprises will be able to seamlessly operate with others” [32]

Mobilising various partners in an integrated network contributes to achieving so called critical mass. Critical mass refers to number of partners adhering to a certain solution, which again makes it attractive for others to join as well. Currently, there are around 20 partners joining the network according to [32], e.g. SINTEF from Norway, LEKS form Italy, SAP AG from Germany and AIDIMA from Spain, just to name a few.

One of ATHENA’s objectives is to define a technology neutral reference model or a framework including methods, tools and techniques, and by that providing various solutions in order to meet interoperability requirements that also will be defined as part of the project. ATHENA recognises the need to integrate already existing, heterogeneous and autonomous ICT systems, but will also address business tasks and processes not necessary supported by current ICT solutions.

ATHENA is actually a program made up of number of interrelated projects [31]. The different projects are categorised based on so called action lines. Action line A addresses research and development, including design and development of the interoperability framework. Action line B addresses community building, such as disseminating the research results to the community as well as addressing training, testing and various other engagement activities. Action line C addresses overall management of the program, including business and technical support functions.

In the context of action line A, several projects have been identified. These are listed here together with the primarily goal or objective of the projects:

- **A1: Enterprise Modelling in the context of Collaborative Enterprises** – where overall goal is “development of methodologies, core languages and architectures, model generated workplaces and services and execution platforms for establishing collaborative on-demand Extended Enterprises and Networked Organisations” [31].
- **A2: Collaborative Business Process Execution** – where the goal is to “provide a solution that enables execution of Cross-Organizational Business Processes” [31].
- **A3: Knowledge Support and Semantic Mediation Solutions** – with “main objective to build an ontology-based platform for the on-line reconciliation of messages and interactions that takes place between software applications that have not been originally conceived to exchange information” [31].
- **A4: Interoperability framework and Services for Networked Enterprises** – “The object of this project is to analyse the approaches, methodologies and results of the other projects in Action Line A” [31].
- **A5: Planned and Customisable Service-Oriented Architectures** – with an objective to “develop service oriented components able to support interoperability and build upon existing legacy systems” [31].

---

24 Information & Communication technology
A6: Model-driven and adaptive interoperability architectures – “The main objective of this project is to provide disparate systems attempting to communicate an adaptive model driven, platform independent architecture to facilitate interoperability” [31].

My focus will be on two of the projects defined within action line A. The first one is A3, a project led by LEKS in Italy, that focuses on an ontology-based platform in order to facilitate interoperability. The second one is A6, a project led by SINTEF in Norway, where the focus is model-driven approach for facilitating interoperability. The following figure is to illustrate the scope of the different subprojects, A3 and A6. These two sub-projects are further elaborated in the following chapters.

Figure 32. ATHENA, the scope of A3 and A6 illustrated

7.2 A3 – Knowledge support and Semantic mediation Solutions (Ontology-driven)

The objective of A3 is to build an ontology-based platform in order to resolve information conflicts involved during software interchange. A3 project has identified set of tools and artifacts that will be created in order to achieve the defined objectives. The following figure illustrates the A3 approach, including the set of tools (Athos, A*, ARGOS and ARES) as well as illustrating the process needed to produce sets of reconciliation rules. The rules can be seen as the result of identifying mappings between local data sources and reference ontology as well as how to convert from local representation to shared representation.
The different tasks needed in order to enable seamless interaction between enterprises using different conceptualisation are the following according to A3:

- Ontology Management
- Semantic Annotation
- Reconciliation rule creation

Each of the different tasks including the different tools supporting them, will be further elaborated on in the following sections.

At run time, when e.g. business documents are to be exchanged, the created rule set is input to a reconciliation engine which executes the rules. This results in a task, where instances are transferred from local way of organising and naming information to an information structure that is organised and labelled according to the reference ontology, as well as the other way around.

### 7.2.1 Ontology Management - Athos

One of the goals of A3 project is to provide methods and tools for ontology management. In order to support this goal, A3 project defines a representation mechanism for capturing the reference ontology, as well as building an ontology management tool for supporting construction and maintenance of the reference ontology.

The representation language to be used is called OPAL [42]. OPAL is short for Object Process Actor modelling Language. An Ontology that is constructed using this representation mechanism has a three layered architecture, a top layer referred to as *Upper Domain Ontology*, middle layer referred to as *Application Ontology* and bottom layer referred to as *Lower Domain Ontology*. The middle layer includes concepts that are specialisation of concepts in the top layer. The bottom layer includes elementary concepts, such as `price`, `street number` etc. The approach typically used is to construct more complex concepts residing at the middle layer, in a bottom up manner, using the core elements from the bottom layer. Examples of concepts residing at middle layer are e.g. `discount`, `invoice`
and customer. These concepts are then categorised and top level grounded using concepts at the top layer. This architecture results in an illustration resembling a chestnut, as indicated in the following figure.

![Ontology chestnut diagram](image)

**Figure 34.** A3 – Ontology chestnut

According to [8], the OPAL language is an extension to OWL with the top level concepts borrowed from UML modelling language. The intention is to provide a language that is targeted towards domain experts currently familiar with UML, but at the same time being able to use advanced reasoning and query services available for OWL and DL expressions.

The relationship between concepts at different layers is indicated in the figure. IS-A relates concepts in middle and top layer, where IS-A can be viewed as UML generalisation/specialisation relationship. PART-OF relates concepts in middle and bottom layer where PART-OF can be viewed as UML composition relationship.

The task of adding concepts to the ontology, is to be supported by a tool called Athos. Athos is developed as a web-based application, based on the Zope platform. Zope is an open source platform that has an underlying plugin-architecture in similar way as the Eclipse platform (recall chapter 6.5.1).

### 7.2.2 Semantic Annotation – A*

Semantic annotation is used to associate formal meaning to information structures and processes. Supporting structural aspects involves annotating data sources such as document schemas or models with respect to the reference ontology. This means that each data source element is linked to a concept in the reference ontology and by that, explicitly defining its meaning. This is to be done using stepwise approach where elements are associated using increasingly complex expressions according to [67].

---

The method to be used has two phases [67]. The former is referred to as diagnostic phase and is aimed at identifying possible relations or mappings between concepts. The latter is referred to as remedial phase and is aimed at building annotation expressions, representing the actual semantic mapping between concepts involved. Each phase has two steps, which implies that the method has four steps all together. The following table includes a brief description of the different steps.

<table>
<thead>
<tr>
<th>Phase 1</th>
<th>Diagnostic Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1</strong></td>
<td>Terminological Semantic Annotation (TSA)</td>
</tr>
<tr>
<td></td>
<td>Concepts in data source are linked to corresponding concepts in the reference ontology, addressing conflicts of type naming, coverage and abstraction.</td>
</tr>
<tr>
<td></td>
<td>Ex: TSA(Buyer)=[Buyer_BA] TSA(hasTelephone)=:[hasPartTelephone], TSA(areaCodeandCountrycode)=:[hasAreaCode, hasCountryCode]</td>
</tr>
</tbody>
</table>

| **Step 2** | Path Semantic Annotation (PSA) |
| | Here, the terms from previous steps are gathered to form a path in the hierarchical structure of the data source. |
| | Ex: PSA(Buyer.hasTelephone.AreaCodeandCountrycode)>>:[Buyer_BA.hasContactInfo.HasPartTelephone.haspartAreaCode,Buyer_BA.HasContactInfo.HaspartTelephone.hasPartCountryCode] |

<table>
<thead>
<tr>
<th>Phase 2</th>
<th>Remedial Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 3</strong></td>
<td>Simple Semantic Annotation (SSA)</td>
</tr>
<tr>
<td></td>
<td>The paths from previous steps are used to compose simple expressions or preliminary mapping between the data source and reference ontology. This is done using abstract operators, ( \oplus ) for binary operators and ( \phi ) for unary operators.</td>
</tr>
<tr>
<td></td>
<td>Ex: SSA(Buyer.hasTelephone.AreaCodeandCountrycode)&gt;&gt;:buyer_BA.HasContactInfo.hasPartTelephone.haspartAreaCode ( \oplus ) Buyer_BA.HasContactInfo.hasPartTelephone.haspartCountryCode</td>
</tr>
</tbody>
</table>

| **Step 4** | Full Semantic Annotation (FSA) |
| | The simple annotation from previous step is refined and coded using OWL DL expressions. |
| | Ex: FSA(Buyer.hasTelephone.AreaCodeandCountryCode)>>: (\( \forall \) relCountrycode_ispartOf.( \( \forall \) Telephone_isPartOf. \( \forall \) ContactInfo_isAttributeOf.Buyer))) complex_operator (\( \forall \) relAreacode_isPartOf.( \( \forall \) Telephone_isPartOf. \( \forall \) ContactInfo_isAttributeOf.Buyer))) |

The tool supporting the semantic annotation is referred to as A*.

