Framework for the Interoperability of Software Engineering Metamodels

Muhammad Atif Qureshi

A thesis submitted for the degree of
Doctor of Philosophy
University of Technology, Sydney

July 2014

Declaration

This is to certify that this thesis comprises my original work toward the Doctor of Philosophy degree and due acknowledgement has been made in the text of all other materials used. The work has not been submitted previously, in whole or part, to qualify for any other academic award. Any editorial work, paid or unpaid, carried out by a third party is acknowledged.

Muhammad Atif Qureshi

(July 2014)

Dedicated to my mother. At her knees my education commenced and to her I owe all that I am or hope to be.

Abstract

A model, represented as a concrete artefact, is an abstraction of reality according to a certain conceptualization. A model can support communication and analysis about relevant aspects of the underlying domain. A model must be expressed in some language and such languages are defined using metamodels. Many metamodels have been proposed and used in the software engineering literature. Some define modelling languages that are general in nature but the literature of modelling is dominated by domain-specific modelling languages or metamodels. Most of these metamodels have been developed independently from each other and any shared concepts are only accidental. Widespread adoption of these metamodels is hindered by differences between metamodels' concepts. Using more than one modelling language during software development requires some sort of interoperability between the metamodels of those modelling languages. This interoperability is also required to allow mappings between models developed using different modelling languages.

These metamodels are not static in nature and are continuously evolving. This evolution has increased their size and complexity over time. This complexity increases when more than one metamodel is used during software development. Interoperability of a pair of metamodels can reduce their joint size and complexity (elaborated in detail in Chapter 7). The need for interoperability between metamodels is also raised by many research communities.

In this thesis, we have developed a framework that can be used for metamodel interoperability. The framework compares metamodel elements based on their syntax, semantics and structure. The semantics of metamodel elements are further investigated for *linguistic* and *ontological* semantics.

Since terms such as *interoperability*, *bridging*, *merging* and *mapping* have all been used, often loosely, with reference to metamodel compatibility, we will define these terms under the generic term *harmonization*.

Metamodels share some similarities with other domains, e.g. ontologies and schemas. In this thesis, we have also explored the techniques available in these domains that might be useful for metamodel interoperability. We have applied our framework to different metamodels and have shown how metamodels can be used in an interoperable fashion.

The results achieved are analysed and we have shown how interoperability of metamodels can reduce their size and their joint complexity, hence making them easier to understand and use.

Acknowledgement

Firstly, my greatest regards to the Almighty Allah for bestowing upon me the blessings, throughout my life, that make me able to achieve this milestone.

There are no words I could possibly write to articulate my gratitude to my mother, whose never ending prayers followed me everywhere and gave me the courage to face the complexities of life and complete this project successfully.

I must express my gratitude to my wife for her continued support and encouragement. Completing this work would have been all the more difficult without her support. She was undoubtedly the best partner I could have had during the ups and downs of my journey. Also, special thanks to my daughters, who always give me a new hope and strength.

I am immeasurably proud of my brothers and my sisters. I am grateful to them, for all their love, care and everlasting support in these many years of physical distance.

I would like to thank my supervisor, Prof. Brian Henderson-Sellers, for the patient guidance, encouragement and advice he has provided throughout my time as his student. I have been extremely lucky to have a supervisor who cared so much about my work, and who responded to my questions and queries so promptly. I would also like to thank Dr Bruce Howarth for his help regarding proof reading and improving the presentation of this thesis.

This thesis is dedicated to all these people. I could not have completed my research without the support of all these wonderful people!

Publications

Publications arising from this thesis.

1. Qureshi, M. A. (2011), Interoperability of software engineering metamodels. InProceedings of the 2011th international conference on Models in Software Engineering(MODELS'11), Jorg Kienzle (Ed.). Springer-Verlag, Berlin, Heidelberg, 12-19.

2. Henderson-Sellers, B., Qureshi, M. A., and Gonzalez-Perez, C. (2012). Towards an interoperable metamodel suite: Size assessment as one input. International Journal of Software and Informatics, 6(2):111124.

3. Qureshi, M. A. (2012), Interoperability of software Engineering Metamodels: Lessons Learned., inUlrich W. Eisenecker and Christian Bucholdt, ed., Software Language Engineering (Doctoral Symposium), CEUR-WS.org, pp. 3-10.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Research is to see what everybody else has seen, and to think what nobody else has thought.

Albert Szent-Gyorgyi

Nowadays, models are used in software engineering as first-class entities, in contrast to their early usage as a means to communicate among stakeholders (Bézivin, 2004). Treating models as first-class entities means that models can, in principle, be executed directly rather than forming a basis for code (Jiang et al., 2007; Koch and Kozuruba, 2012). This change of focus, from code to models, within software engineering has been recognized as model-based software development (MDSD) (Atkinson and Kühne, 2003) where software development is based on the creation and use of models. Modelling, thus, is treated as an act of creating models using a modelling language (Favre, 2004). Metamodelling, on the other hand, is a specific type of modelling that defines and formalizes a modelling language to express models

for some system under study (Gašević et al., 2007; González-Pérez and Henderson-Sellers, 2009; Henderson-Sellers, 2012).

Many metamodels have been proposed in software engineering to cover different aspects of software development. Most of these metamodels were developed independently with very few commonalities. This lack of commonality is seen as a barrier to their wide-spread adoption (Kappel et al., 2006). Also, metamodels are increasing in size and complexity (Recker et al., 2009b; Siau and Cao, 2001) and this complexity increases when several metamodels are used together (as shown in Chapter 7). Therefore, there is a need to formulate ways in which these metamodels can be used in an interoperable fashion. This thesis is a contribution to the interoperability of metamodels in software engineering.

This chapter presents the background of the thesis (Section 1.1) and shows the relevance of the work (Section 1.2). Section 1.3 defines the main objectives of this research and the scope of the work is presented in Section 1.4. The approach followed to carry out this research, together with an overview of the thesis structure is presented in Section 1.5.

1.1 Background

For several decades, modelling has been a vital part of software development and different approaches to modelling software systems have emerged. Examples are flow charts (Gilbreth and Gilbreth, 1921), state charts (Booth, 1967), ER models (Chan, 1976), formal languages such as Z (Spivey, 1998) and B (Lano and Haughton, 1996) and (probably) the most widely used modelling language, Unified Modelling Language (UML) (OMG, 2005).

Models were initially used as sketches (diagrams) to communicate among the stakeholders in software development projects (Bézivin, 2004). Nowadays, models have become the primary artefacts of software development. Models have become first class entities and software development is now heavily dependent on the creation and use of models. As a result, a new paradigm has emerged known as model-driven software development (MDSD, or Model-Driven Software Engineering, MDSE). We will use the generic term, MDE (Model-Driven Engineering), throughout this thesis (see Chapter 2 for details). MDE incorporates many concepts including *metamodels* to define *modelling languages*.

During the last few decades, many domain-specific modelling languages (e.g. ADELFE (Bernon et al., 2003) and Gaia (Zambonelli et al., 2003)) have emerged, as well as several general-purpose modelling languages (e.g. UML (OMG, 2005), BPMN (OMG, 2011a) and SPEM (OMG, 2008a)). The use of diverse domain-specific modelling languages and diverse versions of the same language (e.g. the OMG version of UML (OMG, 2005) and the Eclipse version of UML (Steinberg et al., 2008)) raised the issue of language integration. It is difficult to ensure interoperability between the different models developed using these languages or the interoperability of the different tools (Kappel et al., 2006) used to develop those models. This also increased the accidental complexity by generating a number of overlapping metamodels of those languages and as a result exacerbated the problem of metamodel interoperability (Bézivin et al., 2010). Consequently, the need to organize the collection of metamodels and to have some interoperable or unified metamodels emerged. This need is highlighted by numerous researchers, e.g. (Bettin and Clark, 2010; Bézivin, 2004; Bézivin et al., 2010; Brunet et al., 2006). This scenario also raises the problem of interoperability between metamodelling platforms (Kühn and Murzek, 2005). Mapping of metamodels from one technical space (model-ware) to another (e.g. ontologies) is also a major concern (Favre, 2004).

Consequently, it is evident that interoperable metamodels are more likely to be effective and adoptable.

1.2 Motivation

Metamodelling has been used in software engineering as a basic underpinning of representational approaches (EIA, 1994; ISO, 1998). Suggestions were made (Henderson-Sellers, 1994; Monarchi et al., 1994) that this would be an appropriate means to facilitate the then-required merger of the large number of extant object-oriented modelling notations. In time this led, under the auspices of the Object Management Group, to the creation of a number of metamodels , not only for UML (OMG, 1997) but many others, such as SPEM (Software Process Engineering Metamodel) (OMG, 2008a), SMM (Software Metrics Metamodel) (OMG, 2008b), ODM (Ontology Definition Metamodel) (OMG, 2009a), OSM (Organization Structure Metamodel) (OMG, 2009b) and BPMN (Business Process Modelling No-

tation) (OMG, 2011a).

Although most of these metamodels were endorsed by a single organization (the OMG), nevertheless, they have grown up in *silos* of independent concept sets with shared concepts being only accidental. These metamodels vary in their focus across, for example, process (OMG, 2008a), product (OMG, 1997), organizational (OMG, 2009b) and measurement (OMG, 2008b) aspects of software development.

Another important aspect is that these metamodels are not static in nature. Over the years, new versions have been introduced that typically extend the scope of the focus domain. For instance, the scope of UML moved from object technology to include component technology and, later, execution semantics for code generation capabilities. This leads not only to an increase in size but also, typically, to increased complexity of the metamodel (Henderson-Sellers et al., 2012). This complexity is compounded when more than one metamodel must be used during the life cycle of a software development project. It is not easy for different key players in software development, such as method engineers, methodologists and analysts, to either comprehend or control this compounded complexity.

Hence, it became increasingly clear that for a more widespread adoption of metamodels in the wider software engineering community, a greater degree of interoperability was required (Bettin and Clark, 2010). This need is further endorsed by the rise of industry interest in model driven engineering (MDE) (Bézivin and Gerbe, 2001). Another example is the change in OMG's OMA (Object Management Architecture) to MDA (Model-Driven Architecture). See (OMG,

2001) as well as the various conferences on the topic (Bettin and Clark, 2010).

The above discussion (Sections 1.1 and 1.2) shows that there is a need to formulate ways in which these metamodels can be used in an interoperable fashion. This interoperability can reduce their joint complexity (as we will see in Chapter 7), hence making them easier to understand and use, especially for novice users. The interoperability of metamodels may improve the efficiency of modelling languages by avoiding unnecessary duplication and discrepancies in modelling languages, as well as by increased portability of models across modelling tools, better communication among researchers and focus on improving one (unified) metamodel instead of the many extant metamodels (Beydoun et al., 2009). Therefore this research is focused on finding a way to make these software engineering metamodels interoperable.

1.3 Objectives

This thesis aims at contributing to the topic of metamodels' interoperability, by providing a way in which metamodels can be used in an interoperable fashion. This broader term of interoperability comprises a number of sub-goals on which this thesis is based. To achieve our objectives, the following goals were set.

1. Defining the term *Interoperability* for metamodels. What exactly interoperability means: whether this is a merging of metamodels or is some kind of mapping between them, is addressed in Chapter 4.

2. Identifying the techniques that can be used for metamodel interoperability. This may require the investigation of techniques used in domains similar

to metamodels, e.g. ontologies and schemas.

3. Developing a (semi-automated)framework for metamodel interoperability based on those techniques.

4. Applying the framework to candidate metamodels, mainly focusing on identifying similarities, differences and gaps between metamodels.

5. Evaluating the results and suggesting improvements.

1.4 Scope

To achieve the objectives defined in the previous section, we must assess what degree of commonality exists in these metamodels; whether a straightforward merger of conceptual elements in the metamodels is possible to expand and align the scopes of, say, a pair of metamodels; whether discrepancies and differences in conceptual models require a bridging structure, for instance a mapping between sets; or whether the underlying conceptual models are so very different that any interoperation is virtually impossible, thus suggesting that a combined but newly created and more comprehensive metamodel must be constructed.

The interoperability of metamodels has different aspects. These aspects, terminologically, have often been used loosely, with reference to metamodel interoperability, e.g. *interoperability*, *bridging*, *merging* and *mapping*. Chapter 4 discusses these terms in detail and shows how interoperability of metamodels may result in either *Merging* (Section 4.1.3) or *Alignment* (Section 4.1.4) of metamodels. Both of these aspects of interoperability of metamodels will be addressed in

this thesis. During the course of this research, it has become apparent that *Matching* (discussed in Section 4.1.1) is the most important operation in either approach to metamodel interoperability (Merging or Alignment), so the focus will be more on *Matching* in the following chapters. Matching of metamodel elements depends on finding the *same* or *similar* elements in metamodels, so the objective will be to find different types of similarities among metamodels elements, especially focusing on their syntax, semantics and structure.

Primarily, this thesis focuses more on the abstract syntax of metamodels rather than their notation, which is a separate area of research related to the field of semiotics (Siau and Tian, 2009). Similarly, the integration of different modelling tools that can be used to develop different type of models is not addressed in this thesis.

1.5 The Approach

The approach followed to achieve the objectives of this research is based on design science research (Hevner and Chatterjee, 2010). The design science paradigm has its roots in engineering and the sciences of the artificial (Simon, 1996). It is fundamentally a problem-solving paradigm. It seeks to create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, and use of information systems can be effectively and efficiently accomplished.

The approach we have followed in this research is based on the activities recommended as guidelines of design science research (Hevner

and Chatterjee, 2010). The structure of this thesis reflects these activities as explained below.

- Design as an Artifact: design science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation. The outcome of our research is viable in terms of the artifact (The interoperability framework) that we have developed as a part of our research.

Chapter 5 presents the interoperability framework that we developed for metamodels. This framework is based on some of the techniques discussed in Chapter 4 but is novel in many aspects. We extend some existing techniques (necessary for the domain of metamodels) but we have also used some new techniques for metamodel interoperability. Different types of similarities, as a part of the framework, are also discussed in detail.

- Research Rigor: Design science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.

We have rigorously investigated the literature of similarity techniques for models and metamodels (Section 4.2), ontologies (Section 4.3), schemas (Section 4.4) and business process models (Section 4.5). We found that different types of similarities (e.g. syntactic, semantic and structure) must be sought between a pair of metamodels.

In Chapter 4, Section 4.1, we discussed under the heading of *Harmonization* how to define the various aspects that comprise interoperability. We then explain that metamodel interoperability can be achieved through either *Alignment* or *Merging* or a combination

of them (Figure 4.4). The *Alignment* of metamodels may require *Bridging* (discussed in Section 4.1.5) and/or *Mapping* (see Section 4.1.6) of the metamodels' conceptual elements. It has been shown that finding similar elements between a pair of metamodels through *Matching* (Section 4.1.1) is of paramount imoportance for metamodel interoperability.

- Design Evaluation: The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.

Chapter 6 demonstrates the application of the interoperability framework in some exemplary metamodels. These examples illustrate the approach defended in this thesis, not only syntactic but also the semantic and structural interoperation of metamodels. We demonstrate the application of the framework on five different examples. Section 6.1 shows the application of the framework to BPMN and OSM, both proposed by OMG. In the next section, an extended application of the framework to some multi-agent system (MAS) metamodels is shown and the merging of ADELFE, Gaia and PASSI and then merging of MaSE and Prometheus (Section 6.3) is demonstrated. The merging of a UML class diagram metamodel with ECore is presented in Section 6.4. Finally, to show the usefulness of the merged metamodels , we demonstrate a small example (Section 6.5) of BPMN and SPEM where one metamodel alone was not sufficient to provide modelling constructs to model an example scenario.

Chapter 7 discusses some considerations about the work presented in this thesis. Section 7.2 discusses the quality measures that were used to assess the interoperability framework. Also, improvements

in terms of size and complexity of metamodels are shown in Section 7.3. We show how the interoperability of a pair of metamodels can reduce their size and joint complexity after applying this framework. Chapter 8 concludes this thesis and sheds light (Section 8.1) on some of the limitations and opportunities in applying the interoperability framework to different metamodels, and some future research directions are discussed in Section 8.2.

In the next chapter, the core literature on metamodelling will be reviewed. The notion of model (Section 2.1), modelling language (Section 2.2) and metamodel (Section 2.3) will be discussed in detail. This discussion will serve as the basis for the following chapters. The purpose of this discussion is not to provide some new definitions of these concepts, but to consider some of the definitions available in the literature and to choose (reasonably) the ones used throughout this thesis.

Chapter 2

Model-Driven Engineering

Model Driven Engineering is a subset of system engineering in which the process heavily relies on the use of models and model engineering.

Jean-Marie Favre

The objective of this chapter is to review MDE (Model-Driven Engineering) as a paradigm and to establish basic understanding of some important conceptual terms, e.g. models, metamodels and modelling languages. The relationships among them are discussed in detail as these concepts are cornerstones of the work developed throughout this thesis. We start, in Section 2.1, by discussing models and their various types, e.g. descriptive models, specification models, type models and token models. Models are expressed using modelling languages, so in Section 2.2, the relationships between models and modelling languages are discussed. A modelling language is de-

fined using a metamodel, so Section 2.3 elaborates metamodels and Section 2.4 discusses two important types of metamodelling, *linguistic* and *ontological*. Understanding these types of metamodelling is important as the framework that we propose (Chapter 5) compares two different types of semantics (linguistic and ontological) for metamodel elements. We will see in later chapters (e.g. Chapter 5), how these dimensions affect the interoperability of a pair of metamodels. Section 2.5 concludes the discussion.

Model-Driven Engineering (MDE) is a generic term that can be used for any engineering discipline that relies on the creation and use of models. Model-Driven Software Engineering (MDSE) (Brambilla et al., 2012), often referred to as model-driven software development MDSD (Pardo et al., 2010), relies on the use of models and model engineering (ME) (the creation of models (Favre, 2004)) for software development. For simplicity and better understanding, during the course of this thesis, we will use MDE as a generic term for all these interpretations. MDE in software engineering is a result of an important paradigm shift: the shift from code to model, or from objects to models, to be more precise. This shift has changed the principle of *everything is an object* to *everything is a model* (Bézivin, 2004). This change happened when it became clear that models could be used directly in software development (Bézivin, 2004). This change is also apparent in the OMG's (Object Management Group's) transition from object-management architecture (OMA) to model-driven architecture (MDA) (OMG, 2001). In MDE, it is models that are treated as first-class entities rather than code. The following sections discuss some very basic but important building blocks of MDE.

2.1 Models

This section discusses the term *model* , the most fundamental term in the context of MDE. A model, in its simple form, is an abstraction of reality. This abstraction excludes some details of the reality for some specific purpose. Guizzardi and Halpin (2008) formally defined a model as *a conceptualization of reality* for the purpose of understanding. OMG's MDA guide (OMG, 2001) explains *model* as a description or specification of a system for some specific purpose. This definition was further extended by Kühne (2006b), who interpreted a model as *an abstraction of a system allowing predictions or inferences to be made.* Seidewitz (2003) illustrated a model as a set of *statements about the system under study (SUS).* This definition of a model is similar to that provided by Gonzalez-Perez and Henderson-Sellers (2007). The scope of the term *system* in this context is vast and is the primary element of discourse in MDE (Favre, 2004). Kühne (2006b) argued that a system can be either *real* or *language based* while Favre (2004) categorized systems as *abstract*, *physical*, and *digital*. However, some authors like Hesse (2006) and Kühne (2006b) also argued to replace the term *system* with *subject*.

2.1.1 Descriptive and Specification Models

The relationship between a model and the SUS can be (generally) of different types. A model can be created to describe an existing system. For example, a flight simulator is a model of a real airplane that can be used to train pilots. González-Pérez and Henderson-Sellers (2009) referred to this as a *backward-looking* modelling since such models look backward to the actual SUS. This relationship is also

known as a descriptive mode of modelling. In contrast, a model can be used to specify a system that does not exist. An example is an architectural diagram of a building that is yet to be built. This relationship is also known as a prescriptive modelling, since they are used to prescribe or specify the SUS (Bézivin, 2005; Bézivin et al., 2006; Kühne, 2006a; Ludewig, 2003; Seidewitz, 2003). González-Pérez and Henderson-Sellers (2009) referred to these as *forward-looking* modellig. In the descriptive mode, the SUS holds the truth about reality and a model has to conform to that SUS, while in prescriptive mode, the model holds the truth about reality and the system has to be built in conformance with that model. Gonzalez-Perez and Henderson-Sellers (2007) argued that these terms *descriptive* and *prescriptive* do not express any intrinsic property of models but describe their usage (Gonzalez-Perez and Henderson-Sellers, 2007). The authors also emphasised that a model should be homomorphic with the system it presents, meaning that the structure of the model and the structure of the system must coincide to some degree so that operations that are possible on the SUS must also be possible on the model (Figure 2.1).

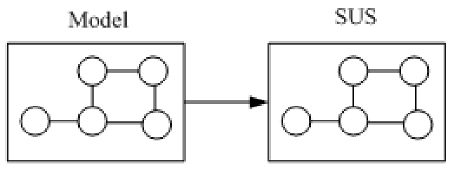

Figure 2.1: Homomorphism between a model and the SUS (González-Pérez and Henderson-Sellers, 2009)

A more generic type of relationship between a model and an SUS is the *representationOf* relationship (Favre, 2004; González-Pérez and Henderson-Sellers, 2009), in which models *represent* the SUS. The *representationOf* relationship is also defined as Equation 2.1 (Favre, 2004). Kühne (2006a) used a formal notion (Equations 2.2 and 2.3) for the *representationOf* relationship between a model and an SUS. He further added that this relationship is many-to-many, which means one model may describe several systems and one system may be described by several models. A detailed discussion on whether an actual SUS is represented by a model or a model is concretized to the original system (O) is highlighted by many authors e.g. Hesse (2006); Kühne (2006a,b).

$$Model = \mu(SUS) \tag{2.1}$$

$$S \lhd M \tag{2.2}$$

$$M = \alpha(S) \tag{2.3}$$

Other types of models, as discussed by (Kühne, 2005) and argued by (Hesse, 2006), are *type* models and *token* models. Kühne (2005) explained token models as models that capture singular aspects of the original elements, e.g. a map (Figure 2.2). In contrast, type models capture the universal aspects of the system by means of some classification (Figure 2.2). Kühne (2005) argued that a type model captures only the types of interest and their associations. Thus *schema model*, *classification model* or *universal model* can be further appropriate names of type models. Although Equation 2.1 can be applied to both

type and token models, Kühne (2005) differentiated them based on their transitivity. The relationship of token models with the SUS is transitive while the relationship of type models with SUS is non-transitive (Equations 2.4 and 2.5) (Kühne, 2005).

$$TokenModel = \mu_T(SUS) \tag{2.4}$$

$$TypeModel = \mu_{NT}(SUS) \tag{2.5}$$

From the above discussion, we can infer some characteristics of a model. A model should be *abstract* and have either a *descriptive* or *prescriptive* relationship with some SUS. In this thesis, we will treat *model* as an abstract representation of some subject or system under study (SUS). We will use the term *representationOf* for the

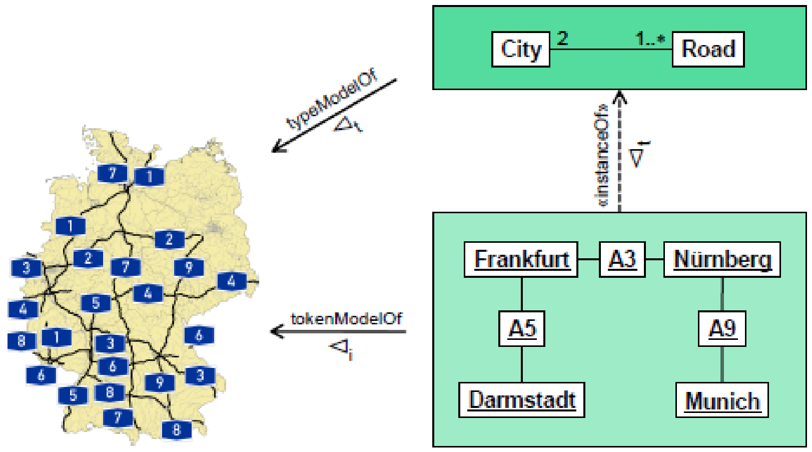

Figure 2.2: Type and token models (Kühne, 2005)

relationship between a model and the SUS. We assume that the *representationOf* relation covers both descriptive and prescriptive modes of modelling, as discussed earlier. Figure 2.3 depicts the relationship between a model and the SUS. The solid arrow represents a relationship between a model and the SUS, while the dotted arrows are just for explanation.

2.2 Modelling Languages

Treating models as *a set of statements* (Seidewitz, 2003) requires a language in which these statements can be expressed. Understanding a model therefore implies understanding the language in which the model is expressed (modelling language). A simple definition of a modelling language could be *a language to express models* (Seidewitz, 2003) but this simplistic view of language does not cover some important characteristics that a modelling language should have.

In linguistics theory (Chomsky, 2000), a language is formalized as a set of sentences, where the notion of set has the same meaning as in mathematics (a collection of items of the same kind). Therefore, Favre (2004) advocated that a modelling language could be seen as *a*

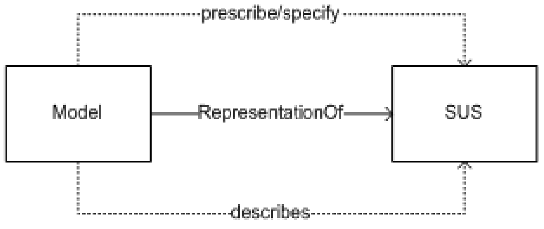

Figure 2.3: Relationship between a model and the SUS

set of models. **The author treated a modelling language as a set with models as elements. Favre (2004) further argued that this definition of a modelling language allows us to treat any natural language as a modelling language. We think that this definition of modelling language is very weak and does not take into account some important aspects of languages. A modelling language, like any other language, has its own *syntax* and *semantics*. Morris (1938), as cited in (Guizzardi and Halpin, 2008), argued that, beside *syntax* and *semantics*, *pragmatics* are also important building blocks of a language. The syntax of a modelling language is categorized as an abstract syntax (e.g. model-unit-kinds) or concrete syntax (e.g. notation).**

Some authors argued that a modelling language should consist of both abstract and concrete syntax (e.g. Fowler and Scott (2000)). Eriksson and Penker (1998) stated that *a modelling language consists of a notation and a set of rules.* However, some researchers like Gonzalez-Perez and Henderson-Sellers (2007) and Greenfield and Short (2004) were convinced that a modelling language should be independent of a concrete syntax or a notation. A modelling language having its own abstract syntax is still valid without having a concrete syntax

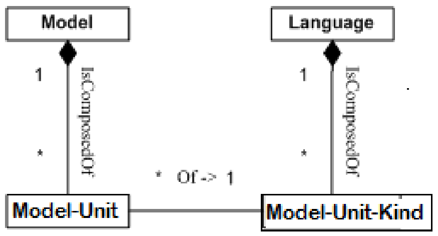

Figure 2.4: Relationship between a language, model and model-unit-kind (Gonzalez-Perez and Henderson-Sellers, 2007)

e.g. (ISO/IEC-24744, 2007). Similarly, a modelling language can use different notations to model the system (see the many-to-many relationship between *Language* and *Notation* in Figure 2.5). For example, UML class diagrams can be shown as a collection of boxes and lines (graphical notation) or as a collection of XML tags (textual notation). Some languages have both abstract and concrete syntaxes defined, e.g. OMG's family of languages like UML.

A more formal, and probably more realistic, definition of a modelling language was given by González-Pérez and Henderson-Sellers (2009): a *model* is described as a collection of *model-units* and every *model-unit* has a *model-unit-kind*. Then, authors defined a modelling language as an organized collection of these *model-unit-kinds* (Figure 2.4).

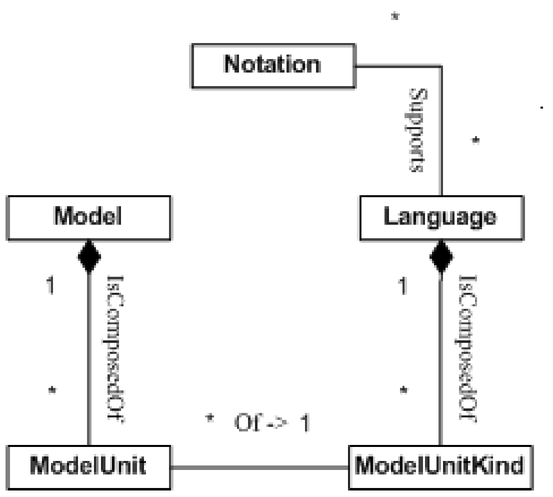

Figure 2.5: The concept of notation (Gonzalez-Perez and Henderson-Sellers, 2007)

Many modelling languages are used as a part of model-driven engineering. Some prominent modelling languages are SysML (OMG, 2012), B (Lano and Haughton, 1996), Z (Spivey, 1998) and UML (OMG, 2005). UML, among these, has become the *de facto*, and perhaps the most widely used, modelling language. The range of different SUSs covered by UML has grown beyond its initial inception in object oriented systems. UML provides not only the abstract syntax of models but also the concrete syntax (i.e. notation) for those models.

An important issue regarding modelling languages is to determine whether a certain model is valid for a given language or not. By considering a modelling language as a *set of models*, (Favre, 2004) could raise the issue of having infinite models in a set. For this purpose we need to determine how to define a modelling language. Also, a prerequisite for treating a modelling language as a collection of *model-unit-kinds* (González-Pérez and Henderson-Sellers, 2009) needs the definition of these *model-unit-kinds*, therefore, of metamodels.

2.3 Metamodels

The term *Meta* is defined as a prefix to indicate a concept which is on a higher level of abstraction than another concept (Oxford, 2012). The prefix *meta* is also used when an operation is applied twice. For example a discussion about how to conduct a discussion is a meta-discussion (Kühne, 2006b). Similarly, Hilbert's work on mathematical proof theory, to make sure that ordinary mathematics could be performed reliably, is known as meta-mathematics (mathematical

methods applied to mathematics itself).

These definitions lead to a wrong interpretation of metamodels in software engineering. Applying modelling twice does not result (necessarily) in a metamodel. If a model is an abstraction of a system then an abstraction of that model cannot be treated as a metamodel. For example, a painting of the solar system is a model of the solar system. A digital photograph of that painting is a model of the painting.

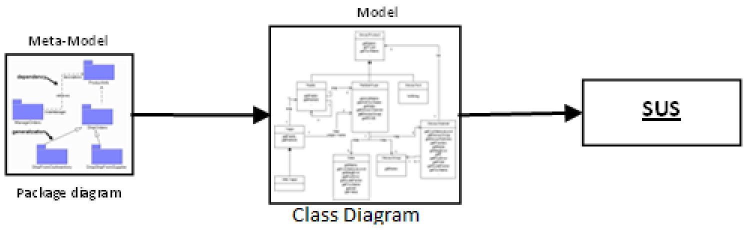

Figure 2.6: Misleading interpretation of a metamodel

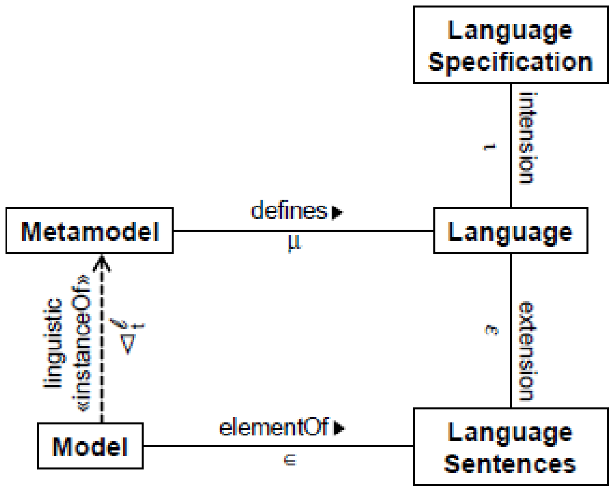

Figure 2.7: Relationships among model, modelling language and metamodel (Kühne, 2006a).

This photograph is now a model of a model (painting), but this does not qualify the photograph to be a metamodel of the solar system. Rather this will result in a *token model* (Section 2.1). Similarly, a class diagram of some system under study can be further abstracted by a package diagram (left hand side of Figure 2.6). This package diagram cannot be considered as a metamodel of the SUS. Kühne (2006a) demonstrated that to create a metamodel we must apply the *type-model-of* relationship twice. The author further illustrated that a model is an instance of a metamodel if it is an element of the set of all sentences which can be generated with the language associated with the metamodel (Figure 2.7).

A metamodel is defined as a *model of models* (Favre, 2004; González-Pérez and Henderson-Sellers, 2009; Seidewitz, 2003). Here, the plural *models* is very important and must not be ignored. Favre (2004) considered that this plural defines a *set* and argued that a metamodel is *a model of a set-of-models.* The author stated that if a modelling language is a *set of models* then it can be said that a metamodel is a

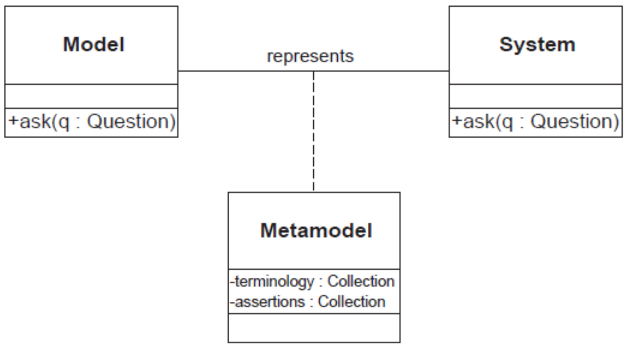

Figure 2.8: Bézivin and Gerbe (2001)'s interpretation of metamodel

model of a modelling language (Favre, 2004). OMG (OMG, 2011b), on the other hand, defined *metamodel* as a model that *defines the language for expressing models.* Treating *metamodel* as a definition of a modelling language has been argued by many researchers and Gonzalez-Perez and Henderson-Sellers (2007) stated that a metamodel *defines the abstract syntax of a modelling language.* Seidewitz (2003) explained *metamodel* as *a specification model for a class of SUSs where each SUS in the class is itself a valid model.* Bézivin and Gerbe (2001) interpreted *metamodel* as the explicit specification of an abstraction that defines a set of concepts and the relations between these concepts. They further added that a metamodel is used as an *abstraction filter* in a particular modelling activity (Figure 2.8).

It should be noted that a metamodel is also a model and all the operations possible on models should equally be possible on metamodels (González-Pérez and Henderson-Sellers, 2009). Models are used not only to specify/prescribe but also to describe the SUS (Section 2.1), so whether a metamodel is used to *specify* or *describe* a modelling language is debatable. Favre (2004) argued that a metamodel should hold both of these characteristics while Gonzalez-Perez and Henderson-Sellers (2007) argued that a metamodel should specify a modelling language using a forward-looking model. These authors also suggested that a UML metamodel is not a specification of UML, but is the UML itself. They treat MOF (Meta Object Facility) as a potential metamodel for UML as MOF specifies UML.

In this thesis, we will treat *model* as a *representationOf* of some SUS. Every model has an *instanceOf* relationship with a metamodel. A

model is *composedOf* model-units and every model-unit has a model-unit-kind. Figure (2.9) is based on Figure 8 in (Kühne, 2006b) and Figure 13 in (Gonzalez-Perez and Henderson-Sellers, 2007). It represents the relationship among models, metamodels and modelling languages.

The *instanceOf* relationship between model and metamodel in Figure 2.9 has been much debated in the literature. Two different types of *instanceOf* relationships (ontological and linguistic) are possible between a model and a metamodel. We will shortly explore these two types of metamodelling. Discussing these relationships requires an understanding that this deliberation started after the OMG's proposal for the MDE infrastructure (OMG, 2001). In the next section we will consider the MDE infrastructure proposed by the OMG and some of the concerns reported in the literature about this infrastructure, and discuss linguistic and ontological metamodelling.

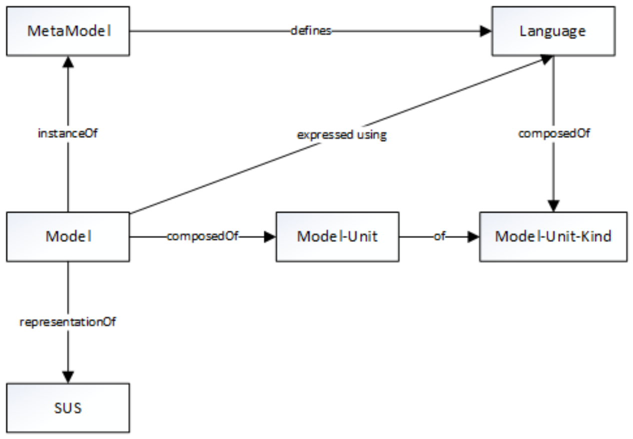

Figure 2.9: Relationship between model, metamodel and modelling language, adapted from (Gonzalez-Perez and Henderson-Sellers, 2007) and (Kühne, 2006b)

2.4 Model-Driven Architecture (MDA)

Figure 2.10 illustrates the traditional model-driven development infrastructure. This infrastructure consists of a hierarchy of model levels. The bottom level, M0, holds the user data (actual data objects the software is designed to manipulate). The next level, M1, holds a model of the M0 user data. User models reside at this level. Level M2 holds a model of the information at M1. Because this is a model of models, it is often referred to as a metamodel. Finally, level M3 holds a model of the information at M2, and therefore is often called the meta-metamodel. Level M3 is also referred as the Meta-Object Facility (MOF).

Each level (except M3) has an *instanceOf* relationship with the level above. This *instanceOf* relationship between different abstraction layers is evidently from the object-oriented domain. The *instanceOf* relationship used in this infrastructure is most likely because of the

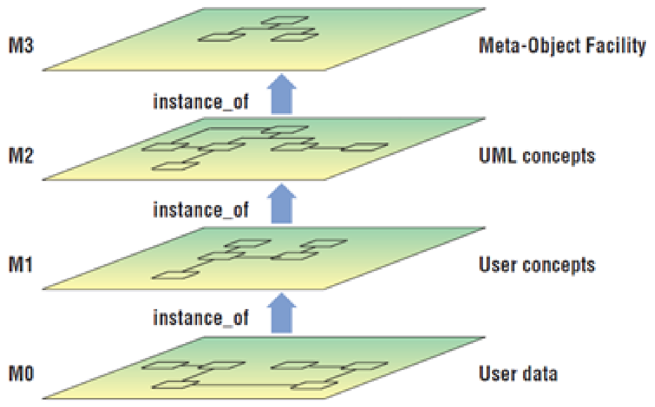

Figure 2.10: OMG four-layer modelling infrastructure, (Seidewitz, 2003)

paradigm shift within OMG from *everything is an object* to *everything is a model*. The OMG's four-layer hierarchy uses the *strict metamodelling* principle defined by Atkinson (1998) (as cited by (González-Pérez and Henderson-Sellers, 2009)) across multiple layers that uses only the *instanceOf* relationship between layers.

Inappropriate use of the *instanceOf* relationship between different layers has been debated extensively in the literature. For example, saying that a model is an *instanceOf* a metamodel leads to much confusion (Bézivin, 2004), discussed below.

Firstly, conceptual entities, in this infrastructure, are presented as *Classes* at different meta-levels and, following the classical object-oriented rules (Haythorn, 1994), *Classes* can be instantiated as *Ob-*

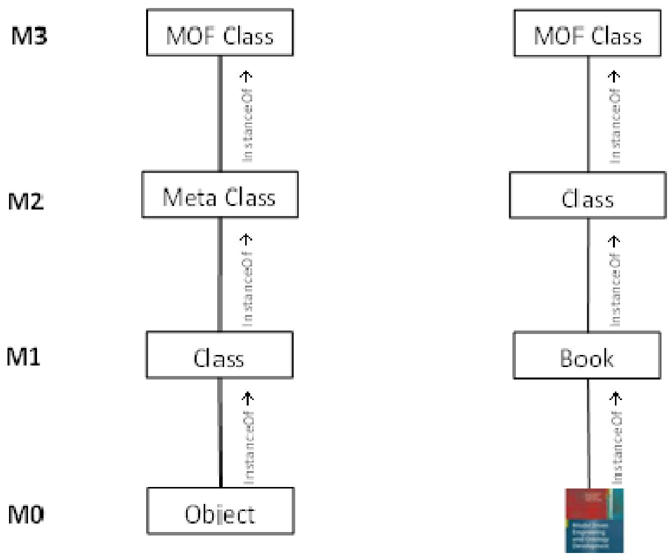

Figure 2.11: *instanceOf* relationship between different levels of models using strict metamodelling

***jects*, but not again as *Classes* (Meyer, 1988). For example, in Figure 2.11 the *Book* class at level M1 has an instance at level M0 which is correct but the same *Book* class itself is said to be an *instanceOf* another class *Class* at level M2, which is not correct.**

Secondly, attributes of any class must be assigned values if that class has been instantiated (Figure 2.12). For example, if a *Book* class has an attribute *name* then *name* must be assigned some value whenever a new object of that *Book* class is created. MDA does not provide this facility in its abstraction layers except for the objects at the M0 level.

An important concern in this regard is that an object should be an instance of one class at a time, but this rule can be easily violated

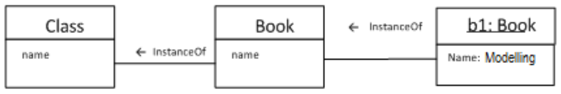

Figure 2.12: Value of an attribute at instantiation

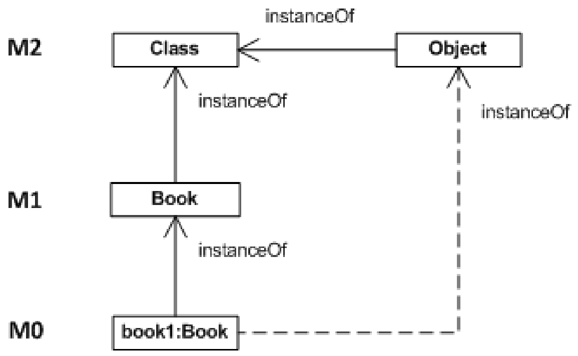

Figure 2.13: Multiple *instanceOf* relationships between class and object

if following the OMG's MDA. Figure 2.13 represents a situation in which *book1* at level M0 is an *instanceOf* a class *Book* at level M1, which is acceptable. However, *Book* itself is an instance of another class *Class* at M2. So, if the *Book* is an instance then it cannot be instantiated again (as discussed earlier). More importantly, *book1* is also an *instanceOf* an *Object* at M2, but *book1* cannot be an instance of both *Book* and *Object* at the same time. Similarly, at level M2, both *Class* and *Object* are classes (as *Object* is instantiated); hence *Object* itself cannot be an instance of a class *Class* at the same time (Atkinson, 1998). This is a clear violation of the strict metamodelling principle. Moreover the *instanceOf* relationship between *book1* at M0 and *Object* at M2 crosses two meta-boundaries, which is also a violation of strict metamodelling (Atkinson and Kühne, 2002).

The problem discussed above arises, possibly, because of the mixture of two different worlds of conceptualisation, *linguistic* and *ontological. book1* is an instance of a *Book* class and also an instance of an *Object* class at M2. The question whether *book1* is a book or an object thus raises the issue that it is both, but from different representations. *book1* is represented by two different entities, which seem to be unrelated. Saying this means that *Book* and *Object* do not belong to the same model. Two different models are involved here, physical and logical, as also highlighted by Atkinson and Kühne (2002). These physical and logical models are also called linguistic and ontological models.

2.4.1 Linguistic and Ontological Metamodelling

The problem discussed in the previous section is caused by using the one-size-fits-all *instanceOf* relationship. It is important to understand the two orthogonal dimensions of metamodelling, *linguistic* (physical) and *ontological* (logical) (Atkinson and Kuhne, 2005). The linguistic aspect is concerned with language definition and hence uses the linguistic *instanceOf* relationship. The ontological dimension is concerned with domain definition and thus uses the ontological *instanceOf* relationship. Both forms occur simultaneously and serve to precisely locate a model element within the language–ontology space (Atkinson and Kühne, 2003).