### 7.2.3 Reconciliation rule creation - Argos

The output of this task is a set of executable reconciliation rules. According to [71], the technology chosen when constructing the rules is based on RDF/Jena2. Jena is a framework
for building Semantic web applications, developed by HP laboratories\textsuperscript{26}. Jena2 is the latest version, providing set of functionality referred to here as the Jena2 suite. A part of the jena2 suite is a rule language together with a rule based inference engine, capable of executing the rules. In addition, the jena2 suite provides functionality for parsing XML-based documents and converting them to RDF's\textsuperscript{27}.

The reconciliation rules in A3 will be constructed using a rule language adopted by the Jena2 suite. Jena rules may contain so called functors in order to increase the expressive power. Functors are a way to include calls to externally defined methods e.g. implemented in Java. The following is an example of Jena rules, including a functor named merge.

\[
\text{[rule1:}(?x \text{ BuyerName } ?y)(?x \text{ BuyerSurname } ?z)(?x \text{ rdf:type RFQ})
\rightarrow
\text{Merge}(?y ?z ' ' ?b)(?a \text{ BuyerpersonName } ?b)(?a \text{ rdf:type RequestForQuote})]\]

\textbf{Figure 35.} Example of Jena rules, including a functor

Argos is the tool that will support the task of producing a set of reconciliation rules. In order to hide the complexity of the Jena rule syntax from the user, a more abstract syntax has been defined. The user will work with the SSA expressions produced during semantic annotation, and apply “Argos rule templates” when refining the abstract operators. In addition, the user is to have access to data source in RDF format and the reference ontology, where the structural path of each element being mapped can be visualised using graph like notation.

The different rule templates can be viewed as abstract representation of the functors used within the formal jena rules. An example of Argos rule template is “MergeRT(node1,…,nodeN, string sep. nodeRes)” where MergeRT takes node1,…,nodeN as input and merge these using separation string “string sep” as delimiter. The following rule templates are identified in [71]:

\begin{itemize}
\item \textbf{MapRT} for assigning value of one node to another.
\item \textbf{MergeRT} for merging content of several nodes to one node.
\item \textbf{SplitRT} for splitting content of one node into content of several nodes.
\item \textbf{MapTableRT} for mapping list of input values (nodes) to list of output values (nodes).
\item \textbf{ConvertRT} for converting numerical value to another using exchange rate.
\item \textbf{MapValueRT} for assigning a fixed value to a node.
\item \textbf{SumRT} for assigning to an output node, a sum of values assigned to list of input nodes.
\item \textbf{MultRT} for assigning to an output node, a value obtained by multiplying set of values assigned to list of input nodes.
\end{itemize}

\textsuperscript{26} Documentation is available at: \url{http://jena.sourceforge.net/documentation.html}

\textsuperscript{27} RDF was explained in chapter 5.3.2
7.3 A6 - Model-driven and adaptive interoperability architectures

The main objective of the A6 subproject is to provide communicating systems, a platform independent, model driven and adaptive architecture in order to facilitate interoperability. Model driven interoperability addresses the MDA approach or Model Driven Development (MDD), where models drive design and development of information systems. Adaptive architecture refers to run time aspects of system development, where P2P technologies are applied in order to enrich systems with dynamic and adaptive qualities.

In order to achieve the main objective of A6, a set of sub goals has been defined. These are according to [32]:
1. To support requirements and validate solutions for the involved sectors from Activity B4, Activity B5 and Project A4.
2. To provide meta models and methodologies for interoperability architecture solutions.
3. To evaluate and extend multiple adaptive autonomous and federated architecture approaches, including Agent and Peer Technologies and the Model-Driven Architecture approach.
4. To provide support for non-functional interoperability aspects, through a model-driven approach.
5. To apply the use of ontologies and semantics in model and service Registry/Repositories for better semantic interoperability.
6. To provide semantic mapping and mediation technologies.
7. To provide executable frameworks and support for active models.

The focus in this section is on sub goal 6 or how to provide semantic mapping and mediation technologies in the context of model driven architecture in order to facilitate interoperability. The goal of the semantic mapping and mediation in A6 is somewhat similar to the goal of A3 project. The difference is that A6 focuses on models and model driven architecture, emphasising that platform independent models should be the context when establishing mapping between two enterprises. In order to support this sub goal, a tool will be developed. High level set of requirements for the tool that is to support semantic mapping, has been defined, ref. [67].

♦ The tool should allow a user to define mappings between UML models
♦ The tool should provide help to perform the mapping using some form of semantics
♦ The tool should integrate into the ATHENA A6 execution infrastructure

When this is written, a first prototype for a tool referred to as “ATHENA Semantic Mapping Tools” (SMT) has been presented. This is only a first prototype with limited functionality. The tool implemented is based on the Eclipse framework as described in [34]. Further, the implementation makes use of EMF28, GEF29 and UML2.0 eclipse plug-ins in order to implement the graphical notation used.

28 http://www.eclipse.org/emf/
29 http://www.eclipse.org/gef/
According to [67], current functionality of the SMT needs to be extended in order to facilitate semantic interoperability using a model-based approach. The following topics are to be addressed:

- Create vertical transformations based on the mappings that are created by the mapping tool. The first target should be XSLT for transformation of XML documents.
- Elaborate on mapping model to support simple functions on the mappings. Examples of functions are: additions, string handling etc.
- Create more complex semantic helper operations, also including the use of ontologies.
- Look further into a graphical notation for the definitions of mappings.

### 7.3.1 Mapping relations – Semantic Mapping Metamodel (SMM)

The tool supports mapping between two conceptually different models addressing same domain, by allowing for creation of links between elements in the different models that are semantically related. The two conceptually different models are expressed in UML. The language used for capturing relations as well as expressions between the related elements is model-based, meaning that the language is defined in a separate metamodel. The current working name of the metamodel is UML2UML. For simplicity I refer to this metamodel hereafter as Semantic Mapping Metamodel (SMM). The following figure shows a segment of the metamodel illustrating how operations are to be captured. When this is written, the SMM is still at design stadium.

![Semantic Mapping Metamodel (SMM) capturing A6 – semantic mapping](image)

The mapping model is based on a general mapping model provided by the Eclipse Modelling Framework [18]. EObject is a MOF model element and is a general class that most parts of instance models inherit from. In that way, the instance of “Mapping” model-element can be
used to relate instances at both M2 and M1 level in the MOF architecture. The shaded classes are SMM extensions needed to capture transformation operations.

### 7.3.2 Semantic mapping tool – graphical notation

Currently, the user in charge of establishing mappings can choose between two kinds of user interfaces. One is based on a graphical notation where user creates an instance of the model element called *Mapping* and then relates elements in target and source models to the mapping element, where both source and target models are presented using graphical notation. The other interface is based on a tree like structure, where user creates mapping instance by dragging elements in source model onto elements in target model. The following figure shows the current graphical interface provided by the tool.

![Figure 37. ATHENA Semantic Mapping Tool – Graphical notation](image)

The mappings are nested in order to capture the structure or the path of each element to be mapped. Properties related to each mapping element are presented in the property panel, and can be viewed by selecting each mapping.

In addition to being able to manually define mappings between elements, current implementation of the tool allows the user to initiate a helping function, referred to as *MappingHelper*. This function is to automatically establish mapping between the different models using what is referred to as *Mapping Strategy*. The mapping strategy implemented so
far is based on string matching of label names, where mapping is automatically created between attributes labelled by same name.
8. Comparing approaches – ontology-based vs. model-based

Based on the information provided in the previous chapters, I will here use different approaches in order to elaborate on the ontology-based approach vs. the model-based approach to semantic mapping.

I will first discuss why and how the ontology-based approach and the model-based approach can be viewed as two alternative approaches for achieving similar tasks. I will focus on similarities addressing the following:

♦ The tasks included as part of the semantic mapping method.
♦ The languages used when addressing each task.
♦ Number of mappings needed.

Then I will elaborate on the strengths and weaknesses of each approach as well as illustrate how the different technologies used to carry out semantic mapping can be bridged in order to enable collaboration (integration and interoperability) between the two approaches.

Finally I will evaluate to which degree the different approaches fulfil the requirements identified in chapter 4.

The information provided is based on the different approaches to semantic mapping used in A3 and A6 respectively where A3 is an example of the ontology-based approach and A6 is an example of the model-based approach.

8.1 As two similar alternatives

The model-based mapping is about mapping between two conceptual models or PIMs. The ontology-based mapping is about mapping between data source and ontology where ontology is a shared conceptualisation or conceptual model agreed upon by different partners, a kind of standard. Despite whether semantic mapping is to be achieved using shared conceptual model or two local models, a relationship between semantically similar concepts needs to be captured.

In addition, the information about techniques needed to convert between different ways of representing similar or same information needs to be captured. The different techniques needed are reflected by the operators identified in table 4, such as merge, substring, add etc. I refer to these operators as transformation operators.

The process of associating and translating between semantically similar concepts in two different sources requires sets of tasks to be carried out. This process is what I have been referring to as semantic mapping. In case of using ontology, the number of mappings to be created is reduced down to two sets as pointed out in chapter 2.5. One set that defines how to
map from local model to ontology, and another set that defines how to map the other way around.

These tasks need to be addressed at design time and are necessary in order to later exchange business documents at run time. The following figure is to illustrate the similarity between the different approaches.

![Figure 38. Semantic mapping in context – Model-based vs. Ontology-based](image)

The main difference is that, during the model-based approach, two PIMs captured using UML are context for mapping while during the ontology-based approach, the context is reference ontology captured using OWL and some data source converted to RDF format. Semantic Gateway is here denoting a transformation engine as well as executable transformation rules that are the output from the semantic mapping process.

### 8.1.1 Semantic mapping method – ontology-based vs. model-based

A set of different tasks to be included as part of Semantic mapping method, can be identified based on discussion in previous chapters.

- **Semantic Coordination**: About automatically identifying semantically related concepts.
- **Affirmation**: About confirming or rejecting automatically identified relationships as well as adding new ones.
- **Conflict Resolution**: About refining relationships with information regarding how to solve conflicts.
- **Formal Mappings**: About refining previously captured information into formal machine interpretable expressions.

The following figure gives an overview of the different tasks needed. Two boxes are decorated with a user, typically representing a domain expert. This is to indicate that these tasks require considerable user interaction. The figure also includes the preparation phase
needed before semantic mapping can be carried out, as well as the executable transformation, which is the output of the semantic mapping.

In addition to illustrating the semantic mapping method (in the middle), the figure also illustrates how the expression mechanisms used during the model-based approach (left side) and the ontology-based approach, relate to the different tasks. The different expression mechanisms are the ones used in the projects A3 and A6 respectively. The abbreviations used (TSA, PSA, SSA, FSA, SMM) where explained in chapter 7.

The different tasks, as well as how they relate to the similar approaches is elaborated on in the following table.

**Table 13. Different tasks involved during semantic mapping**

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparation (PR)</td>
<td>First task is about preparing data sources for semantic mapping. In case of the model-based approach, the context used is PIMs and this task involves converting data sources to this context. This would typically concern applying MOF2Text technology to convert XML schemas to PIMs. The ontology-based approach is to provide a platform for ontology construction, although this is not viewed as part of the preparation phase. The preparation phase during the ontology-based approach, is typically about converting data sources to RDF format as preparation for semantic mapping.</td>
</tr>
<tr>
<td>Task Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Semantic Coordination (SC)</td>
<td>This is about automatically detecting evidence for semantic relationships, in similar way as described in chapter 5.6. Currently, A3 and A6 have limited support regarding this task but according to plans, this functionality will be provided. Rahm and Bernstein [61], give an overview of various algorithms that have been used for detecting semantic similarities, amongst them a usage of auxiliary library. According to A3 plans, WordNet[^30], a lexical database for the English language, will be used to aid in the process of automatically detecting similarities. A6 currently addresses semantic coordination by comparing label names.</td>
</tr>
<tr>
<td>Affirmation (AF)</td>
<td>This task requires user interaction. The semantic relationships detected during semantic coordination need to be confirmed/rejected by a domain expert. In addition, the domain expert can manually add more relationships. The A6 semantic mapping tool, currently supports this functionality by allowing the domain expert to create mappings between related elements. This information is captured as an instance of the Semantic Mapping Metamodel (SMM). The tool A* supports this functionality as well in terms of TSA and PSA where domain experts associates related terms using TSA and PSA expressions.</td>
</tr>
<tr>
<td>Conflict Resolution (CR)</td>
<td>This task is concerned with supplying techniques needed in order to convert between the different ways of representing similar information and thereby addressing the different conflicts described in chapter 3.2. Currently, this task is partially addressed during the ontology-based mapping by creating SSA expressions or when domain expert refines PSA expressions by adding abstract operators. Operators indicate whether transforming between model elements require unary or binary operations. This task is further addressed during reconciliation, where abstract rule templates are used to specify the technique needed to convert between different ways of representing semantically similar information. Currently, how to add transformation operators to the semantic mapping metamodel (SMM), is at design stadium. The SMM is to be extended in order to capture transformation operators during semantic mapping.</td>
</tr>
<tr>
<td>Formal mappings (FM)</td>
<td>This task is about specifying the relations and techniques required using a formally defined, machine understandable language. During the ontology-based approach, this is addressed when producing formal Jena rules. During the model-based approach, this is addressed by using MOFScript transformation language to produce XSLT transformation.</td>
</tr>
<tr>
<td>Executable</td>
<td>This is not an actual task but represents the output of the</td>
</tr>
</tbody>
</table>

8.1.2 Transformation language examples

In this section, I identify a concrete conflict and illustrate how this conflict is to be resolved using the different approaches. This is done in the context of the two ATHENA subprojects, A3 and A6 in order to explore the languages used to express relationships between model elements. I only use subset of the data structures presented in chapter 3.1.3, figure 10, and only address one type of conflict or granularity. Similar approach can be applied to all different types of conflicts identified in chapter 3.2 but this example is sufficient in order to illustrate that the ontology-based approach and the model-based approach, actually are two alternatives for achieving semantic mapping between two different conceptualisations.

The following figure contains a simplified subset of the examples presented in chapter 3.1.3. I use graphical notations to illustrate the mappings to be done. Using the model-based terms, model B is to be updated using model A as input. Using the ontology-based terms, model A is to be annotation using reference ontology B in order to convert from the model A structure to a structure based on reference ontology. The example taken is how to convert buyer’s street name which is represented by two fields in model A, `street` and `number` to buyers street name in model B, which is represented by one field labelled `StreetName`.  

<table>
<thead>
<tr>
<th>Transformation (ET)</th>
<th>semantic mapping method, which is a set of executable rules.</th>
</tr>
</thead>
</table>

The table above illustrates that both the model-based approach and the ontology-based approach to semantic mapping are to address all the tasks identified and can therefore be considered similar.
**Ontology-based**

The following table illustrates the language used during the ontology-based approach when expressing relationships between elements in two models as well as the techniques needed to convert between them.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Example</th>
</tr>
</thead>
</table>
| TSA        | TSA(Order) =: [Order_B]  
TSA(BuyerInfo) =: [BuyerInfo_B]  
TSA(Address) =: [Address_B]  
TSA(StreetName) =: [street_B,number_B] |
| PSA        | Order.hasBuyerInfo.hasAddress.hasStreetname >:  
Order_B.relTo_BuyerParty.relTo_Address.hasPart_street,  
Order_B.relTo_BuyerParty.relTo_Address.hasPart_number |
| SSA        | Order.hasBuyerInfo.hasAddress.hasStreetname >:  
Order_B.relTo_BuyerParty.relTo_Address hasPart_street ⊕  
Order_B.relTo_BuyerParty.relTo_Address.hasPart_number |
| FSA        | Order.hasBuyerInfo.hasAddress.hasStreetname >:  
∀ street_isPartOf.(∀ Address_isReTo.(∀ BuyerParty_isReTo.Order_B))  
U  
∀ number_isPartOf.(∀ Address_isReTo.(∀ buyerParty_isReTo.Order_B)) |
| **Abstract** | MergeRT(Order_B.relTo_BuyerParty.relTo_Address.hasPart_street,  
" " ,Order_B.relTo_BuyerParty.relTo_Address.hasPart_number) ->  
Order.hasBuyerInfo.hasAddress.hasStreetname |
| **Jena Rules** | [rule:  
(?x b:street ?y)(?x b:number ?z)(?x rdf:type b:Address) ->  
merge (?y ?z " " ?c)(?b a:StreetName ?c)(?b rdf:type a:Address)] |
Model-based

The model-based approach captures mapping information using graphical notation. The graphical notation is shown in figure 41, but the following table illustrates the languages used during the model-based approach.