Any entity or concept, when conceptualized using different layers of abstraction, can have both linguistic and ontological *instanceOf* relationships. For example, Figure 2.14 represents the linguistic (physical) and ontological (logical) metamodelling for *Book*. *Book* is an instance of *Class* and *Book* is an instance of *Product*. Similarly, *Object* is an instance of *Class* and *Class* is an instance of a *MetaClass*. All these relationships are ontological *instanceOf* relationships. In addition, *book1* is an instance of *Object*, *Book* is an instance of *Class* and *Product* is an instance of *MetaClass*. These relationships are treated as linguistic *instanceOf* relationships.

The example shown in Figure 2.14 shows how these linguistic and ontological relationships exist within abstraction levels. Linguistic metamodelling focuses on the alignment of the metalevels with the notion of abstraction (González-Pérez and Henderson-Sellers, 2009). The linguistic *instanceOf* relationship crosses the linguistic metalevels,

while the ontological *instanceOf* relationships do not cross those levels.

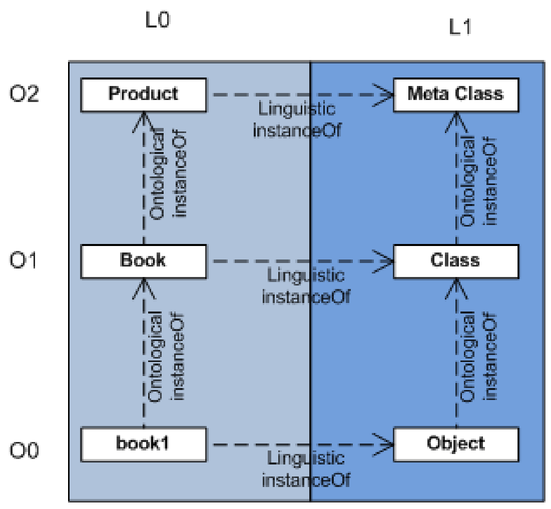

Figure 2.14: Linguistic and ontological relationships, adapted from (González-Pérez and Henderson-Sellers, 2006)

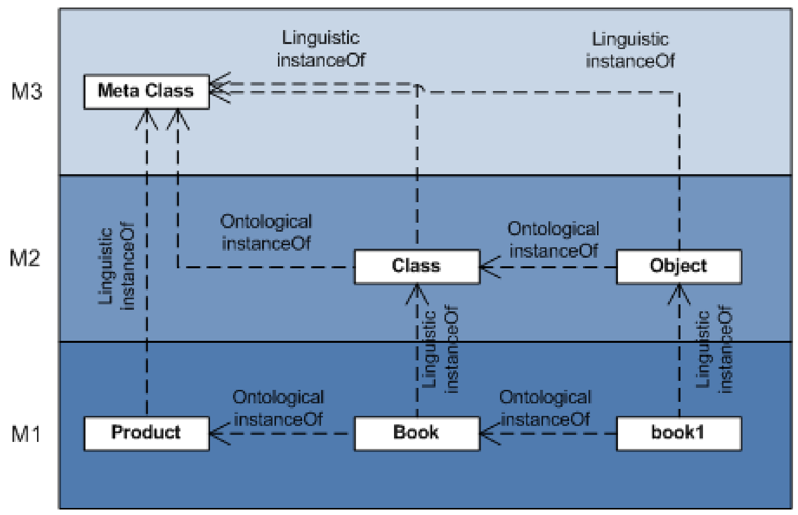

Figure 2.15: Re-arrangements of linguistic and ontological relationships, adapted from (González-Pérez and Henderson-Sellers, 2006)

Applying both ontological and linguistic metamodelling together caused the above problem within the metalevels' hierarchy. Figure 2.15 represents the same relationships, ontological and linguistic, of Figure 2.14 in a rearranged form. We need a *Meta-class* at level M2, just as we have *Object* and *Class* at the same level.

The problem of having an object that is an *instanceOf* two different classes at the same time has been addressed by several researchers in the literature. Johnson and Woolf (1997), for example, suggested decoupling instances from their classes using the *Type–Object* pattern. Pirotte et al. (1994) introduced a *materialization* relationship. Odell (1994) introduced the idea of a *powertype* pattern (see also (González-Pérez and Henderson-Sellers, 2009)). A *powertype* is defined as a type, the instances of which are subtypes of the type. Another type is called the *partitioned type. Powertype* and *partitioned* type are related indirectly through entities that are instances of the *powertype* and at the same time subtypes of the *partitioned type* (Figure 2.16). This means that they are both class and object at the same time, and are called *clabjects* (Henderson-Sellers and González-Pérez, 2005).

The example shown in Figure 2.16 represents the *TreeSpecies* as a powertype of *Tree*, and *Tree* is a type partitioned by *TreeSpecies. SugarMaple* is both a type and an object (*clabject*). We will not discuss powertypes in detail here as they are not the focus of this thesis. Detailed discussion about powertype patterns can be found in (Henderson-Sellers and González-Pérez, 2005).

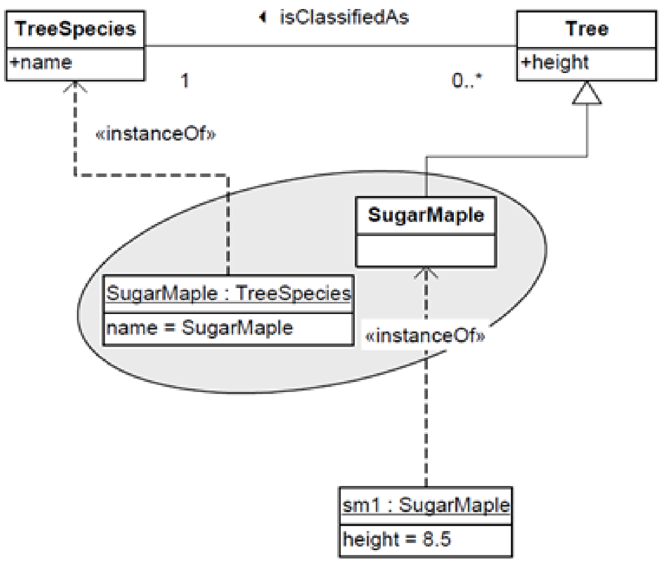

Figure 2.16: Example of powertype pattern (Henderson-Sellers and González-Pérez, 2005)

2.5 Conclusion

In this chapter, we have discussed MDSE and some important concepts used as its cornerstones. We have discussed models (Section 2.1) as representations of some SUS. Models can be either prescriptive/specification or descriptive. Models are composed of different model-units and every model-unit has a model-unit-kind. Models are expressed using a modelling language (Section 2.2) and a modelling language is composed of different model-unit-kinds. A metamodel (Section 2.3) defines a modelling language using specification models. Two different type of metamodelling are known as linguistic (i.e. physical) and ontological (i.e. logical) metamodelling. Although a debate about these two types of metamodelling is not the focus of

this thesis, understanding the linguistic and ontological aspects of metamodelling helps us understand the different types of semantics (linguistic and ontological) that we will match for metamodels (Sections 5.2 and 5.3). We have also discussed the OMG's four-layered model-driven architecture (MDA) and some related problems.

From here onwards, we will concentrate mostly on metamodels, as they are the major focus of this thesis. Metamodels depict some characteristics that are also common in other knowledge areas; especially ontologies. In the next chapter, we will generally explore the role of ontologies in software engineering and the relationship of ontologies with metamodels in particular.

Understanding the relationship between metamodels and ontologies is also important as, we will see in the proceeding chapters, the framework developed in this thesis is based on ontology merging and schema matching techniques.

Chapter 3

Metamodels and Ontologies

Conceptual modeling languages must be rooted in a domain independent, philosophically and cognitively well-founded system of real-world categories, i.e. a foundational (upper-level) ontology.

Luc Schneider

.

In the previous chapter (Chapter 2), we saw that every metamodel depicts not only the syntax of its specific domain of focus but also its semantics. This is especially important when we consider the interoperability of metamodels. To ensure unambiguous semantics, various techniques have been suggested, including the use of ontologies, e.g. (Tran and Low, 2008). Ontologies both improve the semantics of a metamodel (Devedzić, 2002) and provide a potential way in which

these metamodels can be bridged with each other to be interoperable. In this chapter, we will discuss, generally, the role of ontologies in software engineering and their relationship with metamodels in particular.

The term ontology was introduced in the eighteenth century as a general *science of being* (Calero et al., 2006). An ontology can be described as a *philosophical theory concerning the basic traits of the world* (Bunge, 1977). Ontologies in any knowledge area define the basic terms and concepts of the area and their relationship with each other, hence composing the basic vocabulary of that knowledge area (Neches et al., 1991). Ontologies not only allow the sharing of knowledge (Calero et al., 2006) but also clarify the knowledge structure. Ontologies reduce terminological and conceptual ambiguities among people.

Ontologies are widely used in knowledge engineering, artificial intelligence and computer science. Applications of ontologies in these fields are related to knowledge sharing, language processing and other activities. The benefits of using ontologies are the same in any field of human activity, including software engineering. In the following sections we will see the role of ontologies in software development and also the relationship between the ontological domain and the metamodelling domain.

3.1 Ontologies in Software Engineering

The use of ontologies in software engineering and information systems is not new (Gruber, 1993). The software engineering and knowledge engineering communities share a number of common topics (Rech and Althoff, 2004). Ontologies have emerged as a supportive tool in software engineering because they offer clear links between concepts, their relationships and generic theories (Pisanelli et al., 2002). Many authors (e.g. (Happel and Seedorf, 2006)) have discussed the use of ontologies in different phases of the software life cycle.

Decker et al. (2005) explained how the techniques for enhancing *Wiki* content with the help of ontologies (named *Semantic Wikis*) can help in managing requirements in software development. They proposed a new paradigm called *Wikitology.* Lin et al. (1996) explained the use of ontologies in the requirements engineering phase of the software life cycle and showed how ontologies can help requirements engineers to gain domain knowledge. They used first-order logic to define components of the ontology and identify the axioms involved in the objects and their interactions with the aim of answering questions.

Wouters et al. (2000) highlighted how the inherent ambiguities of natural languages in representing requirements can be avoided by the use of ontologies. They presented the idea of a semi-formal approach for use cases, supported by a notion of ontologies, to make them more suitable to communicate requirements in a precise and unambiguous format.

Happel and Seedorf (2006) explained the use of ontologies in component reuse and suggested that ontologies could help to describe the functionality of a component using knowledge representation. They presented ***KOntoR***, an ontology-enabled approach to software reuse. They showed how background knowledge provided in the form of ontologies could increase the value of reuse libraries.

Similarly, in the analysis and design phases, software modelling languages can benefit from the integration of ontologies to reduce ambiguity and enable validation (Tetlow et al., 2006). Knublauch (2005) explained the use of ontologies to map concepts in an ontology to a domain model to provide a single representation of the domain model shared by all participants.

Potential uses of ontologies in other phases of software development, such as coding, testing, deployment and maintenance, were highlighted by (Happel and Seedorf, 2006; Hesse, 2006). Tran and Low (2008) presented ***MOBMAS***, an ontology-based methodology for the analysis and design of multi-agent systems (MAS). They explained the various ways in which ontologies could be used in the MAS development process.

The use of ontologies in software development environments was discussed by (Falbo et al., 2003; Oliveira et al., 2006). Falbo et al. (2003) presented an ontological approach to deal with process and knowledge integration in software engineering environments (SEEs). They also presented an SEE developed using ontologies and named it ODE (ontology-based software development environment). The application of ontologies in software development methodologies was explained by (González-Pérez and Henderson-Sellers, 2006) and they considered the use of ontologies at the methodology, metamodel and endeavour levels.

Categorization of Ontologies Used in Software Engineering

The preceding section mentioned a number of different approaches for using ontologies in software engineering. However, to understand the benefits of ontologies in software engineering, their categorization by level of abstraction and expressivity is required.

Alonso (2006) presented two classifications for the usage of ontologies in software engineering: ontologies of domain and ontologies as artefacts. This classification was extended by (Happel and Seedorf, 2006), who proposed two dimensions of comparison to achieve a more precise classification. Happel and Seedorf (2006) distinguished the roles of ontologies in the context of software engineering between *run-time* usage and *development-time* usage. They also considered the *kind of*

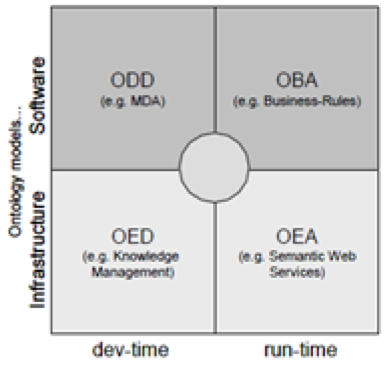

Figure 3.1: Roles of ontologies in software engineering (Happel and Seedorf, 2006)

knowledge the ontology actually covers. The authors distinguished between the problem domain that the software system tries to tackle and infrastructure aspects to make the software more useful or to simplify its development. Figure 3.1 depicts these categories in a matrix form.

ODD (Ontology-driven development) includes the use of ontologies at development time that describe the problem domain itself. This is not necessarily the use of an existing ontology to map to constructs. Rather, this is the integration of ontologies with software modelling languages. The current MDA-based infrastructure provides an architecture for creating models and metamodels, defining transformations between those models, and managing metadata. Though the semantics of a model is defined by its metamodel, the mechanisms to describe the semantics of the domain are rather limited compared. Ontologies can help to overcome this limitation by integrating them to modelling languages. These modelling languages, when used during software development, can result in the development of better (semantically) models. Prime examples are the approaches in the context of MDE. Examples are using UML as an ontology representation language (Cranefield, 2001) by defining direct mappings between language constructs, and using UML as modelling syntax (Baclawski et al., 2002) for ontology development.

OED (Ontology-enabled development) uses ontologies at development time to support developers with their tasks, e.g. component search or problem-solving support. OBA (Ontology-based architectures) uses ontology as a primary artefact at run time. The ontology makes up

a central part of the application logic. Business rule approaches are an example of this kind of application. OEA (Ontology-enabled architectures) leverage ontologies to provide infrastructure support for a software system at run time. An example is semantic web services, where ontologies add a semantic layer on top of the existing web service descriptions, adding functionality for the automatic discovery, matching and composition of service-based workflows.

3.1.1 Ontologies and Metamodels

The relationship of ontologies with models and metamodels in the literature is ambiguous and has been debated extensively. In part, this ambiguity results from overloading of the term *ontology* (Henderson-Sellers, 2011). As we have seen (Section 2.3), a metamodel is also a model, but a model having a target description domain of models; we can say that an ontology, more strictly a *domain ontology* (Bresciani

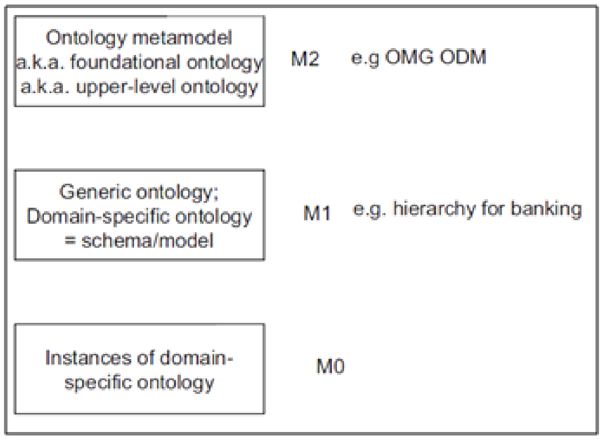
Ontology metamodel
a.k.a. foundational ontology
a.k.a. upper-level ontology
M2
e.g OMG ODM
Generic ontology;
Domain-specific ontology
= schema/model
M1
e.g. hierarchy for banking
Instances of domain-specific ontology
M0

Figure 3.2: Three-level ontology architecture suggested by the Ontology Definition Metamodel of the Object Management Group (after Figure 12 in (Henderson-Sellers, 2011))

et al., 2005), describes entities in a target business domain (brute facts and institutional facts: (Eriksson and Ågerfalk, 2010)) whereas an ontology with ontologies as its target domain can be regarded as a meta-ontology (Bresciani et al., 2005), more commonly called a *foundational ontology* (Figure 3.2). Thus, when interpreting the literature that discusses *ontology* in respect to either models or meta-models, care must be taken as to which of these target domains of the *ontology* is meant. The confusion can also be exacerbated by the use of the same modelling language, often UML class diagrams (Figure 3.3). Discussions relating to domain ontologies include (Spyns et al., 2002), in which ontologies are compared with models. Welty (2006) stated that ontologies are generic as compared with conceptual models, which are specific to the problem in hand. Furthermore, Spyns et al. (2002) emphasized that ontologies are generic, reusable and task neutral while models are task specific.

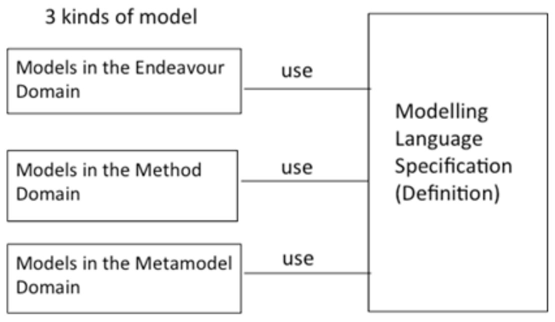
3 kinds of model
Models in the Endeavour Domain
Models in the Method Domain
Models in the Metamodel Domain
use
use
use
Modelling Language Specification (Definition)

Figure 3.3: Three domains defined by the category of users. For models created in any one of these domains, a modelling language (ML) is used. That ML is often the same for the three layers and can be defined in one of a number of ways (after Figure 6.1 in (Henderson-Sellers, 2012))

Similarities between ontologies and metamodels have been investigated by many researchers. For example, Bertrand and Bézivin (2000) suggested that metamodels belong to some domain so are useful for improving domain models while an ontology does so for knowledge models. It was further argued (Devedzić, 2002) that a metamodel can be used to produce different knowledge representations of the same domain but that these representations are incompatible without a common ontology.

Ruiz and Hilera (2006) and Aßmann et al. (2006) pointed out the perplexity between ontologies and metamodels. Ruiz and Hilera (2006) argued that an ontology is descriptive in nature and belongs to the problem domain while metamodels are prescriptive and belong to the solution domain. They also presented and elaborated the idea of ontology-aware metamodels. Devedzić (2002) stated that an ontology is a metamodel that describes how to build models, while Gruber (1993) suggested that ontologies should always be descriptive and never prescriptive.

Comparing ontologies with metamodels also raises the question of where ontologies fit in a hierarchy of abstraction levels such as the one proposed by OMG (OMG, 2001). At the same time the question of whether an ontology can be mapped to more than one metalevel is also important. Dillon et al. (2008) and Falbo et al. (2003) presented their ontologies graphically, mixing the levels of abstraction (e.g. M2, M1). OMG proposed an ontology definition metamodel (OMG, 2011a) in which ontologies are presented at three different levels of abstraction.

Although domain ontologies can be compared with (domain) models and foundational ontologies with metamodels, the ontology matching literature, as investigated here, focuses on domain ontologies whereas our target for harmonization is not domain models but software engineering metamodels. These viewpoints can be reconciled if we remember that the above discussion effectively relates to four similar, yet different in detail, kinds of model. Although both ontologies and metamodels are models, it can be deduced from the literature that, as a consequence of their origins in different problem domains (an ontology is basically a backward-looking or descriptive representation of elements already existing in some domain (physical or conceptual); metamodels arose in software engineering as language definitions to support both forward-looking (prescriptive) and backward-looking (descriptive) models, there are also some inherent differences. For example, the relationship among the elements of an ontology is simple and straightforward and does not contain any information about the type of relationship, while relationships among the elements of

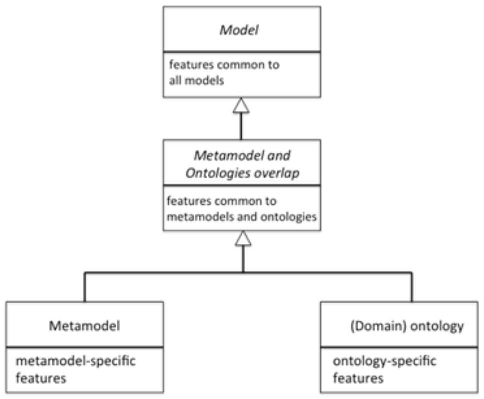
Model
features common to all models
Metamodel and Ontologies overlap
features common to metamodels and ontologies
Metamodel
metamodel-specific features
(Domain) ontology
ontology-specific features

Figure 3.4: Relationship between metamodels and domain ontologies

Table 3.1: Similarities and differences between metamodels and ontologies.

Metamodel	**(Domain) Ontologies**
Models of entities in a target domain	Models of entities in a target domain
Representing concepts and their relationships	Representing concepts and their relationships
A model of models	A model of reality (brute facts plus institutional facts)
Ignores invariants, relies on attributes and relationships	Captures invariants (describes the core essence of the concepts)
Mostly prescriptive	Purely descriptive
Define a modelling language	
Closed world assumption – absence of information means nonexistence	Open world assumption – absence of information does not validate the negative
Implies no uncertainty	Permits uncertainty
Descriptions must be total	Descriptions may be partial
Solution domain	Problem domain
Problem-specific	Generic in nature
Highly specific knowledge	Shared knowledge
Metamodels have more implementation details than ontologies	
Expressible in same language	Expressible in same language
Physical representation is important	Physical representation is not important
Generalization, association and whole–part relationship are equally available	Generalization is main relationship, often in the form of subsumption or *is-a* relationship
Cardinalities on relationships are important	Cardinalities are of less importance
Mostly expressed using semi-formal conceptual modelling languages such as UML	Mostly expressed using knowledge-oriented languages such as OWL, sometimes aided by formal logic

a metamodel can be of different types, e.g. association, aggregation, composition or inheritance. Similarly, the cardinalities of relationships in ontologies are not considered to be very important whereas in metamodels these cardinalities are crucial and important. These similarities and differences are summarized in Table 3.1. Some of

Table 3.2: Definitions of the terms used in table 3.1.