MOFScript refines information captured by SMM model instance and produces executable XSLT transformation.

Table 15. Example languages using the model-based approach

<table>
<thead>
<tr>
<th>Expression</th>
<th>Example</th>
</tr>
</thead>
</table>
| **SMM/XMI** | &lt;?xml version="1.0" encoding="ASCII"?&gt;
&lt;uml2uml:UML2UMLMappingRoot xmi:version="2.0" ... 
&lt;nested&gt;
 &lt;helper xsi:type="mapping:FunctionPair" in2out="/(@functions.0)"/&gt;
 &lt;nested&gt;
 ... 
 &lt;helper xsi:type="mapping:FunctionPair" in2out="/(@functions.3)"/&gt;
 &lt;inputs href="file.uml2#_FVJAReEdq7kbTWa5xyQ"/&gt;
 &lt;inputs href="file.uml2#_FjJHvEreEdq7kbTWa5xyQ"/&gt;
 &lt;outputs href="newe.uml2#_LsaWoErfEdq7kbTWa5xyQ"/&gt;
 &lt;/nested&gt;
 ...

**MOFScript**

//this is an example of the logic behind a concat mapping
uml.Property::writeConcat(location:String,elems:list){
 var expr:String
 &lt;%&lt;%=self.name&lt;/%=&gt;&lt;%=
 elems-&gt;forEach(n:String){
 &lt;%&lt;xsl:value-of select="%&gt; location &lt;/%=&gt;n &lt;/%=&gt;%&gt;
 } 
 nl(1)
 }

**XSLT**

..."Order"
 &lt;BuyerParty&gt;
 &lt;Address&gt;
 &lt;StreetName&gt;
 &lt;xslt:value-of select="Order\BuyerInfo\Address\street"/&gt;
 &lt;xslt:value-of select="Order\BuyerInfo\Address\number"/&gt;
 ...

8.1.3 The model-based approach as $2n$ mappings instead of $2^n$

As pointed out in chapter 2.5, the difference between using a standard when communicating instead of defining mappings between each pair of enterprise, is that the former requires $2n$ numbers of mappings while that latter requires $2^n$. Using ontology when defining semantic mappings can be viewed as using a standard.

Current technology support for the model-based mapping, is illustrated in figure 42. This approach takes two PIMs as input, where the two models might be two proprietary ways of structuring and naming real-world concepts. In this case, the two partners communicating need to agree upon who will be responsible for carrying out the work or whose structure will be used when exchanging information.
The model-based approach – current technology

The context for semantic mapping applying the model-based approach is PIMs. In cases where models are not available, the PIMs is to be extracted from existing systems by means of ADM principles. ADM stands for Architecture Driven Modernisation and is an approach defined by OMG for addressing reverse engineering of existing systems [48].

Relation between different model elements is captured using a language that is defined by a Semantic Mapping Metamodel (SMM). A transformation definition, written using MOFScript language, converts information captured by an instance of SMM to XSLT script. At run time, XSLT script transforms XML document structured according to schema A, into an XML document structured according to schema B.

Current solution for the model-based approach, expects data to be exchanged using XML format. As mention in chapter 2.2.4, XMI can be used to serialise models at any level in the MOF architecture. Data sources represented as instances or object models can be serialised into XMI format. XMI is therefore a possible interchange format in addition to XML. According to [52], XSL can be used for transforming XMI documents but the resulting code tends to be complex and difficult to maintain. Therefore, MDA transformation languages should be preferred if XMI is to be the agreed interchange format instead of XML. Despite approach, transformations need to be written to convert data from each different internal representation formats to the agreed upon format being used for interchange, whether it is e.g. XML, RDF, XMI or some other format. This depends on the transformation technology or engine to be used.

Support for executing e.g. ATL and MOFscript is currently available and these transformation languages both accept XMI models as input. An alternative solution to the current one, where XMI is used as interchange format and ATL as transformation language is illustrated in the following figure:
Yet another alternative that could be considered is to use MOF2.0-QVT, either instead of e.g. ATL or all together. Currently, tools supporting graphical notation of the MOF2.0-QVT language are not available and neither is transformation engine. Given that a revised standard has been submitted, it is reasonable to believe that support for MOF2.0-QVT will be available in the near future.

Previously discussed approaches illustrate a point to point interchange between two enterprises. It could be possible to rely completely on MDA technology for achieving less numbers of mappings. Ontology is conceptual model agreed upon by a community. As pointed out in chapter 5, UML2.0, including OCL as part of the language as well as extension capabilities, can be considered an alternative for capturing ontologies implying that ontologies could be constructed using UML instead of e.g. OWL. Given that OWL is considered better suited because of it’s reasoning capabilities and therefore should be used to represent ontologies, ontology captured using OWL could be transferred to UML. This of course presupposes that the technology for transforming between OWL and UML is available. If a community of business partners would agree upon the following three points, the MDA technology could be used to enable interchange of business messages where the number of mappings is $2^n$ instead of $2n$.

- A shared conceptual model or ontology, defining the structure, syntax and vocabulary to be used when exchanging data.
- Making this ontology available as an UML model.
- Using XMI as interchange format when exchanging data at run time.

The following figure illustrates this approach:
Based on currently available documentation on the ontology-based approach, my understanding of the technology applied to enable transformation between two enterprises is reflected in the following figure.

The two previous figures illustrate possible solutions for achieving data interchange between enterprises where semantically similar data is structured and named differently. In both illustrations, enterprises are to exchange data that is structured according to an ontology, which can be viewed as some standard model. The format used to interchange the data is dependent on the transformation engine to be used to execute the transformations. Figure 44 and 45 illustrate an ontology-based approach using RDF as interchange format and a model-based approach using XMI as interchange format. The Ontology-based approach refers to semantic mapping using the terms semantic annotation and reconciliation.

8.2 As complementary approaches

As has been discussed in previous chapter, the goal of carrying out semantic mapping, whether it is an ontology-based approach or a model-based approach, is the same. The goal
is to create a set of executable transformations in order to convert from one way of representing information to another.

The difference lies in the different technology and the different languages used for the different approaches. When using an ontology-based approach, expression mechanisms common within the field of the semantic-web community are used to capture information and semantic-web-related technology is used to enable transformations. During the model-based approach, expression mechanisms defined according to MOF are used and the MDA-related technology is used to enable transformations.

### 8.2.1 Complementary strengths

Different approaches are bound to have different strength and weaknesses. The following table elaborates on the strengths related to each task involved in semantic mapping, with focus on the model-based approach and the ontology-based approach respectively.

When elaborating on the preparation task, I include some points related to an ontology construction or the availability of ontologies, although I have not included an ontology construction as part of the preparation phase.

<table>
<thead>
<tr>
<th>Task</th>
<th>Strengths</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Preparation (PR)</strong></td>
<td>Model-based</td>
</tr>
<tr>
<td></td>
<td>+ MDA defines how systems can be viewed using different abstraction levels as well as having focus on how to enable transformation between different abstraction levels.</td>
</tr>
<tr>
<td></td>
<td>+ OMG is also focusing on reverse engineering of existing systems through the ADM approach.</td>
</tr>
<tr>
<td></td>
<td>+ The two previous points allow bringing the data sources to be mapped into similar context.</td>
</tr>
<tr>
<td></td>
<td>+ The UML modelling language is a widely accepted de facto standard with both graphical and textual notation as well as wide tool support. This must be considered one of the main strengths when considering ontology construction.</td>
</tr>
<tr>
<td><strong>Ontology-based</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+ Within semantic-web community, the focus is on ontologies and currently, number of ontologies are already being built.</td>
</tr>
<tr>
<td></td>
<td>+ Ontology-based approaches have also been addressing what is referred to as data mining, where e.g. business documents are being scanned in order to identify business concepts to be included in ontologies. An example of such approach is introduced by Missikoff et. al. in [41].</td>
</tr>
<tr>
<td></td>
<td>+ Reasoning capabilities such as automatic classification.</td>
</tr>
<tr>
<td>Semantic Coordination (SC)</td>
<td>Model-based</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>+ Algorithms for detecting semantic relationships between different data elements have been developed within the field of data schema integration, ref. [61]. These algorithms could also be implemented using formally defined UML models as input.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ontology-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Usage of lexical databases such as WordNet is common in the semantic-web community when detecting a relationship between terms.</td>
</tr>
<tr>
<td>+ Current research within semantic-web community is concerned with automatically detecting relationships between two local ontologies.</td>
</tr>
<tr>
<td>+ The semantic-web community is concerned with how to precisely describe meaning of terms. The semantic-web related technology is more advanced when it comes to interpreting e.g. semantics with help of analysing field descriptions.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Affirmation (AF)</th>
<th>Model-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Affirmation is where domain expert verifies or rejects detected relationships. The model-based approach focuses on graphical notation both for modelling languages as well as for transformation languages. Graphical notation is considered better suited for humans.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conflict Resolution (CR)</th>
<th>Model-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Ability to define and extend new machine readable languages using MOF.</td>
<td></td>
</tr>
<tr>
<td>+ MOF compliant languages can be source or target for MDA transformations.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Formal mappings (FM)</th>
<th>Model-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ MOF facilities and MDA transformations provide the ability to define abstract languages that can be refined into executable implementations by means of model transformations.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Executable Transformation (ET)</th>
<th>Model-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Transformation engines capable of transforming data sources in XMI format.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ontology-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Rule engines capable of transforming e.g. data in RDF format by means of rule inference.</td>
</tr>
</tbody>
</table>

The table above illustrates that the strength of the ontology-based technology is related to automatically detecting semantic relationships. The MOF facility and the transformation technologies such as MOFScript, provide a framework for defining languages and automatically generating executable transformations. Currently, the approach taken in A3 for generating executable transformations is manual, and therefore no plusses for affirmation, conflict resolution and formal mappings.
8.2.2 Possible communication points

In chapter 6.2.4, I discussed the role of ODM and referred to research work where bridges are being created between OWL and UML, using ODM together with XSLT transformations. It is reasonable to believe that ODM as well as MOF-QVT2.0 will be accepted as standards belonging to the MDA framework in the near future. Given the attention MDA is currently receiving with large tool providers, it is also reasonable to belief that bridges for bringing ontologies into the MDA world will be available. This opens for using MDA based tools for constructing and maintaining ontologies as well as bringing PIMs into the ontology-based world.

In similar way, bridges could be created between different transformation languages used to express semantic relationships between two conceptual models. This would open for the possibility to bring e.g. models into the ontology-based world in order to carry out semantic coordination using ontology-related technology. Further, the information about semantic relationships could be brought back to the MDA world, e.g. in order to refine relationships using transformation operators or to create formal expressions and automatically transform these into executable code.

The MDA transformation technology can be used to transform information between different worlds or different expression mechanisms. Using the MDA technology requires defining a metamodel for the constructs to be used when e.g. expressing semantic relationships using ontology-related languages. Further, transformations need to be defined between the two metamodels in order to exchange information. Having sketched set of tasks that belong to the semantic mapping process, these tasks also become possible communication points where output from one task can be brought over on the other side in order to continue the process.

The following figure is to illustrate the possible communication points. The circles with the abbreviations refer to the different tasks involved in the semantic mapping process. The boxes denote the expression mechanisms used at each point and the letters indicate the different information captured as follows:

- M: for models capturing static structure.
- S: for semantic relationship between model elements.
- R: for refined semantic relationships where transformation operators needed to resolve conflicts have been identified, e.g. merge.

As an example, the output of the semantic coordination will be a language capable of expressing semantic relationships between the different model elements. If these expressions can be defined in a metamodel, they can be exchanged between the ontology-based and the model-based world, by applying model-to-model transformation technology.

The figure is to illustrate a basic idea. When automatically detecting relationship during semantic coordination, it might also be possible to identify transformation operators needed. Models are e.g. also meant to be available as input to all tasks.
Figure 46. Possible communication points – ontology-based vs. model-based

The blue/solid bi-directional arrow is to indicate the “existence” of a bridge, here by means of ontology definition metamodel (ODM) and ontology UML profile (OUP). This enables e.g. two PIMs to be brought into the ontology-based world in e.g. order to carry out semantic coordination using the ontology-based technology.

The dotted lines indicate transformations to be addressed if information about semantic relationship is to be brought over from e.g. the ontology-based world to the model-based world in order to affirm automatically detected relationships.
8.3 Evaluating approaches

In this chapter, the different approaches to semantic mapping are evaluated using the requirements for semantic mapping identified in chapter 4.

8.3.1 Functional requirements – expressiveness

The following table presents evaluation of whether the expressiveness of the languages used during the model-based approach and the ontology-based approach, fulfil the functional requirements identified in chapter 4.1. The following evaluation criteria have been used:

- $\div$ Not supported
- $+$ Partially supported
- $++$ Fully supported

<table>
<thead>
<tr>
<th>Requirement-ID</th>
<th>Ont.</th>
<th>MDA</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR1-ML-Metadata</td>
<td>++</td>
<td>++</td>
<td>Ontologies and PIM/UML models carry information about structure and naming of real-world concepts to be mapped</td>
</tr>
<tr>
<td>FR2-TL-StringFunctions</td>
<td>++</td>
<td>++</td>
<td>Both approaches support these functions.</td>
</tr>
<tr>
<td>FR3-TL-Math</td>
<td>++</td>
<td>++</td>
<td>OCL can be used to certain extend (recall table 11) but transformation languages such as MOF2.0-QVT allow for initiating externally defined functions. The ontology-based approach applies jena rules, which allow for initiating functions written in external programming.</td>
</tr>
<tr>
<td>FR4-TL-Logic</td>
<td>++</td>
<td>++</td>
<td>Both approaches provide a mechanism for capturing relationships between model elements in different models. Both approaches use an abstract language to achieve this. Currently, the model-based approach does not support identifying homonyms, but the language being developed can be extended to achieve this.</td>
</tr>
<tr>
<td>FR5-TL-TypeCasting</td>
<td>++</td>
<td>++</td>
<td></td>
</tr>
<tr>
<td>FR6-TL-DataAssignment</td>
<td>++</td>
<td>++</td>
<td></td>
</tr>
<tr>
<td>FR7-TL-SemanticRelations</td>
<td>++</td>
<td>++</td>
<td></td>
</tr>
</tbody>
</table>

As the table above illustrates, the ontology-based approach and the model-based approach are very similar with regards to expressiveness of the languages used during semantic mapping.
8.3.2 General requirements

The following table presents evaluation of the general requirements identified in chapter 4.2. When addressing general requirements, I use evaluation criteria different from what I used for functional requirements. Here I consider whether the different requirements are being emphasised when addressing semantic mapping. The evaluation criteria are as follows:

- Not emphasised
- Addressed
- Emphasised

Table 18. General requirements – emphasis, model-based vs. ontology-based

<table>
<thead>
<tr>
<th>Requirement-ID</th>
<th>Ont.</th>
<th>MDA</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR1-Graphical notation</td>
<td>+</td>
<td>++</td>
<td>Graphical notation for the languages used for the ontology-based approach, are generally not defined. Graphical presentation of information to be mapped is somewhat addressed through tools constructed in the ATHENA project. MDA and the model-based approach emphasise on graphical notation of models.</td>
</tr>
<tr>
<td>GR2-Textual notation</td>
<td>+</td>
<td>+</td>
<td>The ontology-based approach addresses textual notation through OWL and RDF. The model-based approach addresses textual notation through XMI.</td>
</tr>
<tr>
<td>GR3-Familiar context and technology</td>
<td>+</td>
<td>+</td>
<td>UML is a widely accepted de facto standard used for modelling software applications, meaning that software applications supporting e-Procurement are likely, but not necessarily, specified using UML models. The model-based approach emphasises on using models as context during semantic mapping.</td>
</tr>
<tr>
<td>GR4-Platform independent view</td>
<td>+</td>
<td>++</td>
<td>PIMs are to be the context used during the model-based approach to semantic mapping and PIMs present platform independent view of both source and target structure. The ontology-based approach emphasises on platform independency with regard to the ontology but this is not addressed in similar degree with regard to information sources.</td>
</tr>
<tr>
<td>GR5-Method support</td>
<td>+</td>
<td>+</td>
<td>Based on the survey of ontology literature, methods or best practices for how to connect ontology to information sources are limited. MDA does not address horizontal transformations directly. Based on analysis of project documentation, the model-based approach is more technical oriented while the ontology-based approach illustrates a process and documents a method for e.g. how to</td>
</tr>
</tbody>
</table>
8.3.3 Table overview

Following table provides and overview of the results indicated in table 16, 17 and 18, omitting comments and explanation of evaluation criteria. This table is just to serve as an overview and should not be interpreted without reading the explanations provided in the referred tables.