Term	**Definitions**
Target Domain	The conceptual domain that we try to understand. Most of the metamodels and ontologies are targeted towards a certain domain, e.g. web ontologies
Brute Facts vs Institutional Facts	Brute facts are supposed to obtain without doing so in virtue of any other facts obtaining. A brute fact is one whose truth does not depend on some more fundamental fact or facts. Institutional facts by contrast obtain because other facts do.
Invariants	Invariants are the facts that do not change among different domains. Ontologies should deal with general assumptions concerning the explanatory invariants of a domain. Metamodels, in contrast, define the same concept differently in different domains.
Closed World Assumption vs Open World Assumption	The open world assumption is the assumption that the truth value of anything is independent of whether or not it is known by any single observer or agent to be true. The closed world assumption is the opposite of the open world assumption and states that any statement that is not known to be true is considered false.

the terms used in table 3.1 are defined in table 3.2. We can also model these relationships as shown in Figure 3.4 and therefore argue that there is both sufficient similarity to warrant our use of ontology matching for metamodel harmonization, yet sufficient differences that such use is not a triviality, as would be the case if metamodels and

ontologies were synonyms or if metamodels were a simple subtype of ontologies (as suggested by (Pidcock, 2003)).

3.2 Conclusion

In this chapter we have explained the role of ontologies in software engineering. We showed that there are many potential areas where ontologies fit with software engineering and that both communities can learn from each other. We have also discussed the relationship between ontologies and metamodels, considering their similarities and differences. We concluded that ontologies and metamodels have many similarities, yet they are different in many aspects (Table 3.1).

Recent research areas of software engineering like metamodel interoperability can benefit from the application of ontologies. In the next chapter, we will discuss different techniques for ontology merging and schema matching and will use them as a basis for our framework presented in Chapter 5.

Chapter 4

Similarity Matching Techniques

Comparing a pair of conceptual models requires the comparison of not only the syntax but also the semantics and structure of their conceptual elements.

In this chapter, we will explore the literature related to the interoperability techniques for models and metamodels. Terms such as *interoperability*, *bridging*, *merging* and *mapping* have all been used, often loosely, with reference to metamodel compatibility. We begin, in Section 4.1, by defining these terms, gathered together loosely under the generic term *Harmonization.* Section 4.2 demonstrates the interoperability techniques available in the modelling domain. We have seen, in Chapter 3, that metamodels and ontologies have some

similarities that can be used as keys to achieve the interoperability of metamodels, so Section 4.3 details the similarity techniques for ontologies. Moreover, we also believe that metamodels and schemas are closely related to each other. Section 4.4 details the similarity techniques for schemas. Section 4.5 discusses some of the interoperability techniques available in the business process modelling domain and Section 4.6 concludes the chapter.

4.1 Harmonization of Terms

We will use the term *Harmonization* in this thesis as a catch-all term for interoperability, matching and so on. Although Kosanke (2006) has identified over 20 different definitions of interoperability, for our context in this thesis, we consider it reasonable to define the interoperability of any two components as their ability to cooperate with each other (Wegner, 1996) despite their detailed differences. For software systems, interoperability is viewed as the ability of software components to exchange data and services with one another (Heiler, 1995). In the software modelling domain, such harmonization is even more vital since interoperability of models has been identified as "one of the promising features of model driven software development" (ECIS, 2011). Interoperability in MDSE covers the interoperability of modelling tools, modelling processes, modelling languages and metamodels. Our focus in this chapter will be particularly on the interoperability of metamodels.

In the following subsections, we explain the various terminology related to metamodel harmonization and interoperability. We discuss

the terms interoperability, merging, alignment, bridging and mapping. To formalize these concepts, we complement them mathematically, as was done for operators in schemas (Bernstein, 2003) and models (Brunet et al., 2006). Rather than reinventing, or even redefining, these terms, we have sought existing terminology, predominantly it turns out, from the ontology and ontology mapping domains. All the terms can be applied at both metamodel-to-metamodel level and at the scale of an individual class within a metamodel, although bridging (Section 4.1.5) predominantly relates to the latter.

4.1.1 Matching

Matching **is a key element in harmonizing metamodels. Matching, in the ontology domain, is defined as a** ***process of finding correspondences*** **for each entity in the given ontologies (InterOP, 2004). Correspondences are based on similarities among ontological elements. In terms of metamodels, we treat matching as** ***a procedure to find similar conceptual elements*** **(classes) in both metamodels.** ***Matching*** **(Figure 4.1) a pair of elements typically involves both a** ***linguistic analysis*** **and an** ***ontological analysis*** **(details in Chapter 5). The matching of ele-**

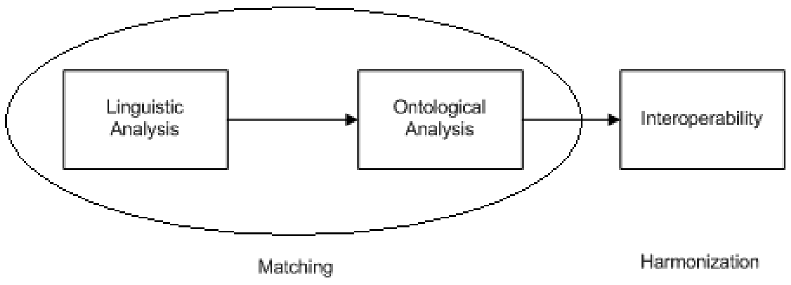

Figure 4.1: Harmonization and matching

ments in a pair of metamodels can be for a class, its behaviour or its relationship with other classes. In our proposed framework (Chapter 5), we will treat these various matchings separately. In particular, we draw attention to the difference between ontology matching, which focuses on structure and static relationships between concepts, and the more dynamic nature of metamodel matching, which must also include behavioural aspects (methods in meta-classes).

In the metamodelling domain, we can say that, if M1 is a set of classes for metamodel MM1 and M2 is a set of classes for metamodel MM2, then matching can be defined as a set, S, of pairs:

$$S = \{(a, b) : a \in M1, b \in M2, a \cong b\} \tag{4.1}$$

Elements a, b in each pair (Equation 4.1) are the elements of M1 and M2 respectively that are similar ($\cong$). The similarity (on a scale of 0% to 100%) of the concepts in a pair can be computed using many different techniques reported in the literature.

Another term, *transformation*, also occurs frequently in the literature (e.g. (Solberg et al., 2005)). Transformation is a process of changing the structure of one metamodel without changing its semantics to make it suitable for some other purpose. Transformation is considered to be a core of MDA (Sendall and Kozaczynski, 2003), where a CIM (computation-independent model) is transformed to a PIM (platform-independent model) and then to a PSM (platform-specific model) (Figure 4.2). However, in metamodels, transformation is often applied to a single metamodel rather than two separate metamodels. For example, in Figure 4.3, a metamodel (in the agent-oriented do-

main) is transformed to another metamodel. Note that in Figure 4.3, the actual semantics of the metamodel are preserved and most of the entities are the same in the transformed metamodel. Transformation of metamodels may require a source meta-metamodel and a target meta-metamodel. However, apart from some metamodels from the OMG (which use MOF), few software engineering metamodels have meta-metamodels. Since transformation does not deal with any aspect of harmonization or interoperability, we do not offer any further discussion of transformation techniques, which are more closely related to the MDA philosophy and architecture.

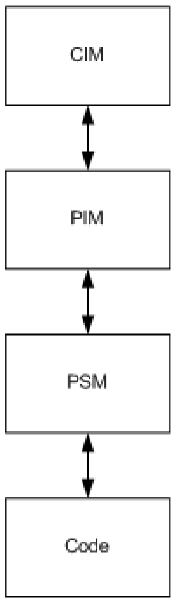

Figure 4.2: The MDA architecture from CIM (computationally independent model) to PIM (platform independent model) to PSM (platform specific model)

4.1.2 Interoperability

Interoperability of metamodels aims to relate two metamodels with the goal of using them jointly. This may require a merger into a single, new metamodel or alignment of the metamodels (Figure 4.4) depending upon the usage scenario (Noy and Musen, 1999). We can

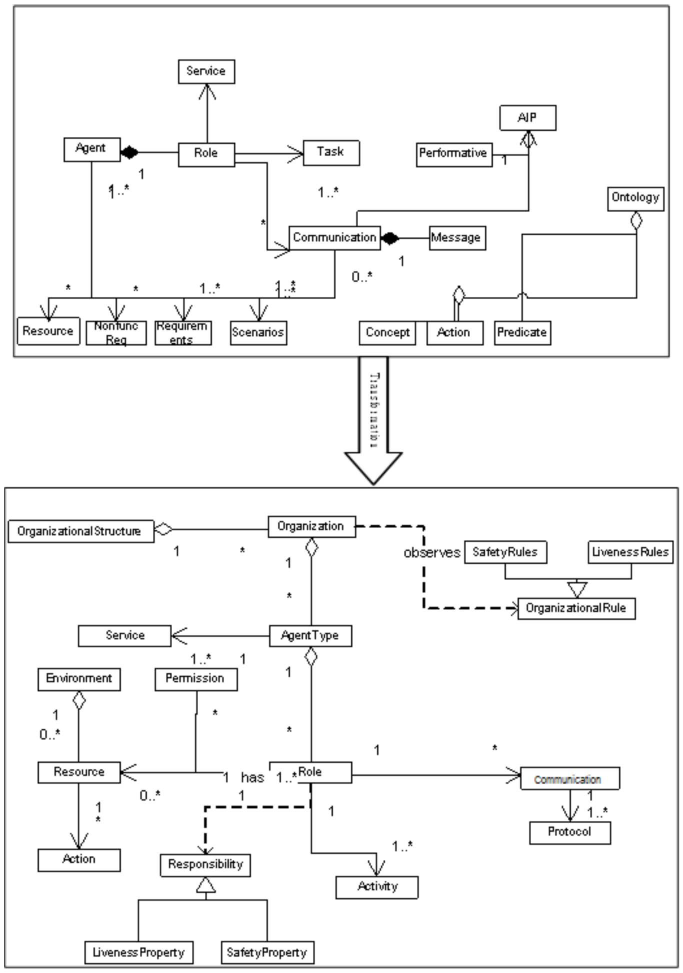

Figure 4.3: Metamodel transformation

therefore say that interoperability of metamodels is achieved by either or both of the operations *merging* and *alignment*, discussed in the proceeding sections.

$$Interop(metamodel, metamodel) = Merging \mid Alignment \tag{4.2}$$

4.1.3 Merging and Integration

The terms *merging* and *integration* are often used interchangeably. *Merging* of metamodels is the creation of a new metamodel from two existing metamodels. *Merging* can be achieved once *matching* (Section 4.1.1) has been successfully determined; for instance, by using the set *S* of correspondences (Equation 4.1). The merge function can be formalized as:

$$merge(metamodel, metamodel, S) = metamodel \tag{4.3}$$

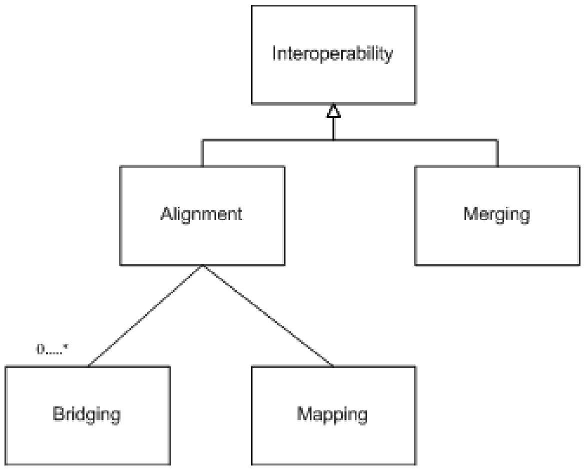

Figure 4.4: Metamodel interoperability

Merging **of two metamodels may require some changes in the conceptual elements of both metamodels. These changes can include the elimination of unrelated concepts or the addition of missing information in either or both metamodels.** ***Merging*** **is thus dependent upon identifying concepts that are syntactically and/or semantically the same and is finalized after both** ***linguistic*** **and** ***ontological*** **analysis (Figure 4.1). For instance,** ***Property*** **is a concept that is the same (syntactically and semantically) in BPMN (OMG, 2011a) and OSM (OMG, 2009b) metamodels.** ***Property*** **is shown, in Figure 4.5, as a single or merged concept in the unified metamodel. On the other hand, the concept** ***Person*** **in OSM (OMG, 2009b) is not the same (syntactically) as** ***HumanPerformer*** **in BPMN (OMG, 2011a), but semantic analysis (Chapter 5) reveals that they can easily be merged as a single concept.**

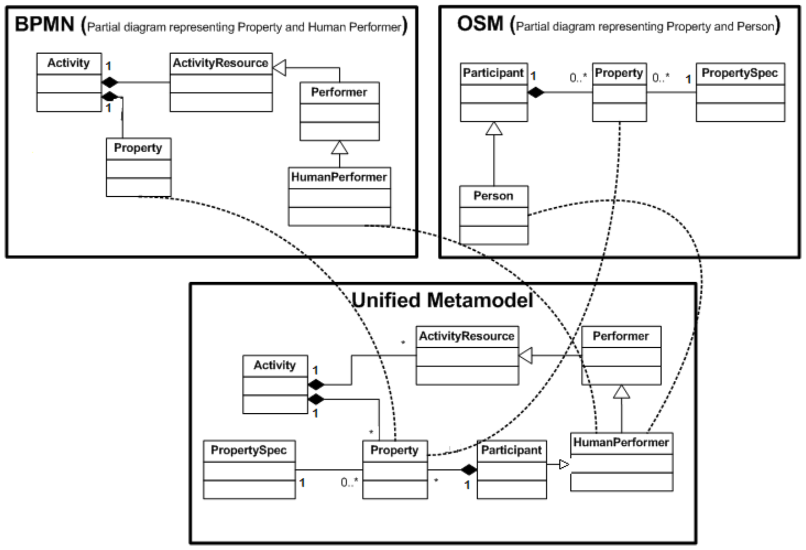

Figure 4.5: Merging of concepts from BPMN and OSM

4.1.4 Alignment

In the domain of ontologies, *alignment* is a process for making two ontologies coherent and consistent with each other (InterOP, 2004). *Alignment*, in ontologies, is achieved by using matchings (Section 4.1.1) between the ontologies (ISO, 2011). In metamodels, by alignment we mean connecting two metamodels in a way that both can be used together when required. We can reach the alignment of a pair of metamodels when we have a set S of corresponding elements (Equation 4.1) between metamodels. *Alignment* of metamodels should not require any changes to the basic structure of either metamodel. It may require the direct mapping (Section 4.1.6) of elements to each other or elements may require an intermediary element to be added to provide a fulcrum to link the two metamodels; this is known as bridging (Section 4.1.5).

4.1.5 Bridging

Bridges between concepts are created when two concepts in two different metamodels are not the same but are linked through a concept that is either the same or similar in both metamodels. Bridging does not require metamodels to have a concept in common. For example in (Figure 4.6), *Activity* in BPMN (OMG, 2011a) and *Participant* in OSM (OMG, 2009b) were not found to be similar (based on the analysis presented in Chapter 5), so they cannot be merged; instead, both of these concepts are associated with the concept *Property* in their respective metamodels. To align these two metamodels, a bridge must be created between the *Activity* of BPMN and *Participant* of OSM using *Property*, preserving the relationships (aggregation and

association) of both metamodels.

4.1.6 Mapping

Mapping **can be undertaken by attaching an element from the original metamodel to an element in the unified metamodel while preserving the relationships of the original metamodel. For example, in (Figure 4.7)** ***Constraints***, ***Spec*** **and** ***Property Spec*** **are associated with** ***Property*** **in OSM (OMG, 2009b). Since there is no similar concept (for details, see Chapter 6) in BPMN, we map it to** ***Property*** **in the unified metamodel.**

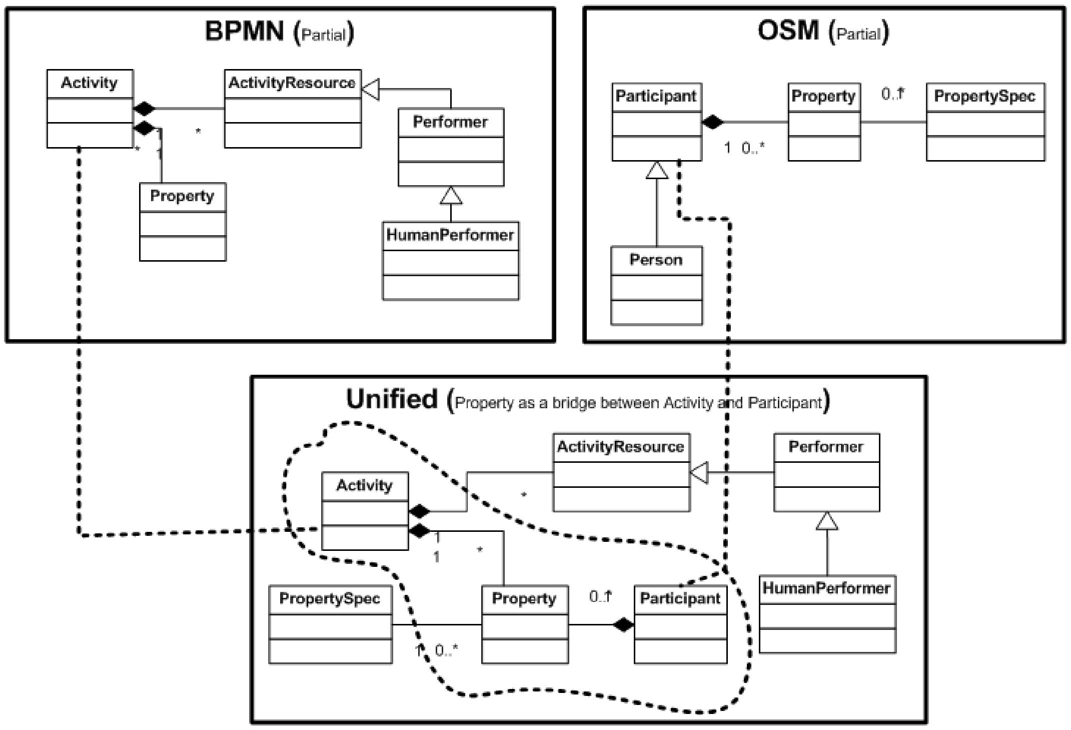

Figure 4.6: Bridging of concepts in BPMN and OSM

Overall, we can see that the most important part of interoperability for either merging or alignment is to find the set S of matchings between metamodels (Figure 4.1). Matches can be found based on the similarities among the metamodels' concepts. Different types of similarities are used in the literature among conceptual entities. The next few sections survey matching techniques used with models and metamodels. We will also explore matching techniques in areas related to metamodels, such as schemas, ontologies and graphs.

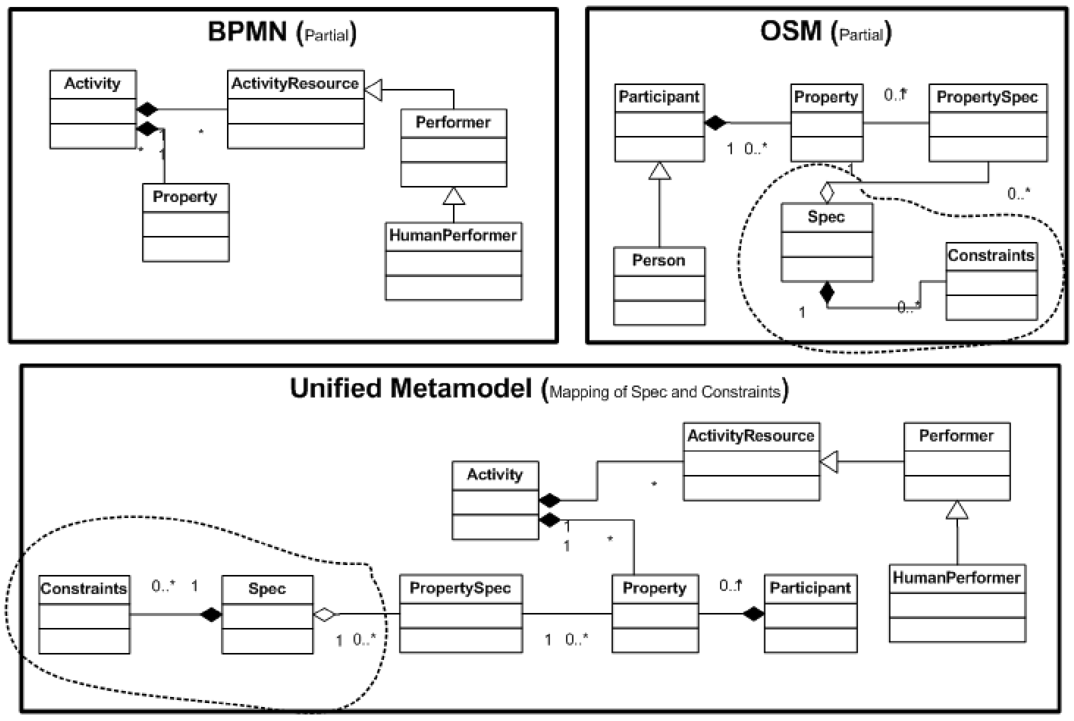

Figure 4.7: Mapping of concepts in BPMN and OSM

4.2 Similarity Techniques in Models and Metamodels

Numerous techniques have been reported in the literature for finding the similarities between models and metamodels. These techniques can be broadly grouped as tool based, transformation based, language based and general. The following paragraphs will discuss those techniques.

A prototype and the architecture of a tool called *Model Bus* was presented in (Blanc et al., 2005), where the authors highlighted the issue of interoperability among modelling services and demonstrated how *Model Bus* could be used to connect the services of different modelling tools. An approach using ontology matching tools to find correspondences between metamodels was demonstrated by (Kappel et al., 2006) and they reported their experience of matching UML, ECore, WebML and EER. An algorithm for metamodel matching was elaborated in (de Sousa Jr et al., 2009) where semiautomatic mappings were found and matching suggestions were made. These matching suggestions were then evaluated by the user. This work is an enhancement and adaptation of the work by (Chukmol et al., 2005a), used in MDA. A tool called SAT4MDE (SemiAutomatic Tool for Model Driven Engineering) was implemented by de Sousa Jr et al. (2009) as an extension of their own work (Lopes et al., 2006b). The authors illustrated the application of this tool to UML and C# metamodels.

Transforming models and metamodels into other conceptual entities and then finding their similarities are also covered in the literature. For example, Kapsammer et al. (2006) presented a semiautomatic approach to transforming metamodels to ontologies and then using ontology matching techniques to find mappings between the metamodels. They also explained how refactoring patterns can be applied to the resulting ontologies. Tolosa et al. (2009) presented an approach for metamodel interoperability based on model transformations. Their approach was aimed at achieving higher levels of interoperability by raising the abstraction level of the transformation models. They also proposed a transformation model taxonomy. The idea of changing UML models to Petri Nets (PNs) and then mapping components of a PN to other PNs was presented by (Saldhana and Shatz, 2000). Their approach was based on the conversion of UML state charts to PNs, which is straightforward and easy. The limitation of this approach is that it has not been used to convert class diagrams to PNs. As most metamodels are represented using class diagrams, the viability of this approach for metamodels is limited. Transforming metamodels to graphs and then applying graph matching techniques was discussed in (Voigt and Heinze, 2010a). The authors adopted an algorithm from (Neuhaus and Bunke, 2004) and their approach was based on planar graph theory. Planarity is a barrier to adopting this approach as most metamodels are nonplanar, e.g. UML, WSDL, OCL and OWL. Converting them into planar graphs is not straightforward.

Bettin and Clark (2010) proposed *Gmodel*, a metalanguage that can be used to specify and instantiate a modelling language. They illustrated the use of *Gmodel* in model-driven integration with an example

of the Eclipse Modelling Framework (ECore). Kolovos et al. (2006) discussed the special requirements for model merging and introduced the Epsilon Merging Language (EML). EML is a rule-based language with tool support for merging models of diverse metamodels.

Henderson-Sellers and González-Pérez (2004) demonstrated a generic metamodel SMSDM (standard metamodel for software development methodologies) by comparing and mapping the elements of four well-known metamodels: SPEM (software and systems process engineering metamodel), OPF (OPEN process framework), OOSPICE (Software Process Improvement and Capability Determination for Object Oriented or component-based software development) and CSCW (computer-supported collaborative work). The authors mapped the elements of the metamodels using heuristics but did not discuss any technique for matching conceptual elements of these metamodels. Kühn and Murzek (2005) presented an overview of interoperability issues according to conceptual domains in metamodelling platform architectures. They divided the interoperability problem into different levels such as *meta2 model*, *metamodel* and *model of each platform.* Ouksel and Sheth (1999) categorized interoperability as "information heterogenity and system heterogeneity." Uhrig (2008) presented an approach to finding the minimal-cost match of classes in a class diagram, but did not provide details of the matching process. Pottinger and Bernstein (2003) defined the *Merge* operator for model merging. They also defined and classified the conflicts that can arise while combining models. They categorized these conflicts as *representation*, *metamodel* and *fundamental* conflicts.

4.3 Similarity Techniques in Ontologies

Different approaches to the interoperability of ontologies are generally called ontology mediation (Bruijn et al., 2006). Ontology mediation can be classified as *ontology mapping* (Maedche et al., 2002; Scharffe and Bruijn, 2005), *ontology aligning* (Ehrig and Staab, 2004; Ehrig and Sure, 2004) and *ontology merging* (Noy and Musen, 1999) techniques. The following paragraphs will discuss some of these techniques reported in the literature.

A survey of ontology merging techniques was presented by Bruijn et al. (2006), in which the authors also described an approach to representing ontology mappings using a mapping language and discovering mappings using an alignment process. They claimed that this algorithm could be used with the ontology management tool OntoMapand offered a plugin for OntoMap. Hitzler et al. (2005) argued that the problem of merging ontologies could be solved by constructing "pushouts" from category theory. They viewed category theory as a universal metalanguage that enables us to specify properties and relationships of ontologies in an implementation-independent way.

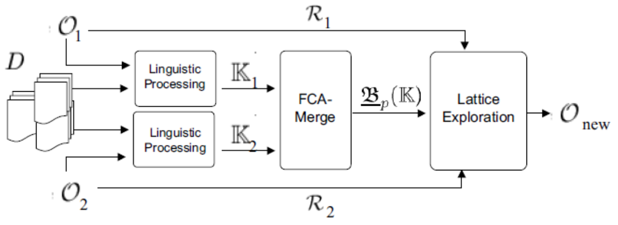

Figure 4.8: Ontology merging method (Stumme and Maedche, 2001)

Siricharoen (2008) presented an approach for semiautomatic construction of an object model from ontologies and demonstrated the integration of different XML-based formats to merge and modify ontologies into an object model. Formal concept analysis (FCA) has also been used (Stumme and Maedche, 2001) to merge two ontologies in a bottom-up fashion. This approach was based on three steps: developing formal contexts based on linguistic analysis, merging of two contexts and building up a concept lattice and finally semiautomatic ontology creation (Figure 4.8).

Mohsenzadah et al. (2005) presented an algorithm for the merger of two ontologies. They categorized ontologies as global and local and studied the merging of local ontologies without the help of a global ontology. Their algorithm has two major phases: *syntactic checking* and *semantic checking*. Syntactic similarities were measured using an edit-distance formula. Semantic checking was undertaken using the lexical database of WordNet. Noy and Musen (2003) presented a tool *PROMPT* to align and merge ontologies based on an algorithm. Noy and Musen (1999) presented a semi-automated SMART (not an acronym) algorithm for ontology merging in a five-step approach. Some of these steps are performed automatically (e.g. initial list, automatic updates) and some are performed manually by the user (e.g. load, select operation). They also defined some operators that could be used during the merging and aligning process. Magnini and Speranza (2002) explained the process of interoperability between linguistic ontologies. They presented a methodology for the integration of global ontologies with specialized linguistic ontologies. They used several *Plug Relations* (Plug-synonymy, Plug-near-synonymy and Plug-

hyponymy) to connect many concepts in two ontologies.

HeÃŸ (2006) presented a method for mapping ontologies based on string-distance metrics. Their approach was based on an iterative algorithm that computes *intrinsic* and *extrinsic* similarities. They used algorithms from graph theory to compute one-to-one mappings. Kim et al. (2006) suggested an ontology-mapping algorithm for heterogeneous classifications. They also used *sensitivity* and *precision* to compare their algorithm with PROMPT.

Resnik (1995) proposed a new measure for semantic similarity in an IS-A taxonomy, based on the notion of information content. Jean et al. (2011) presented OntoQL, a language that combines the capabilities of metamodelling languages with ontology query languages. The OntoQL language is based on a core metamodel containing the common and shared constructs of most of the usual ontology models. This core metamodel is used to define ontologies.

4.4 Similarity Techniques in Schemas

Rahm and Bernstein (2001) proposed a taxonomy that covers many of the existing approaches (Figure 4.9) for schema matching and described these approaches in some detail. They distinguished between schema and instance level, element and structure level and language- and constraint-based matchers and discussed the combination of multiple matchers. They used the taxonomy to characterize and compare a variety of previous match implementations.

Lopes et al. (2006a) presented an approach to schema matching in the context of MDE. They developed an algorithm for schema matching and implemented it as a plugin for Eclipse. They illustrated their algorithm using UML and Java metamodels.

Chukmol et al. (2005a) described a schema-matching algorithm EXSMAL that, by using an XML schema as the pivot format, automates the semantic correspondence discovery between EDI (Electronic Data Interchange) models. Melnik et al. (2002) presented *similarity flooding*, a matching algorithm based on fixed-point computation that is usable across different scenarios. The algorithm takes two graphs (schemas, catalogs or other data structures) as input and outputs a mapping between corresponding nodes of the graphs.

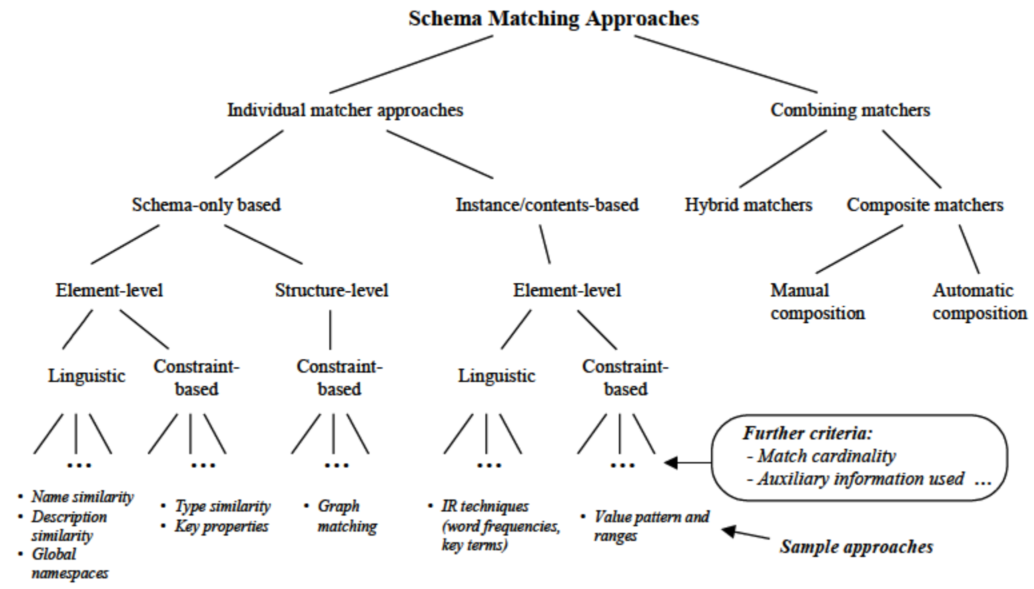

Figure 4.9: Classification of schema matching techniques (Rahm and Bernstein, 2001)

4.5 Similarity Techniques in Business Process Models

Gehlert and Pfeiffer (2005) proposed a framework to compare conceptual models. Their framework was based on four stages, called *domain term extractor*, *domain model builder*, *type conflict resolver* and *isomorphism analyzer*. They identified three different theoretical levels of conceptual modelling, conceptual models, modelling language and application domain language. They based their work on the argument of Werner et al. (2004) about two different types of semantics of conceptual models, abstract semantics and concrete semantics. Werner et al. (2004) also argued that complete automation of comparing conceptual models is not possible and we need human intervention at the semantic level. They did not implement or test their framework empirically.

Ehrig et al. (2007) proposed an approach to detect similarities between business process models. They used the idea of translating the business process models to PNs and then to OWL (Web Ontology Language) to measure the similarity. Dumas et al. (2009) also compared business process models. Their inspiration was to identify process models in a repository that most closely resemble a given process model. They used three different types of similarities, *label similarity*, *structural similarity* and *behaviour similarity*. They reviewed different techniques in the literature to find these similarities and also compared the precision of these techniques by applying them to different process models.

An approach to measuring the degree of similarity between business process models was presented by van Dongen et al. (2008). The authors used the EPC (event process chain) to represent business process models and used casual footprints to calculate their similarity based on the vector-space model from information retrieval. They validated their approach using an SAP reference model and the results highlighted the need for automated similarity detection. Dijkman et al. (2011) used the approach of van Dongen et al. (2008) and presented three parameterized similarity metrics between business process models. They calculated the precision and recall of these techniques.

Zhong et al. (2002) proposed an algorithmic approach to semantic search by matching conceptual graphs. They provided some definitions of semantic similarities between concepts, relations and conceptual graphs. The similarity between two concepts c_1 and c_2 was measured by the distance ($d_c(c_1, c_2)$) between them, which was calculated using milestones (Equation 4.4) and the similarity was defined as $sim_c(c_1, c_2)$ = 1 - $d_c(c_1, c_2)$.

$$milestone(n) = \frac{1/2}{e^{l(n)}} \tag{4.4}$$

Similarity between relations is computed using a similar approach (sim_r (r_1, r_2) = 1 - $d_r(r_1, r_2)$) and the distance is calculated from the relations' respective positions in the relation hierarchy. To simplify the process, the authors suggested assigning standard values to the similarity of relationships (Equation 4.5). The similarity between conceptual graphs is then calculated using these two similarities, which

can be seen in detail in (Zhong et al., 2002).

$$sim_r(r_Q, r_R) = 1 - d_r(r_Q, r_R) = \begin{cases} 1, & r_Q \quad subsumes \quad r_R \\ 0, & others \end{cases} \tag{4.5}$$

Generally, business process models are dynamic in nature as compared with metamodels that represent static structures. Applying the similarity techniques of business process models to metamodels is thus not straightforward.

4.6 Conclusion

In this chapter, we have discussed the harmonization of different terms used in the context of metamodel interoperability (Section 4.1). We have seen that metamodel interoperability can be achieved using either *alignment* or *merging* or a combination of them (Figure 4.4). The *alignment* of metamodels may require *bridging* and/or *mapping* of metamodels' conceptual elements. We showed that finding similar elements between a pair of metamodels through *matching* (Section 4.1.1) is of paramount importance to metamodel interoperability.

We next investigated the literature of similarity techniques for models and metamodels (Section 4.2), ontologies (Section 4.3), schemas (Section 4.4) and business process models (Section 4.5). Based on the above survey, we found that different types of similarities (e.g. syntactic and semantic) should be sought between pairs of metamodels. In the next chapter, we present the interoperability framework that we have developed for metamodels. This framework is based on some of the techniques presented in this chapter, yet is novel in many ways.

We have not only extended some of these techniques (necessary for metamodels) but have also used some new techniques for metamodel interoperability. We will also discuss different types of similarities in detail as part of the framework.

Chapter 5

The Interoperability Framework

It is the framework which changes with each new technology and not just the picture within the frame.

Marshall McLuhan

.

The central concern of this thesis is to devise a way in which metamodels can be used in an interoperable fashion. The interoperability of metamodels, as we have discussed in Chapter 4, is based on the identification of potential common concepts (conceptual elements) that can be shared between a pair of metamodels (set S in Equation 4.1). This chapter presents a framework that can be used for metamodel interoperability. It was developed by applying ontology merging and schema matching techniques. The framework is presented as a BPMN (OMG, 2011a) diagram in (Figure 5.1).

As metamodels have both linguistic and ontological (see Section 2.3) perspectives (Atkinson and Kühne, 2003), this framework performs both linguistic and ontological analysis on metamodels to identify shared/common concepts. The framework has three major stages, *linguistic analysis* (Section 5.2), *ontological analysis* (Section 5.3) and *interoperability* (Section 5.4). These activities are divided into further subactivities and are discussed in the following sections of this chapter with the help of some examples.

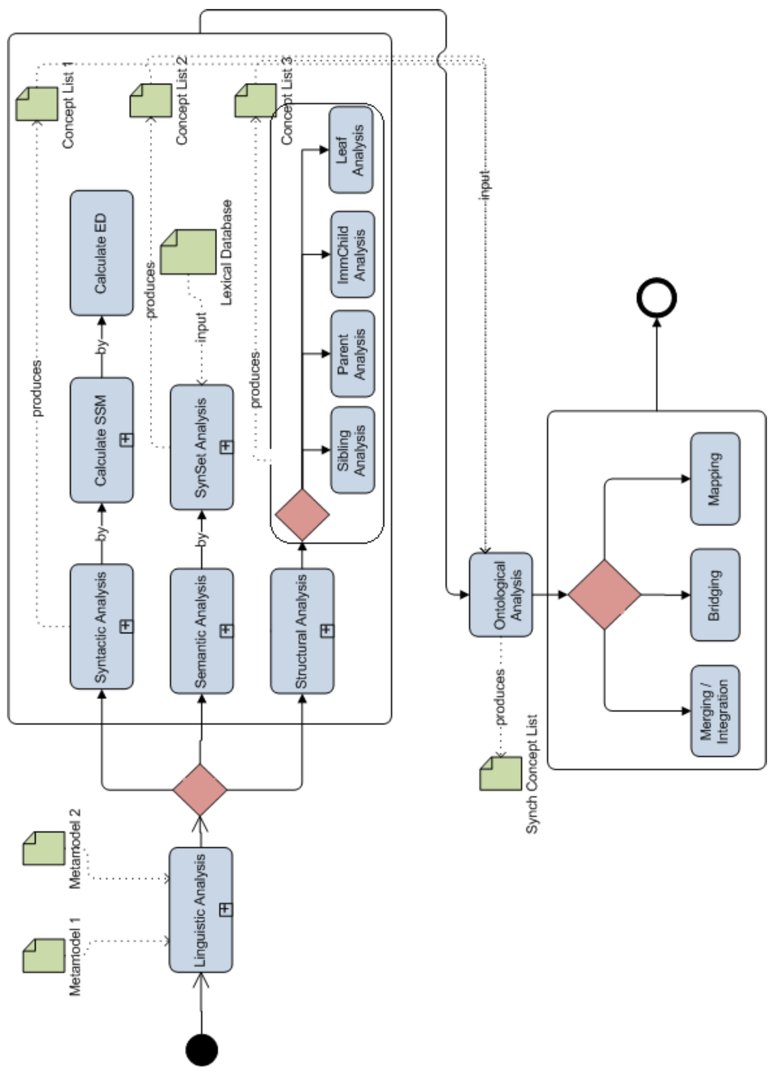

Figure 5.1: Metamodel interoperability framework

5.1 Selection of Metamodels

The framework presented here (Figure 5.1) is generic and can be used for any pair of metamodels. For better results and applicability of this framework, we assume some commonality between the pair of metamodels among their conceptual elements. The framework can be applied to metamodels a pair at a time. If the intention is to achieve interoperability of more than two metamodels, then this framework should still be applied to pairs of metamodels. For example, if the framework is to be applied to metamodels A, B and C, then it should be applied to pairs (A, B), (A, C) and (B, C).

Another important factor to consider in selection of metamodels is that metamodels have increased in size and complexity over time (Henderson-Sellers et al., 2012). Most metamodels contain a large number of conceptual elements, sometimes in the hundreds, e.g. BPMN (OMG, 2011a) has more than 100 conceptual classes. Performing activities on such frameworks manually can be a time-consuming task. It is preferable to select metamodels that have some sort of formal specifications, so that the list of conceptual elements can be extracted in an automated fashion or with minimum analyst effort.

This list should contain all the conceptual elements and their parent elements in that metamodel. These parent elements are required for structural analysis (Section 5.2.3). Once the lists of both metamodels have been prepared, the next step is to perform the first comparison, known as linguistic analysis.

5.2 Linguistic Analysis

Linguistic analysis is a three-stage activity that checks the linguistic similarity between the concepts of two metamodels. Linguistic analysis comprises syntactic, semantic and structural analysis of conceptual elements of metamodels. The following sections elaborate these in detail.

$$SSM(S_i, S_j) = max\left(0, \frac{min(|S_i|, |S_j|) - ED(S_i, S_j)}{min(|S_i|, |S_j|)}\right) \tag{5.1}$$

5.2.1 Syntactic Analysis

Syntactic similarity analysis is the comparison of metamodels' elements based on their names. We will analyse the syntactic similarity between pairs of conceptual elements, using the so-called Syntactic Similarity Measure (SSM) proposed by (Maedche and Staab, 2001) (Equation 5.1). SSM is calculated using the edit-distance formula (ED) proposed by Levenshtein (1966). ED views each string as a list of tokens and counts the number of token insertions, deletions and substitutions needed to transform lexical string S$_i$ to S$_j$, viewing each string as a list of tokens. For example, the ED between the two strings *Brian* and *Brain* is two, as we must replace only two characters of the first string (Brian) to make it the same as the second (Brain).

The value of SSM will always be between zero and one, where one represents exactly *the same*, zero represents a *bad match* and all other values mean that elements have some *similarity.* Small values of ED indicate greater similarity between a pair of concepts. For example,

the ED value between Brain and Brian was 2, so their SSM value will be 0.6, which means they have 60% syntactic similarity. After calculating SSM for each concept pair in the metamodels (Concept List 1 in Figure 5.1), the concepts that are the same or have a strong similarity are listed for further evaluation.

5.2.2 Semantic Analysis

Identifying only the syntactic similarities of metamodels elements, isolated from their semantics, is not sufficient. For example, conceptual elements like *Confirmation* and *Notification* have approximately 60% syntactic similarity and *Confirmation* and *Verification* have 50% syntactic similarity. Considering their semantics, it is easy to determine that *Confirmation* has a greater degree of similarity with *Verification* than with *Notification.*

The purpose of semantic analysis is to identify the concepts (conceptual elements) in both metamodels which were either not identified as similar or had low syntactic similarity because of their names. It is possible that two different concepts having the same semantics may be named differently in two metamodels. For example, the concepts *Activity* and *Task* seem to be semantically similar but cannot be identified as similar using SSM. Semantic analysis is done by identifying different synonyms and word senses for each concept in both metamodels. These synonyms can be found using any lexical database like WordNet (Miller, 1995). Sets of synonyms, called *synSets*, should be formed for each concept of both metamodels. Each concept in one metamodel and all of its synonyms should then be compared with each concept and all of its synonyms in the other metamodel using

(Equation 5.2).

$$semSim(C_i, C_j) = \frac{\sum_{i=1}^{|synSet(C_1|)} \sum_{j=1}^{|synSet(C_2)|} M[i][j]}{|synSet(C_1)| * |synSet(C_2)|}, \tag{5.2}$$

where M is the matrix containing the SSM values of all synonyms for both concepts $\mathbf{C}_1$ and $\mathbf{C}_2$. For example, to compare the two conceptual elements *Activity* and *Process*, we first build the synSets containing their synonyms separately. Below are example *synSets* for both elements containing some of the synonyms we found using WordNet (Miller, 1995).

$$synSet(Activity) = \{Action, Process, Doing, Motion\}$$
$$synSet(Process) = \{Procedure, Course, Method, Action\}$$

Table 5.1 shows the ED values for both concepts (Activity, Process) and their synonyms. The concept *Activity* and all of its synonyms, synSet(Activity), are placed horizontally, while the concept *Process* and all of its synonyms, synSet(Process), are placed vertically. The concepts having similar names resulted in low Ed values, e.g. the value 0 (zero) for the synonym *Action* in the last row and the second column.