<table>
<thead>
<tr>
<th>Complementary strengths</th>
<th>Ont</th>
<th>MDA</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparation (PR)</td>
<td>+++</td>
<td>++++</td>
<td>Refer to table 16</td>
</tr>
<tr>
<td>Semantic Coordination (SC)</td>
<td>+++</td>
<td>+</td>
<td>Refer to table 16</td>
</tr>
<tr>
<td>Affirmation (AF)</td>
<td></td>
<td>+</td>
<td>Refer to table 16</td>
</tr>
<tr>
<td>Conflict Resolution (CR)</td>
<td>++</td>
<td></td>
<td>Refer to table 16</td>
</tr>
<tr>
<td>Formal mappings (FM)</td>
<td></td>
<td>+</td>
<td>Refer to table 16</td>
</tr>
<tr>
<td>Executable Transformations (ET)</td>
<td>+</td>
<td>+</td>
<td>Refer to table 16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expressiveness</th>
<th>Ont</th>
<th>MDA</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR1-ML-Metadata</td>
<td>++</td>
<td>++</td>
<td>Refer to table 17</td>
</tr>
<tr>
<td>FR2-TL-StringFunctions</td>
<td>++</td>
<td>++</td>
<td>Refer to table 17</td>
</tr>
<tr>
<td>FR3-TL-Math</td>
<td>++</td>
<td>++</td>
<td>Refer to table 17</td>
</tr>
<tr>
<td>FR4-TL-TypeCasting</td>
<td>++</td>
<td>++</td>
<td>Refer to table 17</td>
</tr>
<tr>
<td>FR5-TL-DataAssignment</td>
<td>++</td>
<td>++</td>
<td>Refer to table 17</td>
</tr>
<tr>
<td>FR7-TL-SemanticRelations</td>
<td>++</td>
<td>++</td>
<td>Refer to table 17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Emphasis</th>
<th>Ont</th>
<th>MDA</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR1-Graphical notation</td>
<td>+</td>
<td>++</td>
<td>Refer to table 18</td>
</tr>
<tr>
<td>GR2-Textual notation</td>
<td>+</td>
<td>+</td>
<td>Refer to table 18</td>
</tr>
<tr>
<td>GR3-Familiar context and technology</td>
<td>+</td>
<td>+</td>
<td>Refer to table 18</td>
</tr>
<tr>
<td>GR4-Platform independent view</td>
<td>+</td>
<td>++</td>
<td>Refer to table 18</td>
</tr>
<tr>
<td>GR5-Method support</td>
<td>+</td>
<td>+</td>
<td>Refer to table 18</td>
</tr>
</tbody>
</table>
9. Conclusions and future work

9.1 Conclusions

The goal of this thesis was to investigate how the ontology-based approach could aid the model-based approach in obtaining semantic interoperability.

The conclusion I have gradually come to, through the work of this thesis, is that the model-based approach and the ontology-based approach for solving semantic interoperability, are conceptually very similar approaches. Both approaches use semantic mapping as a mean to address semantic interoperability where semantic mapping denotes a set of tasks needed to relate semantically similar data elements in different data structures as well as to identify techniques for addressing various conflicts. Basically, the difference lies in the different technology and languages used to support the different tasks involved.

The similarity is due to the similar way in which semantic mapping is to be achieved, both regarding the tasks involved as well as the similar context in which semantic mapping is to be carried out.

Conceptual models carry the semantics or the meaning of information to be exchanged and reflect how real-world concepts have been captured in order to enable computers to process them. Both approaches emphasise on using the conceptual model as a context when addressing semantic interoperability. The model-based approach emphasises on using PIMs where PIMs can be viewed as conceptual models independent of platform specific details. The ontology-based approach emphasises on using ontologies as context where ontologies can also be viewed as conceptual models independent of platform specific details.

The difference, in addition to different technology and languages used, is that the ontology is to serve as a reference model needed to reduce number of mappings to be constructed from exponential to linear. The ontology-based approach is to provide a framework for creating such a reference model, but is not to address the process of how to agree upon a shared conceptual model. This is a complex process involving mediation between a number of distributed actors. Both ontology-based languages as well as model-based languages can be used to capture such a reference model or ontology. Whether using a global ontology or not, the semantic mapping is basically about associating semantically similar data elements in two autonomous models as well as identifying how to convert between the different ways of representing them.

This is the conclusion I come to after studying the literature and carefully studying project documentation available, finding answers to the questions I asked my self in the beginning. The different questions are addressed throughout the thesis but following table provides an overview:
## Questions and answers

<table>
<thead>
<tr>
<th>No.</th>
<th>Questions and answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>What is ontology?</td>
</tr>
<tr>
<td></td>
<td><strong>Ontology can simply be viewed as a conceptual model, capturing entities and relationships between them. This conceptual model is to be agreed upon by a community and captured using a formal, machine-readable language. A more thorough answer to this question is provided in chapter 5.</strong></td>
</tr>
<tr>
<td>2.</td>
<td>How is ontology linked to information sources when explicitly defining their semantics?</td>
</tr>
<tr>
<td></td>
<td><strong>Current ontology literature does not provide detailed description regarding how this is to be done, as pointed out in chapter 5.5. An example of how ontologies can be linked to information sources, is the approach used in A3 and referred to as semantic annotation and reconciliation.</strong></td>
</tr>
<tr>
<td>3.</td>
<td>Can MDA support semantic mapping as this indicates mapping between different systems and not between different models of the same system?</td>
</tr>
<tr>
<td></td>
<td><strong>The principles of MDA transformation technology can be applied to support semantic mapping. This is addressed in chapter 6.1 through 6.3.</strong></td>
</tr>
<tr>
<td>4.</td>
<td>Do ontologies and PIMs overlap?</td>
</tr>
<tr>
<td></td>
<td><strong>Yes, ontologies and PIMs do overlap. An example is that UML, commonly used for capturing PIMs can also be used for capturing ontologies. Chapter 5.2 and 5.4 address this further.</strong></td>
</tr>
<tr>
<td>5.</td>
<td>What are reasoning capabilities, and are reasoning capabilities essential during semantic mapping?</td>
</tr>
<tr>
<td></td>
<td><strong>Reasoning is about asking questions and getting answers and can be compared to querying. Different reasoning mechanisms depend on the different modelling constructs used to capture models. According to the literature, available reasoning capabilities for web-based languages such as OWL are limited. This is further addressed in chapter 5.3.</strong></td>
</tr>
<tr>
<td>6.</td>
<td>Can MDA transformation technology support semantic mapping: a) through tool support? b) through graphical notation of languages involved?</td>
</tr>
<tr>
<td></td>
<td><strong>MDA based transformation is getting a lot of attention in the commercial tool market. Support for graphical notation of transformation languages is currently limited. MDA based transformation tools are available and functions needed to support semantic mapping can be provided. This is further addressed in chapter 6.4 and 6.5.</strong></td>
</tr>
<tr>
<td>7.</td>
<td>How do ontology-based and MDA-based approaches to semantic mapping differ, and what are their respective strengths?</td>
</tr>
<tr>
<td></td>
<td><strong>The difference between the two approaches lies in the different technologies used as well as different emphasis. In table 16, the strengths of the different approaches related to different tasks are identified. This table indicates that the ontology-based technology is better suited for automatically detecting semantic relationships and can thereby contribute to the model-based approach.</strong></td>
</tr>
</tbody>
</table>

In this thesis I have illustrated, using figure 39 (Semantic mapping) in chapter 8.1.1, together with figure 46 (Possible communication points) in chapter 8.2.2, how the different tasks that denote the semantic mapping, can serve as entry and exit points for translating between different languages used during the different approaches. Enabling this translation is a necessary condition for being able to utilise the strengths of each approach. Currently, how to transform between OWL and UML, is receiving considerable attention as pointed out in chapter 6.2.2 and 6.2.4.
9.2 Future work

During the work on this thesis, I have identified some problem areas that need further considerations when addressing semantic interoperability. In addition, I have thought of some examples regarding how the model-based approach and the ontology-based could be combined. In this section I will address these issues.

9.2.1 Combining the ontology-based and the model-based approach

The model-based approach and the ontology-based approach to semantic mapping can be combined in various ways. The different available paths in figure 46 (Possible communication points), illustrate different combinations. One reason for combining the approaches can be in order to utilise complementary strengths. Another reason for combining approaches is that one enterprise might prefer the model-based approach to semantic mapping while another enterprise might prefer the ontology-based approach. The next two subsections contain examples of two different ways to combine the model-based approach and the ontology-based approach. Both examples are illustrated using subset from figure 46. The abbreviations were explained in table 13 as well as in chapter 8.2.2, and are as follows:

♦ SC: Semantic coordination - automatically detecting evidence for semantic relationships
♦ AF: Affirmation - the task of confirming/rejecting automatically detected relationships
♦ CR: Conflict Resolution – identifying functions needed to convert between different ways of representing similar data
♦ FM: Formal mappings – specifying the semantic relationships and the techniques needed using a formally defined, machine understandable language
♦ M: Models
♦ S: Semantic relationships
♦ R: Refined semantic relationships, including transformation operators needed to convert between different ways of representing similar data

Exchanging data using ontology-based interchange format

A realistic situation is an interaction between two enterprises A and B, where enterprise A, uses a model-based approach for software development, and enterprise B, uses e.g. software developed by 3rd party. In that case, enterprise A might prefer the model-based approach to semantic mapping while enterprise B might prefer the ontology-based approach to semantic mapping. Given that a global ontology exists, and that the ontology can be translated between the model-based world and the ontology-based world, the data interchange format still needs to be addressed. As pointed out in chapter 8.1.3, the transformation technology depends to certain degree on the data interchange format. Deciding on an ontology-based technology or RDF for data interchange, the model-based transformation language, could be transformed to ontology-based transformation language prior to creating executable transformations. This could be achieved by applying e.g. model-to-text transformations using MOFScript. This is illustrated in the following figure, a subset of figure 46.
Another realistic situation is when two enterprises apply model-driven approach for software development and want to interact using the model-based technology. It can further be assumed that the ontology for the relevant domain is either captured using UML or has been transformed from OWL to UML. Given that the ontology-based technology will provide better results when automatically detecting semantic relationships between two models, the semantic coordination should be carried out using ontology-based approach. This implies that the two models, captured using model-based languages are to be transformed to ontology-based languages using ODM and model transformations. The ontology-based technology for detecting relationships between two models is then applied. Thereafter, the information about the semantic relationships detected, needs to be transformed back to the model-based world in order to confirm or reject the detected relationships, involving ADM principles or text to model transformations. This is illustrated in the following figure.

Detecting similarities using ontology-based technology

What this implies could also be illustrated using following use-case. Here, the domain expert initiates a service for automatically detecting relationships between two models. The model-based service uses another service implemented using ontology-based technology to carry out the task. The model-based technology is then used to affirm detected relationships where
the user is presented with a graphical representation of the models as well as the relationships between them.

![Graphical representation of models and relationships](image)

**Figure 49.** Utilising complementary strengths illustrated with a use-case

### 9.2.2 Data types

Data types play important roles when dealing with interoperability. Using different data types when representing semantically similar information, results in what in chapter 3.2, was referred to as type conflict. What kind of a typing system to use becomes a challenge when creating PIMs. UML is the language for expressing some platform independent model, but UML does not have any typing system. According to [63], the intention is to use the data types provided by the implementation platform, where these should be made available through application of profiles. This means, that although UML can be used as a language for expressing platform independent concepts, the language lacks a platform independent typing system.

OWL can be considered as a platform independent language for capturing real-world concepts. OWL uses a typing system as defined by XSD [77].

It could be an interesting research area to look further into how data types should be handled in PIMs e.g. if ontologies where constructed using UML.

### 9.2.3 Reasoning capabilities

In this thesis, the focus has been on structural aspects of the information to be exchanged. In order for enterprises to inter-operate, behavioural aspects and non-functional aspects need to be communicated as well. The usefulness of reasoning capabilities of ontology-based languages is associated with automatic classification of services as well as e.g. detecting whether constraints are being satisfied, something that might be very useful when modelling non-functional aspects.

Therefore, the strength of ontology-related technology for addressing semantic interoperability related to behavioural and non-functional aspects, should be further investigated.
9.2.4 Usability tests

The task of relating semantically similar terms will never be fully automated. Semantic mapping needs to be supported with documented methods and tools that support the users when carrying out this complex and error-prone task. The knowledge needed to address semantic mapping, will require different expertise. The domain expert knows more about the domain being addressed and thereby which data elements are semantically related, provided that he has the necessary information available. The domain expert is also more fitted to affirm automatically detected semantic relationships than a software engineer or a knowledge engineer. An engineer on the other hand, has the expertise needed to construct functions needed to convert between different ways of representing semantically related data. The software engineer has more knowledge regarding the model-based approach and the knowledge engineer has more knowledge regarding the ontology-based approach.

Despite the technology being used, the usability of the tools that support the task of semantic mapping, will determine their success. Comparing the usability of the different tools being developed addressing model-based and ontology-based approach respectively, could be an interesting study.
10. Reference


Computational Intelligence and Information Technologies, Nis, Serbia and Montenegro, 2003, pp. 51-54.


Appendixes
A. Different ontology definitions

A.1 Ontology explained using natural language

Some definitions use natural language as analogy, when defining ontology. Ontology is explained as the mechanism for providing the vocabulary and semantic structure in order to exchange information. The following definitions reflect this:

“An ontology defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary.” [44]

“The goal is to create an agreed-upon vocabulary and semantic structure for exchanging information about that domain.” [72]

What is ontology when viewed as a vocabulary and the semantic structure needed to exchange information? A vocabulary is simply a list of all the words that can be used in a certain language or domain for that matter. Semantics is about adding meaning to the words. Maybe we can view semantics as a translator. If we don’t understand the meaning of the words in a certain language, we can use a mechanism to translate the words from the unknown language to the language we do know. The dictionary is such a mechanism, adding semantics or meaning to the words. What dictionary does is that it explains the words, either by explaining the words in different language or using different set of words within the same language.

Vocabulary is quite understandable but semantic structure is somewhat more complex. Ontology is to provide both, so in order to better understand what semantic is, we take a look at the rules of natural language. W. Nelson Francis [20] explains how language is composed of five systems or parts. Each part has its own organization but the parts are also related. These five parts are:

- **Vocabulary**: a collection of words or word parts being the most loosely organized system, and therefore the most open to change.
- **Grammar**: a set of rules for how the words can be combined together in a sentence
- **Semantic system**: complex system that relates the words to things, events and ideas that we talk about by a system of meanings.
- **Phonology**: controls how words are translated into sounds that can be spoken and heard.
- **Writings System**: the most recent one, providing either symbols for each word (Chinese) or a set of characters based on the phonological structure.

Further he states that the relations are complex but that they might be suggested in the following diagram. Here the system of language is inside our mind but it provides input and output interfaces to the real-world:
A.2 Ontology explained using logic

As mentioned in chapter 5.1.3, knowledge based systems use inference rules or goal statements to aid in reasoning. Ontology provides the definition of concepts that reasoning is done about, and is needed in order to be able to reuse or share the work that is put into creating a knowledge base.

The Prolog example presented in chapter 5.1.3 is based on symbolic logic. Jisowa’s writings [69] can be consulted in order to get a better understanding of what ontology is supposed to provide, using symbolic logic as context.

“In logic, the existential quantifier $\exists$ is a notation for asserting that something exists. But logic itself has no vocabulary for describing the things that exist. Ontology fills that gap: it is the study of existence, of all the kinds of entities – abstract and concrete – that make up the world. It supplies the predicates of predicate calculus and the labels that fill the boxes and circles of conceptual graphs. The two sources of ontological categories are observation and reasoning. Observation provides the knowledge of the physical world, and reasoning makes sense of observation by generating a framework of abstractions called metaphysics.”

The rules, also referred to as inference rules, are constructed with the help of logic. Formal logic can assert concepts without really telling what kind of concepts are dealt with. Those can be just X’s and Y’s, e.g. "If X is the father of Y, or X is the mother of Y, then X is a parent of Y."

It can be said that the ontology is implicitly contained within the knowledge base as being the X’s and the Y’s. But the terms or concepts that X’s and Y’s are representing are to be explicitly agreed upon at different level, in order to understand that the X’s and Y’s in my knowledge base are the same as the A’s and B’s in yours. That would be a step in being able to reuse or share the inference engine, the rules and the facts.
B. Architecture of ontologies - different levels

In [44], Neeches et al. envision the future architecture of ontologies. They hope that knowledge based systems will be able to provide a framework for top-level abstraction hierarchies, laying down the ground rules for modelling a domain.

The essential idea is that knowledge bases are to consist of layers of increasingly specialized knowledge. The top hierarchies that are application independent are what Neeches et. al refer to as ontologies. They mention that application systems commonly contain different levels of knowledge but that top levels are often implicitly included in “today’s” systems.

“When these top-level abstraction hierarchies are represented with enough information to lay down the ground rules for modelling a domain, we call them ontologies.” [44]

Less specialized knowledge, residing at higher levels, is easier to share and reuse according to Neches et al. Authors also state that knowledge at lower levels can only be shared if other systems accept the models in the above levels. A higher level also represents commitments to provide services to applications that are willing to adopt to their model of the world.