Table 5.1: Edit distance values between *Process* and *Activity*

	Activity	**Action**	**Process**	**Motion**
Process	8	7	0	6
Procedure	9	8	4	8
Course	7	6	5	5
Method	7	4	7	3
Action	4	0	7	2

Table 5.2: SSM values between *Process* and *Activity*

	Activity	**Action**	**Process**	**Motion**
Process	0	0	1	0
Procedure	0	0	0.57	0
Course	0	0	0.16	0.16
Method	0	0.33	0	0.5
Action	0.33	1	0	0.66

Table 5.2 shows the SSM values (calculated using Equation 5.1) for the concepts (Activity, Process). This table can be considered as the matrix M in Equation 5.2. It is to be noted that the synonyms that have low ED values in Table 5.1 lead to high SSM values, e.g. the concept *Action* resulted in SSM = 1 (one), which means that both concepts are the same. Table 5.2 is then used to calculate the semantic similarity of both concepts using (Equation 5.2), which resulted as semSim(Activity,Process) = 0.235. This means that the concepts (*Activity* and *Process*) have 23% semantic similarity.

The synonyms of metamodels' concepts can be stored in a database and queries can be initiated to check the similarity of concepts and their synonyms. The result of this activity will be another concept list (Concept List 2 in Figure 5.1). This list may contain some duplicate entries because of the matching of multiple synonyms of the same concept. These duplicates should be removed from the list before further structural analysis.

5.2.3 Structural Analysis

To find similar elements, we believe that the *structure* of metamodels' elements should also be compared along with their syntax and semantics. Structural similarity can be measured using different ap-

proaches proposed in the literature. Structural analysis in our framework is based on the techniques used by (Chukmol et al., 2005a) and de Sousa Jr et al. (2009). The structural similarity of two concepts C_1 and C_2 is calculated based on the similarity of their structural neighbours. Structural neighbours of a concept C_1 are *ancestors(C_1)*, *siblings(C_1)*, *immediateChild(C_1)* and *leaf(C_1)* (Figure 5.2). The structural similarity is obtained as a function of partial similarities

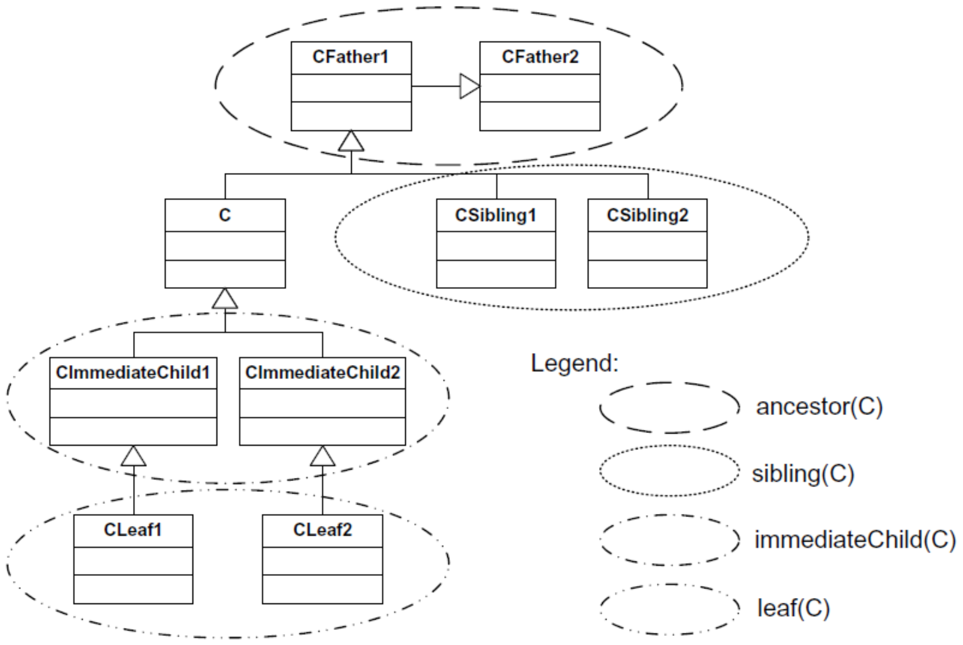

Figure 5.2: Structural neighbours of a class C (de Sousa Jr et al., 2009)

(Equation 5.3).

$$\begin{aligned} structSim(C_i, C_j) = ancSim(C_i, C_j) * ancCoef + \\ sibSim(C_i, C_j) * sibCoef + \\ immChSim(C_i, C_j) * immChCoef + \\ leafSim(C_i, C_j) * leafCoef \end{aligned} \tag{5.3}$$

Each of the coefficients (*$ancCoef$*, *$sibCoef$*, *$immChildCoef$* and *$leafCoef$*) must have a value between zero and one. Further, $ancCoef + sibCoef + immChildCoef + leafCoef = 1$. These partial similarities are then calculated using the mean values of all the neighbouring concepts. For example, the similarity between the ancestors of two concepts C_1 and C_2 can be calculated using Equation 5.4.

$$ancSim(C_i, C_j) = \frac{\sum_{i=1}^{|ancestors(C_1)|} \sum_{j=1}^{|ancestors(C_2)|} M[i][j]}{|ancestors(C_1)| * |ancestors(C_2)|} \tag{5.4}$$

M is the matrix containing SSM values for all concepts in the ancestor sets of both C_1 and C_2.

The next step is to synthesize concept lists 1, 2 and 3 and filter out the candidate concepts that can be unified (Synch Concept List in Figure 5). The overall similarity is then calculated for all pairs of concepts using Equation 5.5. The synthesized list of concepts that have values of overall similarity greater than a threshold value (set by the analyst) represents the correspondences of mapping points in both metamodels. The value of the threshold can vary for each pair of metamodels. Some metamodels have more similar conceptual elements (e.g. metamodels belonging to the same domain like BPMN

and OSM), so the threshold value can be higher as compare to the metamodels belonging to two different domains. For example, we can see, in the chapter 6 (Table 6.19), that how two metamodels (ECore and UML2CD)belonging to the same domain represents the same ontological concepts, so we can expect higher structural similarity, which is 40%. In some cases, this threshold value may need to be set lower where we don't expect more similarity between conceptual elements of a pair of metamodels (e.g structural similarity in Table 6.21). The value of the threshold does affect the overall similarity between metamodels, but this value can be adjusted (setting it lower) to get more pairs of conceptual elements to be evaluated for ontological analysis

$$\begin{aligned} similarity(C_1, C_2) = SSM(C_1, C_2) * synCoef + \\ semSim(C_1, C_2) * semCoef + \\ structSim(C_1, C_2) * structCoef \end{aligned} \tag{5.5}$$

Values of coefficients (e.g. $synCoef$**) in Equation 5.5 have the same conditions as the coefficients in Equation 5.3 and are selected by the analyst, depending upon the importance (for the analyst) of some similarity measure. For example, assigning** $synCoef = 0.4$ **means that the analyst has weighted the syntactic similarity to contribute 40% of the overall similarity. This synthesised list is used as an input to the next activity, ontological analysis, which requires human intervention.**

5.3 Ontological Analysis

The purpose of the ontological analysis is to critically evaluate the *Synch Concept List* (Figure 5.1) produced by the linguistic analysis (Section 5.2). This evaluation is based on the definitions of those concepts as specified in their metamodels. We call this evaluation checking *ontological semantics* which we believe is different from *linguistic semantics*. Different types of ontological semantics must be evaluated between pairs of conceptual elements. The following sections briefly explain these checks, which are helpful in evaluating the *Ontological Semantics* of metamodels.

5.3.1 Linguistically Same Concepts

This is the evaluation of concepts identified as *same* in the linguistic analysis. These concepts need to be checked based on their definitions in the metamodels. A concept, such as *Communication* may exist in two different metamodels with the same name. This concept will be identified as *same* (syntactically and semantically) in the linguistic analysis stage (Section 5.2), but may have totally different definitions and semantics in terms of the specifications of the metamodels. For example, a concept *Category* may be found linguistically *same* in BPMN and SPEM but a *Category* represents the categories of *Flow Elements* in BPMN (OMG, 2011a) and *Category* in SPEM (OMG, 2008a) represents the categories of *Roles* and *Work Products*. We need to be careful while matching such concepts, as it is sometimes not a straightforward merger.

5.3.2 Linguistically Similar Concepts

This evaluation checks those concepts which were identified as *similar* in the linguistic analysis stage. For example, the concepts *OrgRelationship* in OSM (OMG, 2009b) and *Relationship* in BPMN (OMG, 2011a) may have been identified as similar because of their names. OSM defines an organization model as *a set of relationships between people* (OMG, 2009b). These relationships are $n$ary and define associations of two or more *OrgRoles.* On the contrary, *Relationship* in BPMN enables BPMN and non-BPMN *Artefacts* to be related in a nonintrusive manner (OMG, 2011a). We can see that the concepts *orgRelationship* and *Relationship* have different semantics in OSM and BPMN. Therefore, we must be careful when deciding whether these concepts relate to each other as representing the same meaning or not, according to their actual definitions in the specifications of these metamodels.

5.3.3 Linguistically Different Concepts

This is the most crucial part of ontological semantics. The objective is to investigate those concepts that seem similar but were not identified as similar in the linguistic analysis stage. This is possible because of the way these concepts are named in metamodels, e.g. concepts like *OrgRole* and *Specs* in OSM (OMG, 2009b). Investigating their actual definitions in the specification of OSM (OMG, 2009b) reveals that these concepts actually represent *Organization Role* and *Specifications*, which could be more likely to be matched with similar concepts in other metamodels, e.g. with *Interaction Specification* in BPMN (OMG, 2011a). We have to analyse these concepts in meta-

models' specifications.

5.3.4 Holonyms and Meronyms

An important consideration during the ontological analysis is to check the relationships between concepts. Some concepts may appear as *holonyms* (whole) of other concepts. A holonym is a concept that is a *includes* of some other concept. That other concept is a *part* of the *holonym* and is known as a *meronym*. For example, *Body* is a holonym of *Leg* and *Apple* can be a meronym of *Apple Tree*. Similarly, a concept like *Activity* in BPMN (OMG, 2011a) is a holonym of another concept *Activity Resource* (Figure 5.3). *Activity Resource* is a *meronym* of *Activity*. This sort of relationship generally appears as an *aggregation/composition* relationship in object-oriented meta-models. Hence, concepts identified as the *same* or *similar* during the linguistic analysis stage (Section 5.2) may be required to accommodate the *holonyms* and *meronyms* of those concepts as well.

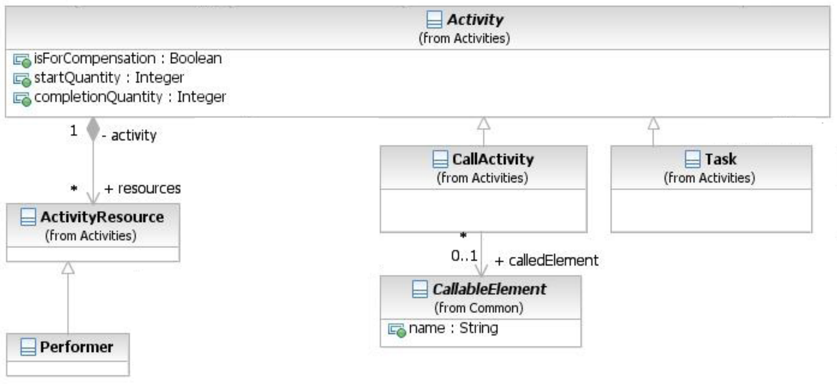

Figure 5.3: Activity and ActivityResource as holonym/meronym in BPMN (OMG, 2011a)

5.3.5 Hypernyms and Hyponyms

A *Hypernym* serves as a generalisation of some specific concept. That specific concept is called a *hyponym*. For example, *Dog* is a hyponym of *Animal* while *Animal* is hypernym of *Dog*. Similarly, *Event* in BPMN (OMG, 2011a) is a hypernym for *Throw Event* and *Catch Event* (Figure 5.4). *Throw Event* and *Catch Event* are hyponyms of the concept *Event*. This type of relation appears as a generalisation (Inheritance) in object-oriented metamodels. Care needs to be taken to accommodate generalised concepts during the merging of specialised concepts.

It is vital to emphasize here that the objective of this thesis is to formulate a way in which metamodels can be used in an interoperable fashion. The framework presented in this chapter is a semiautomated approach. Fully automated merging/mapping of metamodels seems to be an ideal solution but is not the objective of this thesis, nor is it feasible. If concepts of metamodels are to be matched only syntactically (e.g. based on their names), automation may be successful.

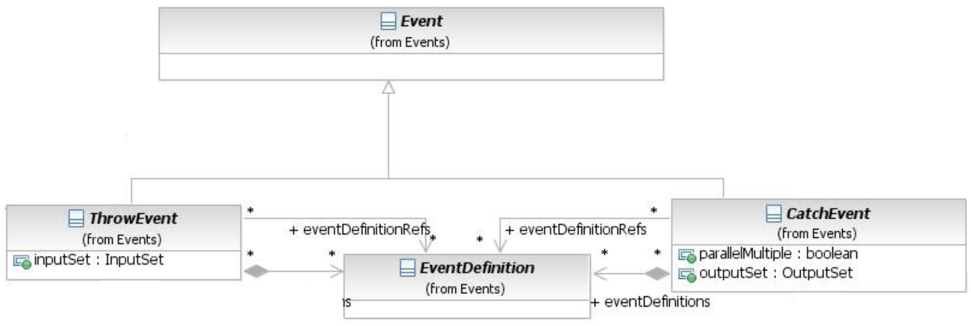

Figure 5.4: *Event* as hypernym in BPMN (OMG, 2011a)

However, to match the semantics of metamodel elements, rational input by the analyst is still necessary (Shvaiko and Euzenat, 2013).

The checks required (Sections 5.3.1 to 5.3.5) during the ontological analysis phase cannot be automated for many reasons, not least because most metamodels lack a formal definition. Metamodels are mostly represented using a collection of diagrams, tables and text. Human intervention can be minimized by using available automated tools for syntactic and semantic matching (e.g. edit distance calculators, APIs for WordNet and Thesaurus to find synonyms), but cannot be eliminated. During the application of this framework to some exemplary metamodels (Chapter 6) we automated the process of syntactic matching by writing code in VB.Net, also by storing and querying the synonyms for semantic matching. This partial automation saves time as compared with manual examination. Another possibility is to have formal specifications of metamodels (e.g. XML/XMI specifications for the OMG family of metamodels) and then to automate the mapping/merging algorithm to use those specifications as input. The analyst intervention during the ontological analysis may be considered as a limitation and could be a barrier to a fully automated solution. At the same time, it is necessary and important to involve human intellect to perform such analyses. At the end of this activity we will have three different types of concepts from both metamodels: exactly the same (both syntactically and semantically), similar and dissimilar (bad matches). This gives an adequate and useful analysis of the concepts in the metamodels and we can now start to make them interoperable.

5.4 Interoperability

As explained earlier (Chapter 4), the interoperability of metamodels can be achieved either by *merging and integration* or by *alignment*. *Alignment* can include either or both of *bridging* and *mapping* (Figure 4.4). Merging of metamodels' conceptual elements is easy when we have the same concepts from both metamodels. The same concepts can be presented as a single concept in a unified/merged metamodel. If a pair of conceptual elements results as similar then we have to decide about their merging/integration after their ontological analysis. If the ontological analysis reveals that both concepts from different metamodels, although having the different syntax but have the same ontological meaning, then we can merge them as a single concept (as illustrated in Chapter 6). The *same* concepts can be merged to a single concept. *Similar* concepts can be *aligned* using *bridges* (Section 4.1.5) or *mappings* (Section 4.1.6).

5.5 Conclusion

In this chapter we presented a framework for the interoperability of metamodels. The framework consists of three major activities; linguistic analysis (Section 5.2), ontological analysis (Section 5.3) and interoperability (Section 5.4). The linguistic analysis of metamodels requires the metamodels' elements to be compared for their syntax (Section 5.2.1), semantics (Section 5.2.2) and structure (Section 5.2.3). Analysing the linguistic semantics of metamodels' elements requires a set of synonyms (synSet) for each pair of conceptual elements in both metamodels. We showed that the structural similarity

of any pair of conceptual elements is based on the aggregate similarity of their structural neighbours (Equation 5.3). The overall linguistic similarity can be computed using Equation 5.5.

Ontological analysis is then required to compare the metamodels' elements based on their definitions in the original specifications of the metamodels. The checks required for ontological analysis are discussed in Section 5.3. In the next chapter we will present the application of this framework to some exemplary metamodels. These were performed to validate and improve the framework based on the results obtained.

Chapter 6

Application of the Framework

It is the weight, not numbers, of experiments that is to be regarded.

Isaac Newton

In this chapter, we present five illustrations of the framework presented in Chapter 5. First, in Section 6.1, we apply the framework to two metamodels BPMN and OSM to show their interoperability. The rationale for selecting these metamodels as a first example is that: (1) these metamodels are endorsed by a single organization (OMG); but also, and probably more importantly, (2) one might conjecture that models emanating from a single organization would be more coherent than metamodels from different organizations. The second example (Section 6.2) shows the interoperability of some agent-oriented metamodels. In this example, we also show how the interoperability

framework can be applied to more than one pair of metamodels. This study is extended as a third example study by showing the merger of another pair of agent oriented metamodels (MaSE and Prometheus) (Section 6.3). The fourth example study, in Section 6.4, is about the interoperability of UML (Class Diagram) and Ecore metamodels. The fifth example study (Section 6.5) represents a scenario for which we need to have interoperable metamodels. Section 6.6 concludes the chapter. For the purpose of readability and convention, we have capitalized and italicized the concepts of metamodels (i.e. the class names) as is typically done in the quoted standards.

6.1 Example 1: BPMN and OSM

6.1.1 BPMN

BPMN (Business Process Modelling Notation) is the result of a standardization effort started in 2001 by BPMI (Business Process Management Initiative) and transferred to the OMG in 2006. BPMN is designed to support modelling of end-to-end business processes (OMG, 2011a). Three basic types of models in BPMN are *Processes* (both private and public), *Choreographies* and *Collaborations*. Private processes can be either executable or nonexecutable. *Collaborations* may include *Processes* and/or *Choreographies*.

Five basic categories of modelling elements used in a BPMN model are *Flow Objects, Data, Connecting Objects, Swimlanes* and *Artifacts*. *Flow Objects* are the main elements for defining the behaviour of a business process. Three types of flow objects are *Events*, *Activities* and *Gateways*. Data are represented as *Data Objects*, *Data*

Inputs, ***Data Outputs***, ***Data Stores*** and ***Properties***. Flow objects are connected to each other through ***Connecting Objects***. There are four different types of connecting objects: ***Sequence Flow***, ***Message Flow***, ***Association*** and ***Data Association***. ***Pools*** and ***Lanes*** are used to group modelling elements. ***Events*** and ***Gateways*** also have extended sets of elements.

The list containing all concepts in BPMN is presented in Appendix A. The BPMN metamodel is not depicted here as it is: (1) not a single diagram containing all conceptual elements of BPMN; and (2) such a large single diagram would require too much space.

6.1.2 OSM

The creation of OSM (Organization Structure Metamodel) (OMG, 2009b) is an ongoing project of the OMG to provide support for modelling organizational structures in modern enterprises. Modern organizations typically have a large variety of job titles and relationships such as groups, teams and communities, and enterprise relationships with other organizations.

OSM could be a point of integration for many other metamodels like BMM (Business Motivation Model), SBVR (Semantics of Business Vocabulary and Rules) and BPMN. The specification of OSM is not yet fully mature but it does define an initial metamodel for organizational structures worthy of discussion. This metamodel contains 11 classes (Appendix A), among which ***Participant*** **is the main generalized class with specialization classes of** ***Person***, ***OrgRole*** **and** ***OrgRelationship***.

The linguistic analysis (discussed in Section 5.2) of these two metamodels is elaborated in Sections 6.1.3 to 6.1.5. As we have seen, the linguistic analysis consists of syntactic analysis, semantic analysis and structural analysis. We will have a closer look at each of these in the following sections.

6.1.3 Syntactic Similarity of BPMN and OSM

In total, 155 BPMN concepts and 11 OSM concepts (Appendix A) were evaluated for their syntactic similarity. SSM values were then calculated using Equation 5.1. Table 6.1 lists only those concepts from both metamodels that have small values for edit distance ED. Small values of ED indicate greater similarity, e.g. *Participant* in OSM and *Participant* in BPMN have ED of zero, so they are syntactically the same. The values of SMM range from zero to one, where one indicates that the two concepts are exactly same and zero means there is a bad match.

From (Table 6.1), we can identify *Participant* and *Property* as being exactly the same in both metamodels. The pairs of concepts from both metamodels that have some syntactic similarity are, (*Person* and *Performer*), (*OrgRole* and *PartnerRole*), (*OrgRelationship* and *Relationship*), (*OrgRelationship* and *RelationshipDirection*), (*PersonSpec* and *Performer*), (*RelationshipSpec* and *Relationship*), (*RelationshipSpec* and *RelationshipDirection*) and (*PropertySpec* and *Property*). The table also includes some pair of concepts which were identified as a bad match. Table 6.1). The *same* and *similar* matches were further investigated for semantic and structural similarity.

Table 6.1: Syntactic similarity between BPMN and OSM

OSM	**BPMN**	**ED**	**SSM**	**Results**
Participant	Participant	0	1	Exactly Same
Participant	Participant Multiplicity	12	0	Bad Match
Participant	Participant Association	11	0	Bad Match
Property	Property	8	1	Exactly Same
Property	Correlatoin Property	11	0	Bad Match
Property	Correlation Property Binding	18	0	Bad Match
Property	Correlation Property Retrieval Expression	30	0	Bad Match
Person	Performer	5	1/6	Similarity
Person	HumanPerformer	10	0	Bad Match
OrgRole	PartnerRole	6	1/7	Similarity
Org Relationship	Relationship	3	3/4	Similarity
Org Relationship	Relationship Direction	12	1/5	Similarity
PersonSpec	Performer	5	4/9	Similarity
PersonSpec	HumanPerformer	10	0	Bad Match
Relationship Spec	Relationship	4	2/3	Similarity
Relationship Spec	Relationship Direction	7	9/16	Similarity
Spec	InputOutput Specification	20	0	Bad Match
Spec	Interaction Specification	20	0	Bad Match
Property Spec	Property	4	1/2	Similarity
Property Spec	Correlation Property	15	0	Bad Match

6.1.4 Semantic Similarity of BPMN and OSM

Semantic checks were made with the help of the lexical database WordNet (Miller, 1995). WordNet is a registered trademark for Princeton University and contains more than 118,000 word forms and 90,000 different word senses. The complete lists of synonyms for the concepts of both metamodels can be found in Appendix A. Table 6.2 lists pairs that have some similarity between their concepts in both metamodels and their synonyms. The semantic similarity is calculated using Equation 5.2. In Table 6.2, we can observe that only two concepts of OSM (*Participant* and *Property*) were repeatedly compared with the concepts of BPMN and their synonyms. We were not able to find synonyms for the remaining OSM concepts because of the way they are named, e.g. *OrgRole*, *OrgRelationship*.