In [24], authors present two different ways for classifying ontologies and provide a short description of different approaches. In addition to the classification based on different abstraction levels they mention one approach where ontologies are classified based on the “richness of their internal structure”. By that, authors are referring to the representation mechanisms used for describing relations between terms. This classification results in linear categorisation, ranging from controlled vocabulary such as catalogues at one end, and ontologies using first order logic to express relations between terms at the other end.

A classification based on the “subject of conceptualisation” or abstraction level, seems to be more general approach and confirms to the vision of future architecture described in [44]. In [24], authors provide several examples of proposals where abstraction level is used as topology. Authors also provide an overview of different types of ontologies that can be identified from the literature, based on abstraction level as classification. The following is an overview of the different ontology types as presented in [24]:

- **General or common ontologies**: Ontologies representing common knowledge reusable across domains. This includes vocabulary related to time, causality and different units. In [44], authors refer to this level as “Topic-Independent fundamental models”.

- **Top-level ontologies or Upper-level Ontologies**: Ontologies that provide the concepts to which root elements of other ontologies should be linked. Examples of concepts define in these ontologies are objects, events, properties. According to [24], the problem is that there are already several proposals for such ontologies following different philosophical criteria.

- **Domain ontologies**: Ontologies that are reusable in a given specific domain such as engineering, enterprise, automobile etc. The concepts and their relations defined in these
ontologies are normally specialisation of the concepts already defined in top-level ontologies.

- **Task ontologies:** Ontologies describing the vocabulary related to generic task or activity such as diagnosing, scheduling, selling etc.
- **Domain task ontologies:** Ontologies that are reusable within a given domain but still application independent. A domain task ontology concerning trip schedule would include concepts such as next city, previous city etc.
- **Method ontologies:** Ontology that defines relevant concepts and relations in order to achieve a particular task needed in order to specify a reasoning process.
- **Knowledge Representation (KR) ontologies:** KR ontologies capture the representation primitives used to formalise knowledge using the different knowledge paradigms. The most representative ontology of this kind is the Frame ontology capturing primitives used in frame based languages.

The following figure is copied from [24] and is to illustrate the usability reusability trade-off problem when it comes to reusing existing ontologies. Apparently, more general ontologies are easier to reuse but more is gained by being able to reuse more specific ontologies. This illustration also gives an overview of the architecture as envisioned in [44] where ontologies are layered in terms of abstraction level, from general to increasingly specified knowledge.

**Figure 51.** Different ontology layers – Ontology architecture
C. Methods and concepts addressing ontology construction

C.1 Methodology

In [24], authors provide an overview of different approaches that have been used and describe the methods used in the process. The approaches being addressed are mentioned here without further explanation. What should be noted is that the available methods are description of single approaches. Identifying best practices from several successful approaches seems to be something that is yet to be achieved in the field of ontology construction.

♦ The CYC method
♦ Uschold and King’s method
♦ Grüniger and Fox’s methodology
♦ Methontology
♦ SENSUS based method
♦ On-To-Knowledge

In [24], authors compare the different approaches and summarise this in four tables addressing the following:

♦ **Construction strategies** – comparing the following:
  ♦ Types of life cycles used (Incremental or Evolving prototypes).
  ♦ Whether methods for constructing ontologies are independent of the applications using them.
  ♦ Whether core ontologies can be used as starting point.
  ♦ Which strategy is used for identifying concepts (bottom-up, top-down or in-between).

♦ **Technological support of the approaches** – identifying whether and which tools exist in order to support different methods and methodologies.

♦ **The ontology development process** – analysing whether the different approaches identify and describe different activities. The following activities are identified
  ♦ Management activities
  ♦ Development oriented activities
  ♦ Support activities.

♦ **Usage of the defined methods and methodologies** – analysing whether the different methods as well as the ontologies constructed by applying the methods, have been used in different context, such as by other projects or external organisations.

The comparison of the different approaches reveals a wide diversity of in which degree the different criteria’s above are being addressed. The comparison also reveals the fact that most approaches focus mainly on development activities and pay less attention to other important activities such management and evolution of ontologies. According to [24], this is due to the fact the ontological engineering field is still in its childhood.
In [79], authors agree to this by stating that during their survey of existing approaches of an ontology-based integration of information, they found a striking lack of adequate methods for supporting the process of both developing as well as using ontologies.

C.2 Ontology integration and merge

When making use of already existing ontologies the literature has used the term “integration” to encompass several things, including integration of existing ontologies, mapping between different ontologies addressing same domain as well as integrating ontologies in information systems using them. This is pointed out in [58] where authors give examples of different ways using the terms in the ontology-based approaches. The authors suggest use of three terms to distinguish between different application areas and approaches. The terms suggested are integration, merge and use. Here I address the terms integration and merge with similar semantics as suggested in [58]. How to use ontologies is discussed in chapter 5.5.2, using the term mapping.

C.2.1 Integration

The term integration refers to an approach where several ontologies are being reused as a part of resulting ontology. The ontologies to be integrated are usually addressing different domains and the resulting ontology should be such that there is no similar ontology already built. The following figure is to illustrate this approach where a resulting ontology O has been constructed by integrating several ontologies (O1, O2, …, On) addressing domain (D1,D2,…,Dn), as well as augmented with new concepts.

![Figure 52. Illustrating ontology integration](image)

The concepts belonging to the integrated ontologies can be used in different ways such as:

♦ Used as they are
♦ Adapted or modified
♦ Specialized, leading to a more specific ontology on the same domain
♦ Augmented by new concepts

A set of operation is needed in order to enable this integration process. These operations can be viewed as composing, combining, or assembling operations needed to support the different usage of concepts. Authors of [58] point out that, although some operations have been defined and described in documented approaches, a larger set of integration operations needs to be identified, specified and defined.
C.2.2 Merge

The term *merge* addresses the approach where different ontologies (O1, O2, … , On) addressing same or similar subject S, are unified in order to create a single ontology O. Authors of [58] distinguish between two types of merge, *unification* and *alignment* where alignment allows concepts from the source ontology to be identified and unification does not. The following figure illustrates the merge approach using alignment.

![Diagram illustrating ontology merge and merge-alignment](image)

**Figure 53.** Illustrating ontology merge and merge-alignment

The approach or methodology for how to conduct merging of ontologies is lacking according to [58]. The same concerns are also expressed in [24] and [79]. In [24], authors describe few approaches for ontology merge stating that ONION is the most elaborated one. The different methodologies, mentioned in chapter C.1 do not address approaches for doing integration and merge according to [24].
D. Model-based mapping – MOFScript example

An example of MOFScript, taking an instance of UML2UML metamodel as input (referred to in thesis as SMM) and produces XSLT transformation based on information captured by model.

textmodule uml2uml2xslt (in uml:UML2UML)

uml.UML2UMLMappingRoot::main()
{
  file("C:/workspaces/3.1Final/runtime-workspace/newStart/example.xslt")
  <%<xml version="1.0" encoding="ISO-8859-1">
  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      self.nested->forEach(m:umlMapping){
        m.createMapping(""
      }
    tab(1)</xsl:template>
  </xsl:stylesheet>%
}

uml.Mapping::createMapping(location:String)
{
  var elems:list

  stdout.println(self.inputs)

  self.inputs->forEach(n:uml.NamedElement){
    if(n.oclIsTypeOf(uml.Class)){
      location = location + "/" + n.name
    }
    if(n.oclIsTypeOf(uml.Property)){
      elems.add(n.name)
    }
  }

  self.outputs->forEach(n:uml.NamedElement){
    if(n.oclIsTypeOf(uml.Class)){
      <%<%>n.name<%>%
      nl(1)
    }
    if(n.oclIsTypeOf(uml.Property)){
      if(self.helper.in2out.oclIsTypeOf(uml.ConcatOperation)){
        n.writeConcat(location,elems)
      }
      //or other operations...
    }
  }
}
self.nested->forEach(m:uml.Mapping){
    m.createMapping(location)
}

self.outputs->forEach(n:uml.NamedElement){
    if(n.oclIsTypeOf(uml.Class) or n.oclIsTypeOf(uml.Property)){
        <%<%>n.name<%>%>
        nl(1)
    }
}

//this is an example of the logic behind a concat mapping
uml.Property::writeConcat(location:String,elems:list){
    var expr:String
    <%<%>self.name<%>%>
    elems->forEach(n:String){
        <%<xsl:value-of select="" location <%/%> n <%"/>}%>
    }
    nl(1)
}