The next step is to check the structural similarity between the conceptual elements of BPMN and OSM.

6.1.5 Structural Similarity between BPMN and OSM

Structural similarity between the concepts of OSM and BPMN was calculated using Equations 5.3 and 5.4. All concepts for both metamodels were extracted from XMI (XML metadata interchange) files for these metamodels. These sets of concepts for both metamodels were used as input to the code written to compute the neighbours (ancestor, sibling, immediateChild and leaf) of all the concepts in both metamodels. After identifying these sets for each individual concept in both metamodels, their similarity was calculated using Equation 5.4.

Table 6.2: Semantic similarity between BPMN and OSM

OSM_Concept	**OSM_Synonym**	**BPMN_Concept**	**BPMN_Synonym**
Participant	Associate	Participant	Associate
Participant	Contestant	Participant	Associate
Participant	Associate	Participant	Contestant
Participant	Contestant	Participant	Contestant
Property	Geographical area	Property	Geographical Area
Property	Possession	Property	Geographical Area
Property	Attribute	Property	Geographical Area
Property	Concept	Property	Geographical Area
Property	Geographical area	Property	Possession
Property	Possession	Property	Possession
Property	Attribute	Property	Possession
Property	Concept	Property	Possession
Property	Geographical area	Property	Attribute
Property	Possession	Property	Attribute
Property	Attribute	Property	Attribute
Property	Concept	Property	Attribute
Property	Geographical area	Property	Concept
Property	Possession	Property	Concept
Property	Attribute	Property	Concept
Property	Concept	Property	Concept

Table 6.3: Structural similarity (structSim) between BPMN and OSM

OSM	BPMN	Structural Similarity
Participant	Participant	0
Property	Property	0
Person	Performer	0.4
OrgRole	PartnerRole	0.23
OrgRelationship	Relationship	0.84
OrgRelationship	RelationshipDirection	0
PersonSpec	Performer	0
RelationshipSpec	Relationship	0.30
RelationshipSpec	RelationshipDirection	0
PropertySpec	Property	0

For overall linguistic similarity in this study, we have set the threshold to 50% (0.5), which means that we will consider the concepts having linguistic similarity of 50% or more. The values of coefficients of ancestors (Equation 5.4), siblings, immChild and leaf were set to 0.25 each because we think that all these measures are equally important in checking the structural similarity of two concepts. More than 3800 different permutations were computed for the concepts of OSM and BPMN. Table 6.3 presents the structural similarity values for those concepts that were identified as similar in the previous steps.

Table 6.4: Candidate concepts of BPMN and OSM for interoperability

OSM	BPMN
Participant	Participant
Property	Property
Person	Performer
OrgRole	PartnerRole
OrgRelationship	Relationship
OrgRelationship	RelationshipDirection
PersonSpec	Performer
RelationshipSpec	Relationship
RelationshipSpec	RelationshipDirection
PropertySpec	Property

Table 6.4 shows the candidate concepts identified as same or similar based on the last three steps. These results will be further refined by ontological analysis to check the exact contexts in both sets.

6.1.6 Ontological Analysis

For the candidate concepts in Table 6.4, we now examine the ontological semantics (Section 5.3) of the concepts identified as same or similar in the linguistic analysis. The following subsections illustrate these concepts and their relationships according to their definitions in

formal specifications to see whether they can realistically be mapped.

Participant

Participant in OSM (Figure 6.1) is defined as an element that can participate in a *Relationship* through roles. *Participants* have names, properties and effective dates. In BPMN, a *Participant* represents an entity or role that takes part in a *Collaboration* and is often responsible for the execution of a *Process* (Figure 6.2). When *Participants* are defined they are contained in an *InteractionSpecification*.

Careful examination of Figures 6.1 and 6.2 shows that *Participant* is not the same in BPMN and OSM, even though they were identified

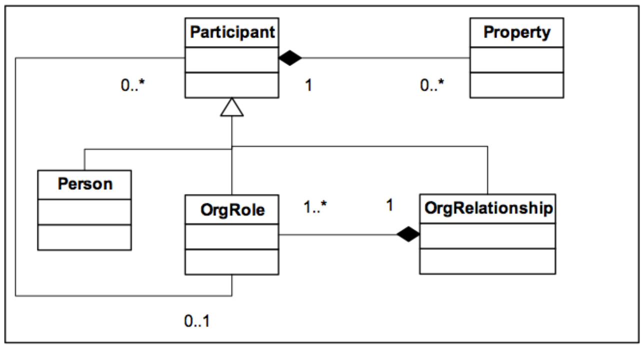

Figure 6.1: Participant in OSM

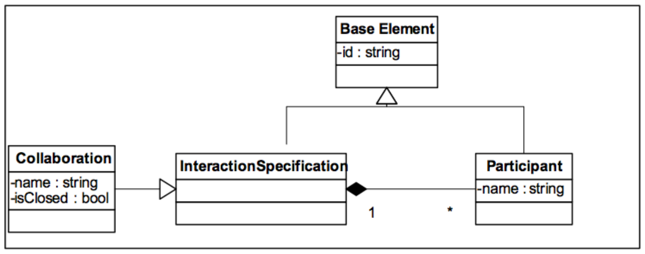

Figure 6.2: Participant in BPMN

as similar (Table 6.3). BPMN considers *Organization* as a set of *InteractionSpecifications* like *Collaboration* and treats *Participant* as a part of that interaction, so that every interaction in BPMN has one or more *Participants*. OSM treats *Organization* as a set of *Relationships*, defined as *OrgRelationship*, itself a special type of *Participant*. A more appropriate mapping alternative may be on the *Activity* level, which will be discussed in the following sections.

Property

In BPMN, certain flow elements, particularly *Processes*, *Activities* and *Events*, may contain *Properties* (Figure 6.3). The *Property* class is a *DataElement* that acts as a container for data associated with flow elements. *Property* elements must be contained within a *FlowElement* such as *Activity*.

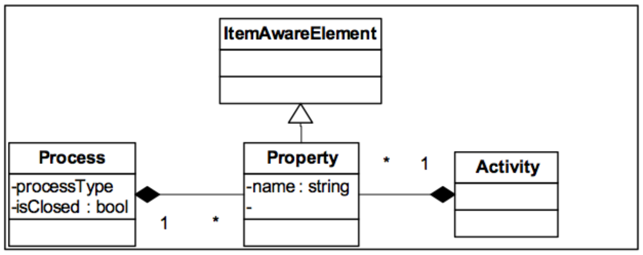

Figure 6.3: Property in BPMN

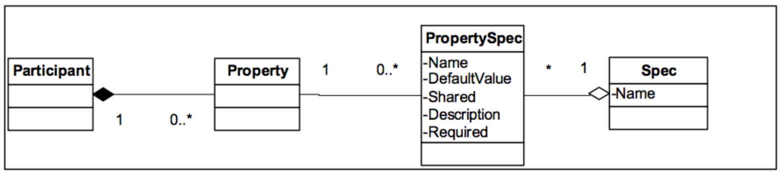

Figure 6.4: Property in OSM

In OSM, *Properties* are associated with model elements (Figure 6.4), providing information about the characteristics of the particular model element types. *Participants* may have *Properties*. *Person*, *OrgRole* and *OrgRelationship* are specialized classes of *Participant* (Figure 6.1), so these three elements may or may not have properties as well.

An interesting observation about OSM is that it offers the flexibility of assigning properties to all of its elements (*Person, OrgRole and OrgRelationship*), while BPMN assigns properties only to flow elements (*Activities, Events and Gateways*). We suggest that other nonflow elements like *Containers* should also have the flexibility to have properties. Another critical finding is that BPMN differentiates between properties and attributes of elements that, most of the time, are used interchangeably.

Person

***Person* in OSM was matched with *Performer* in BPMN (Table 6.4). OSM defines *Person* as an atomic element of an organizational structure and the whole organization model is defined as a relationship between *Persons*. A *Person* may have properties that qualify the**

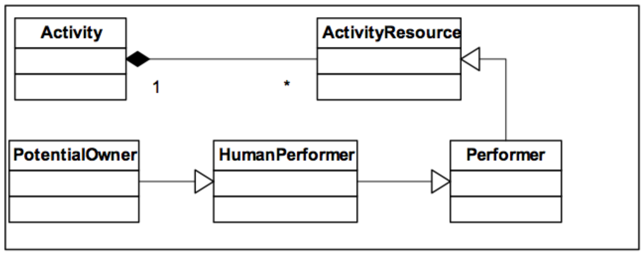

Figure 6.5: Performer in BPMN

entity to participate as an individual or as a particular role. Every *Person* participating in an *OrgRelationship* participates through an *OrgRole*, which defines the manner in which a *Person* can participate in a relationship.

***Performer* in BPMN (Figure 6.5) appears to be closer to *OrgRole* in OSM than to *Person*. *Performer* will perform or be responsible for an *Activity* and can be a specific individual, a group, an organizational role or position, or an organization. The BPMN 2.0 specification treats *Performer* as a role. Probably the more appropriate match in BPMN for *Person* in OSM is *HumanPerformer*, which is a specialized class of *Performer*. *HumanPerformer* can be assigned to any *Activity* in various roles, so it seems appropriate that *Person* in OSM should be matched to *HumanPerformer* in BPMN.**

OrgRole

***OrgRole* in OSM was syntactically matched to *PartnerRole* in BPMN (Table 6.4). OSM defines *OrgRole* as the manner in which a participant is involved with an *OrgRelationship*. BPMN defines *PartnerRole* as one of the possible types of *Participant*. The concepts seem to be similar but BPMN has exactly the same definition for *PartnerEntity*, although with no explanation of the actual difference. We deduce that *OrgRole* in OSM has the same semantics in BPMN as *Performer*, a generalized type of role (Figure 6.5) with specializations of *HumanPerformer* and *PotentialOwner*.**

OrgRelationship

OrgRelationship **(Organization Relationship) in OSM is similar to *Relationship* and *RelationshipDirection* in BPMN (Table 6.4). OSM defines an organization as a set of *Relationships* and provides one class, *OrgRelationship*, for all types of relationship. Keeping this definition in mind, we can say that this relationship is among *Participants*, *Persons* or *Roles* in the context of OSM, as discussed in previous sections. This view of relationship differs from that in BPMN where *Relationships* are defined as external relationships in terms of their relationship with non-BPMN artefacts like UML. As depicted in Figure 6.6, the *Relationship* class extends *BaseElement* with source and target *Elements* from the *Common Meta Object Facility*.**

OrgRelationships **in OSM have a closer semantic matching with what BPMN calls *InteractionSpecification*. Generally speaking, every *Process* in BPMN has a set of *FlowElements* including *Activities* and every *Process* is a specialization of *InteractionSpecification*, as depicted in Figure 6.7. Thus, we can conclude that *OrgRelationship* in OSM is similar to *InteractionSpecification* in BPMN.**

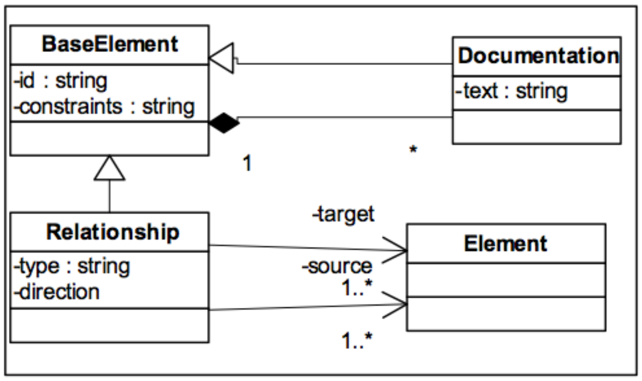

Figure 6.6: Relationship in BPMN

Specifications

In OSM, elements like *Person, OrgRole, OrgRelationship* and *Property* have specifications associated with them that describe their types and constraints. All these specifications (*PersonSpec, RoleSpec and RelationshipSpec*) are a special type of *Spec*.

BPMN defines only two types of specifications: *InteractionSpecification* (discussed earlier) and *InputOutputSpecification*. *InputOutputSpecification* aggregates *InputSet, DataInput, OutputSet* and *DataOutput* to capture data required by and produced by *Process* and *Activities*. *InputOutputSpecification* differs semantically from *Spec* in OSM as it specifies only input and output information. A more suitable match for *Spec* is *Documentation*. All BPMN elements that inherit from *BaseElement* have the capability, through *Documentation*, to have one or more text descriptions. Thus, OSM's *Spec* should be matched to *Documentation* of BPMN.

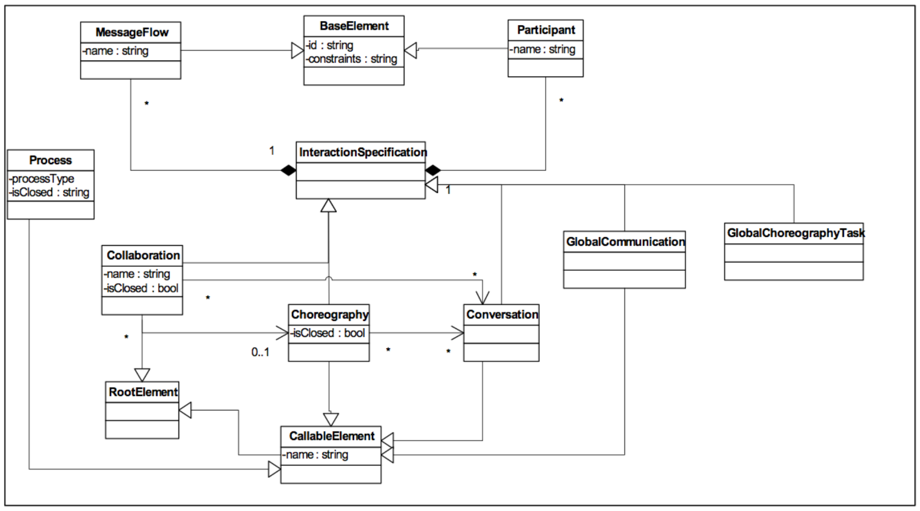

Figure 6.7: Interaction specification in BPMN

6.1.7 Interoperability of BPMN and OSM

Table 6.5: Matched concepts of BPMN and OSM after ontological analysis

OSM	BPMN
Property	Property
Participant	ActivityResource
Person	HumanPerformer
OrgRole	Performer
OrgRelationship	InteractionSpecifications
Spec	Documentation

Based on the discussion in previous sections, we are now confident of mapping elements from OSM to BPMN. Table 6.5 shows the concepts from both metamodels that will be merged as a result of ontological analysis. We can see the differences between the concepts from both metamodels in Table 6.5 and Table 6.4 that show how ontological analysis has radically changed the matching results. Figure 6.8 shows our proposed mapping between the OSM Metamodel and the part of the BPMN metamodel that contains relevant elements. The dashed lines between two metamodels show mappings of concepts from OSM to BPMN.

We have intentionally mapped elements of OSM to BPMN as a prelude to consideration of a possible merger rather than the other way around for two reasons. Firstly, the BPMN metamodel is more mature, formal and rich in elements than OSM. Secondly, the OSM metamodel is new and prone to changes in the future, so that any change in OSM will not adversely impact a merger when done in this direction.

In the next section we will illustrate the application of the framework to some agent-oriented metamodels.

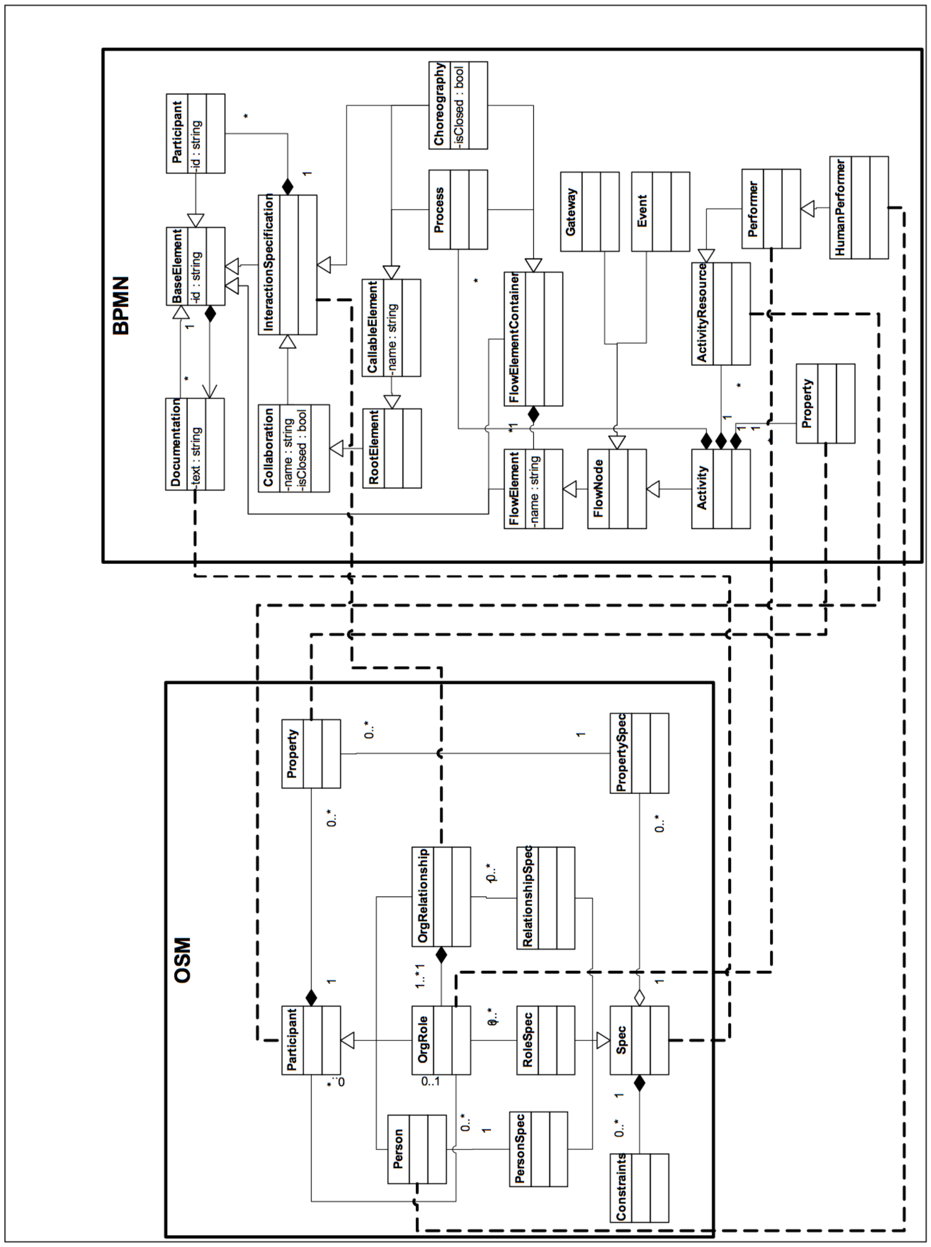

Figure 6.8: Interoperability of BPMN and OSM (dashed lines show mappings between elements of the two metamodels)

6.2 Example 2: MAS Metamodels

As we discussed in Chapter 5, the interoperability framework can be applied to any pair of metamodels. In this example, we extended the application of the framework in two different dimensions. Firstly, we have applied the framework to some multiagent systems (MAS) metamodels. An MAS is a system composed of multiple interacting agents. The agents in an MAS have several important characteristics, such as autonomy. Many methodologies have been proposed for multiagent systems like MESSAGE (Caire et al., 2001) and TROPOS (Castro et al., 2001). Secondly, this section shows the application of the framework to more than two metamodels (the number of metamodels in the previous example). We have applied the framework to three MAS metamodels; ADELFE (Bernon et al., 2003), Gaia (Zambonelli et al., 2003) and PASSI. The interoperability of two other agent-oriented metamodels MaSE (Garcia-Ojeda et al., 2008) and Prometheus (Winikoff and Padgham, 2004) is discussed as a separate example in Section 6.3.

The purpose of this illustration study was not to undertake any intermethodology comparisons, since that has been adequately covered in, for example, (Cernuzzi and Zambonelli, 2011; Dam and Winikoff, 2012; Welty, 2006). Nor are any uniting definitions of the constituent parts of an MAS, such as a definition for agent, proposed, since again such definitions have been widely discussed elsewhere, e.g. (Beydoun et al., 2009; Woolridge, 2001). Here we analyse the syntax and semantics of each of the selected methodologies on its own merits to seek successful interoperation.

Sections 6.2.1 to 6.2.3 briefly explain these three metamodels. Their syntactic analysis is presented in Section 6.2.4 followed by their semantic analysis in Section 6.2.5. Section 6.2.6 presents the ontological analysis and Section 6.2.7 discusses the interoperability of the elements found to be *similar* or the *same* in these three metamodels.

6.2.1 ADELFE

ADELFE (Bernon et al., 2003) is a methodology for adaptive multiagent systems (AMAS). Adaptive software is used in situations in which the environment is unpredictable. ADELFE is devoted to the software engineering of AMAS, using UML and AUML notations.

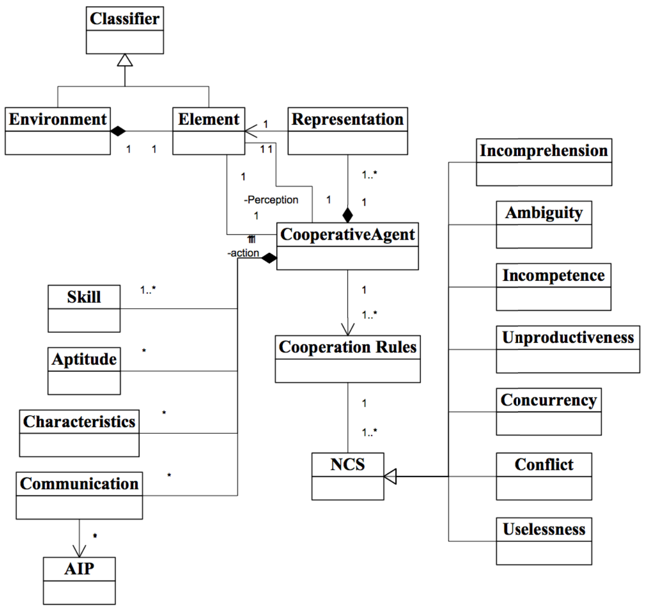

Figure 6.9: MAS metamodel adopted in ADELFE (Bernon et al., 2003)

ADELFE is built upon the Rational Unified Process (RUP), an iterative software development process framework created by the Rational Software Corporation (ISO/IEC19505, 2012). The metamodel adopted for ADELFE is explained in Figure 6.9. It is focused around *Cooperative Agents*. A *Cooperative Agent* has *Skills*, *Aptitudes* and *Characteristics*. Every *Cooperative Agent* has *Communication* with other agents and its *Environment*. An *Agent* has some *Cooperation Rules*. Besides local cooperation rules, there are some *NCS* (Non Cooperative Situations) which depict different types of cooperation failures. An agent possesses *Representations* as beliefs about other agents, itself and its *Environment*.

6.2.2 Gaia

The Gaia (Garo and Turci, 2003) methodology allows one to proceed systematically from requirements to design in a way that allows easy implementation. Some of the terminology and notation in Gaia is borrowed from object-oriented analysis and design. Gaia encourages the building of agent-based systems as a process of organizational design and is designed to model social aspects of open agent systems. This methodology is therefore focused on the organizational structure of the system (Figure 6.10).

An agent in Gaia is an entity that plays one or more *Roles*. A *Role* describes a specific behaviour played by the agent and is defined in terms of *Permissions*, *Responsibilities* and *Activities* and its interactions with other *Roles*. *Responsibilities* determine the functionality and are divided into *LivenessProperties* and *SafetyProperties*. *Permissions* are the rights associated with a Role and are used to realize

Responsibilities. ***Activities*** **of a** ***Role*** **are the computations associated with the** ***Role*** **that may be carried out without interacting with other** ***Agents.*** ***Protocol*** **represents the manner in which a** ***Role*** **can interact with other** ***Roles.***

6.2.3 PASSI

In PASSI , the multiagent system is depicted in both the problem and solution domains. The concepts of a multiagent system are divided into the problem domain, agency domain and solution domain (Figure 6.11). The problem domain contains concepts such as ***Requirements,***

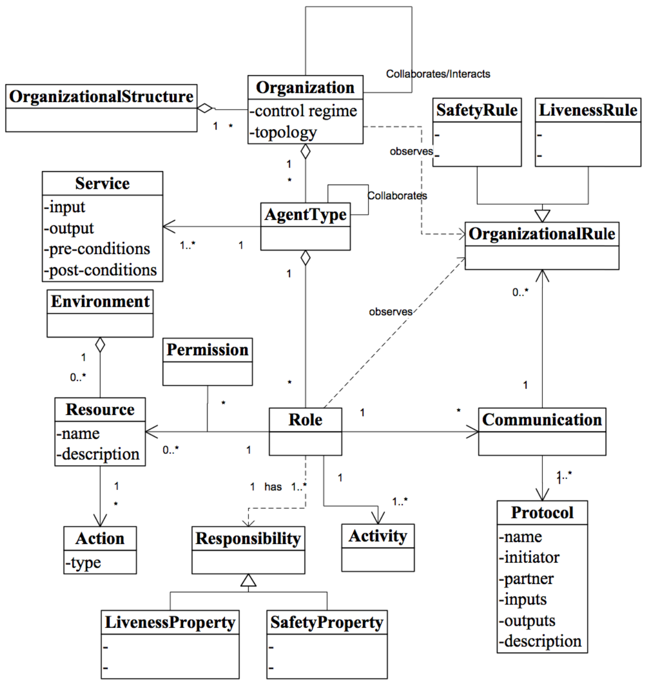

Figure 6.10: MAS metamodel adopted in Gaia (Bernon et al., 2003)

Scenarios, ***Nonfunctional Requirements*** and ***Ontologies.*** **The agency domain has major agent-oriented concepts such as** ***Agent***, ***Role***, ***Tasks***, ***Interactions*** **and** ***Communication.*** **The solution domain is concerned with the structure of the code solution, for instance the chosen FIPA-compliant implementation platform.**

6.2.4 Syntactic Similarity of ADELFE, Gaia and PASSI

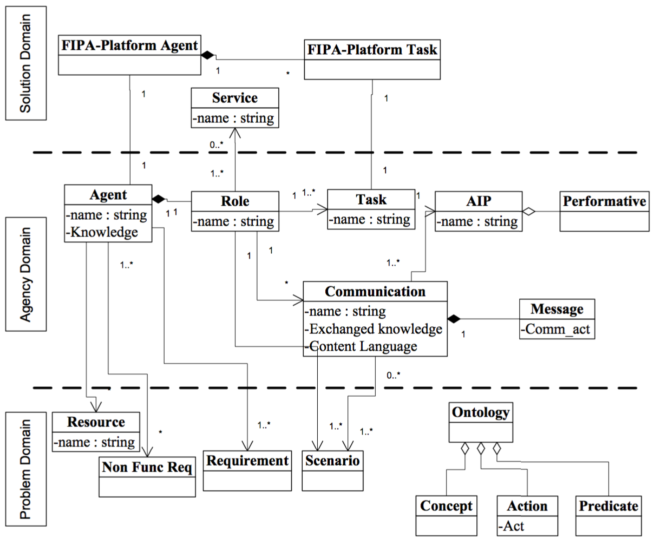

Figure 6.11: MAS metamodel adopted in PASSI (Bernon et al., 2003)

Table 6.6: Syntactic similarity between Gaia and ADELFE

Gaia Concept	ADELFE Concept	ED	SSM	Result
Organization	Representation	8	0.33	Similar
Organization	Communication	6	0.50	Similar
Organizational Rule	Cooperation Rule	10	0.41	Similar
Environment	Environment	0	1	Exactly the Same
Communication	Representation	9	0.31	Similar
Communication	Communication	0	1	Exactly the Same
Responsibility	Representation	10	0.29	Similar
Activity	Aptitude	5	0.38	Similar
Activity	Ambiguity	4	0.50	Similar

Conceptual elements of ADELFE, Gaia and PASSI were first identified from their metamodels. The syntactic similarity analysis (SSM) between the concepts of these metamodels was calculated by first comparing the concepts of Gaia with ADELFE, then the concepts of Gaia with PASSI and finally the concepts of ADELFE with PASSI. Tables 6.6, 6.7 and 6.8 present the SSM values between these metamodels. For efficient use of space, we have shown only the matchings

Table 6.7: Syntactic similarity between Gaia and PASSI

Gaia Concept	**PASSI Concept**	**ED**	**SSM**	**Result**
Organization	Communication	6	0.50	Similar
Service	Service	0	1.00	Exactly the Same
Environment	Requirement	6	0.45	Similar
Resource	Resource	0	1.00	Exactly the Same
Role	Role	0	1.00	Exactly the Same
Communication	Communication	0	1.00	Exactly the Same
Action	Action	0	1.00	Exactly the Same
Activity	Action	4	0.33	Similar

Table 6.8: Syntactic similarity between ADELFE and PASSI

ADELFE Concept	**PASSI Concept**	**ED**	**SSM**	**Result**
Environment	Requirement	6	0.45	Similar
Representation	Communication	9	0.31	Similar
Cooperative Agent	FIPA-Platform Agent	11	0.35	Similar
Communication	Communication	0	1.00	Exactly the Same
AIP	AIP	0	1.00	Exactly the Same
Conflict	Concept	4	0.43	Similar

Table 6.9: Syntactically same concepts among Gaia, ADELFE and PASSI

Concept	**Gaia**	**ADELFE**	**PASSI**
Communication	Same	Same	Same
Service	Same		Same
Resource	Same		Same
Role	Same		Same
Action	Same		Same
AIP		Same	Same
Environment	Same	Same	

that have a value of SSM more than 0.30 (30% similarity). Table 6.6 shows that, for Gaia/ADELFE, only *Environment* and *Communication* are the *Same*. The rest are either *Similar* or *Bad Match* (not shown in the table). For Gaia and PASSI, *Service*, *Resource*, *Role*, *Communication* and *Action* are found to be exactly the same (Table 6.7), whilst for ADELFE/PASSI, only *Communication* and *Agent Interaction Protocol (AIP)* are exactly the same (Table 6.8). As a summary, Table 6.9 presents the concepts identified as syntactically the same in these metamodels.

After identifying syntactically the *same* concepts in these metamodels, the next step was to identify semantic similarity between the concepts of all three metamodels. Careful examination of Table 6.9 shows that there is low similarity between the concepts of ADELFE and the other two metamodels. Only two concepts of ADELFE (*Communication* and *AIP*) are (syntactically) matched to the same concepts in PASSI and two of its concepts (*Communication* and *Environment*) have a *same* syntactic match in Gaia. Therefore, we can expect better interoperability between Gaia and PASSI although this needs to be verified by semantic mapping of these concepts.

6.2.5 Semantic Similarity of Gaia, ADELFE and PASSI

Table 6.10: Semantic similarity between Gaia and ADELFE

Gaia Concept	Gaia Synonym	ADELFE Concept	ADELFE Synonym
Environment	Surrounding	Environment	Surrounding
Communication		Communication	

Semantic mapping was made with the help of WordNet (Miller et al., 1998). Using WordNet, synonyms for concepts in all three metamodels were identified. As a next step, we identified the similarities between concepts and/or their synonyms in these metamodels. The objective was to check whether there is some semantic mapping between these concepts. Tables 6.10, 6.12, 6.11 list synonym pairs that

Table 6.11: Semantic similarity between Gaia and PASSI

Gaia Concept	Gaia Synonym	PASSI Concept	PASSI Synonym
Service	Overhaul	Service	Overhaul
Role	Function	Role	Function
Role	Character	Role	Function
Role	Function	Role	Character
Role	Character	Role	Character
Action	Activity	Action	Activity
Activity	Action	Action	Activity
Activity	Process	Action	Activity
Action	Activity	Action	Carry through
Action	Carry through	Action	Carry through
Communication		Communication	
Resource		Resource	

Table 6.12: Semantic similarity between ADELFE and PASSI

ADELFE Concept	PASSI Concept
AIP	AIP
Communication	Communication

Table 6.13: Candidate concepts for merging among Gaia, ADELFE and PASSI

Gaia	PASSI	ADELFE
Communication	Communication	Communication
Service	Service	
Resource	Resource	
Role	Role	
Action	Action	
	AIP	AIP
Environment		Environment
Activity	Action	

have syntactic and/or semantic similarity for these three pairings.

Potential concepts were then identified for merging following both syntactic and semantic mappings among the three metamodels. This step should be refined by human intervention to check the exact context in these metamodels.

6.2.6 Ontological Analysis

For the candidate concepts identified in the last two steps, we examined their places in the definitions of the metamodels. The following subsections illustrate these concepts according to their definition to see whether they can be mapped successfully.

Communication

In the agent-oriented paradigm, agents are intelligent entities that interact and communicate with other agents and the environment. *Communication* as a concept is matched both syntactically and semantically in all three metamodels (ADELFE, Gaia and PASSI). *Communication* in PASSI is defined as an interaction between two agents. Usually it is composed of more than one message, each associated with one *Performative*. Gaia has an *Acquaintance Model* to define the *Communication* links that exist between agent types. It does not define what messages are sent or when messages are sent; it simply indicates that communication pathways exist.

Both Gaia and PASSI associate *Communication* with the *Role* an agent plays while *ADELFE* associates it directly with the *Agent (CooperativeAgent)*. Indeed, we have observed in the syntactic and semantic mapping of these concepts that Gaia and PASSI have a greater tendency to be interoperable. We therefore suggest associating Communication with *Role* (Figure 6.12). We think that an agent may initiate or participate (through *Roles*) in one or more communications but it is not composed of *Communications* (as in ADELFE). Thus, an *Agent* through its *Role* participates or initiates one or more *Communications*.

Another important observation is that *Communication* in all three metamodels has one or more *Protocol(s)*, which, in the case of ADELFE and PASSI, is the *Agent Interaction Protocol (AIP)*, while Gaia just defines *Protocol*. We therefore recommend that every *Communication* should have one or more *AIPs*.

Agent

Agent, a basic entity in the agent-oriented paradigm, is defined with different names in these three metamodels: *CooperativeAgent* in ADELFE, *AgentType* in Gaia and *Agent* in PASSI. Every agent is by nature cooperative and there is no need to separate it from the generic term of *Agent*. *AgentType* (Gaia) limits the scope of an *Agent*, as there can be more than one agent type, e.g. interface agent, information agent. We propose that a more suitable relationship could be that an *Agent* has a type denoted by *AgentType*. In our merged metamodel (Figure 6.12), we have used the term *Agent* rather than *AgentType* or *CooperativeAgent*, where every *Agent* can have an *AgentType*.

Service

Service in Gaia was matched with *Service* in PASSI both syntactically and semantically. Gaia defines *Service* as a function of an *Agent*. In PASSI, a service is a coherent block of activity in which an *Agent* will engage. It should be clear that every activity identified at the analysis stage corresponds to a service, though not every service will correspond to an activity. *Services* are associated with *Roles* in both Gaia and PASSI while ADELFE has no concept like *Service*.

We therefore recommend that, for the purpose of unification, *Services* should be associated with *Roles*. Another important observation is that an *Agent* in an agent-oriented context is a cooperative entity and has to provide services. Consequently, the relationship between a *Role* and a *Service* should be 1..* as compared with 0..* in both PASSI and the Bernon et al. (2004) unified metamodel.

Another difference from the Bernon et al. (2004) unified metamodel is that *Services* are associated with *Agents*; in fact they should be associated with *Roles* according to the descriptions of both these metamodels (Figure 6.12).

Resource

Resource in Gaia was syntactically and semantically matched with *Resource* in PASSI. In Gaia, every *Role* has associated *Permissions*. These *Permissions* are used to identify *Resources* that are available to that *Role* to help it realize its *Responsibility*. A *Resource* in PASSI is defined as an entity that can be acquired, shared or produced by an *Agent*. In the (Bernon et al., 2004) unified metamodel, *Resource* is attached to *Role*. We suggest attaching it to *Agent* for clarity. An agent may or may not have resources available to perform some activity.

Role

Both Gaia and PASSI have a syntactically and semantically matched concept: *Role*. *Role* in PASSI is a portion of the social behaviour of an agent that is characterized by a goal and/or provides a functionality or service. A *Role* is a collection of tasks performed by an agent in pursuing a subgoal; an agent can play one or more roles in the system. Each role describes an aspect of the agent's life cycle and is often related to a service offered by the agent to the society or to the achievement of one of its goals. *Role* in Gaia defines what is expected to be undertaken in the organization. An Agent's *Role* is defined in terms of the specific tasks that it has to accomplish in the context of the overall organization. *Role* is defined by *Protocols*, Activities, *Permissions* and *Responsibilities*. Consequently, both these concepts can be merged. An agent can perform one or more *Roles* to realize its goals or to perform its activities.

An important observation is that *Role* in Gaia has an aggregation relationship with *Agent* while *Role* in PASSI has a composition relationship with *Agent*. In our merged metamodel we have used aggregation because a *Role* can be a part of an *Agent* but it is not necessary for an *Agent* to be composed of *Roles*.

Action

Action in Gaia was syntactically and semantically matched with *Action* in PASSI while ADELFE has no concept like *Action*. Action in PASSI and Gaia has different semantics according to their definitions in their specifications. In PASSI, *Action* is a composite part of *Ontology* while in Gaia every *Resource* has permitted *Actions*. We thus consider that it is not possible to merge the various forms of *Action*. Rather, it is better to bridge them with both *Resource* and *Ontology* as was also done in the Bernon et al. (2004) unified metamodel.

Environment

Environment was matched both syntactically and semantically in ADELFE and Gaia, but it has a different representation in the two metamodels. In Gaia, it is a part of *Resource* while in ADELFE it is a part of *Element*, every *Representation* having an *Element* and *Representation* being a part of *CooperativeAgent*.

We have concluded that *Environment* in ADELFE and *Environment* in Gaia cannot be merged. To make them interoperable we have created a bridge between *Environment*, *Agent* and *Resource*. It is a little different from the similar relationship shown in the (Bernon et al., 2004) unified metamodel as there is a relationship between *Ontology* and the *Environment* that they neither explained nor justified from these metamodels.

Activity and Task

Task in *PASSI* and *Activity* in Gaia were not matched either syntactically or semantically due to their name clash, but close observation shows that they refer to the same concept in these two different metamodels. We therefore suggest that *Activity* in Gaia should be merged with *Task* in PASSI (Figure 6.12).

6.2.7 Interoperability of Gaia, ADELFE and PASSI

Based on the discussion in the previous sections, we are now confident that we have an interoperable metamodel for ADELFE, Gaia and PASSI. Figure 6.12 shows our proposed metamodel (bottom right section) that contains relevant elements, where dashed lines between the three metamodels are used to show mapping of concepts from ADELFE, Gaia and PASSI to the merged metamodel.

The merged metamodel shown in Figure 6.12 is partial and shows only elements that depict these merging, mapping and bridging concepts, e.g. the concept *Communication* was the same in all three metamodels and was merged as a single concept. Similarly, *Resource* in Gaia was bridged with *Ontology* of PASSI through *Action*, which was common (but with different semantics) to the two metamodels. Mapping can be done by attaching the element from the original metamodel to an element in the merged metamodel while preserving the same relationships as in the original metamodel, e.g. *FIPA Platform Task* was associated with *Task* in PASSI and can be mapped to *Task* in the merged metamodel.

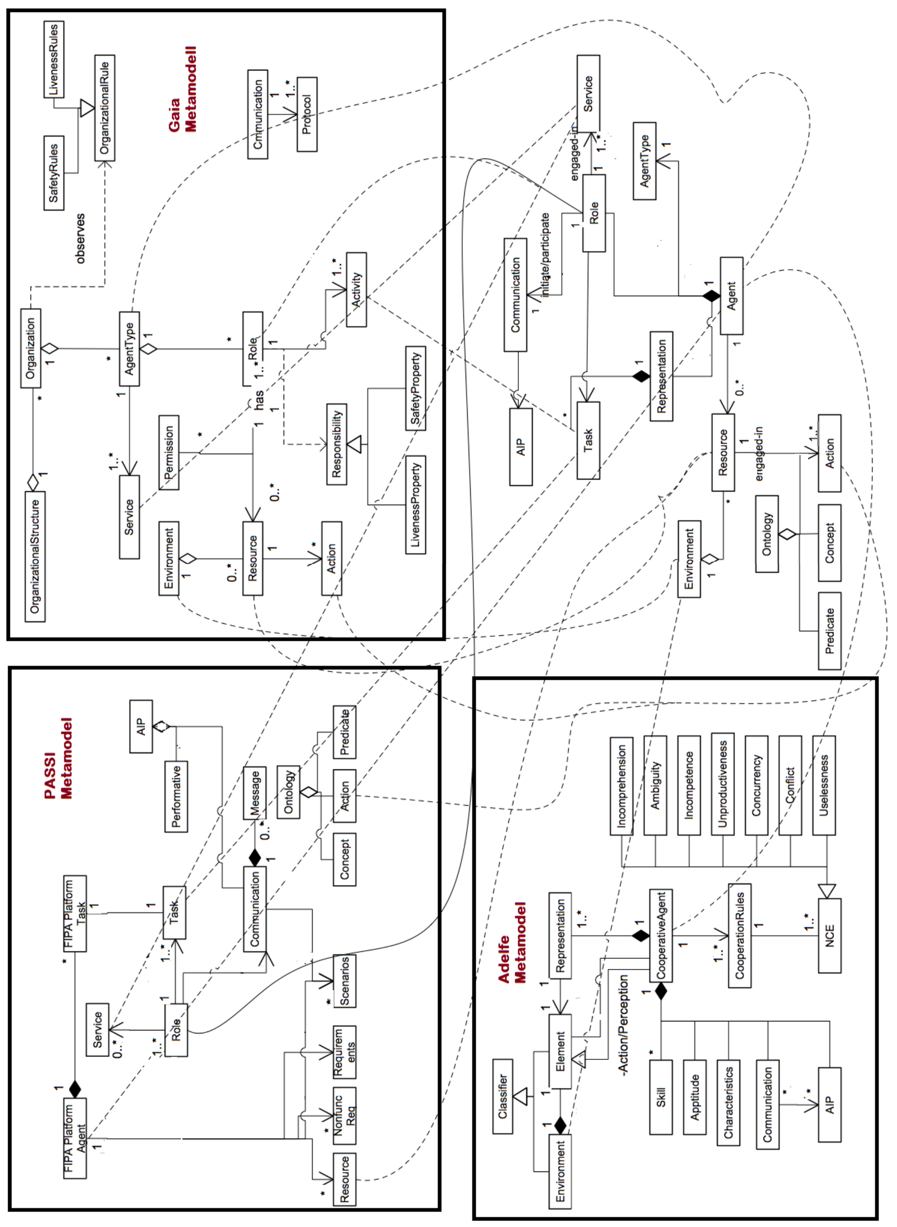

Figure 6.12: Mapping/merging of concepts to our proposed merged metamodel

Merging the above three metamodels (ADELFE, Gaia and PASSI) has resulted in a metamodel that is similar to the unified metamodel produced by (Bernon et al., 2004) but has an ontological rather than a subjective basis. This similarity between the two derived metamodels validates our technique.

6.3 Example 3: MaSE and Prometheus

Merging the Gaia, ADELFE and PASSI metamdoels give us the confidence to apply it to other metamodels. We have thus applied the framework to two other metamodels: MaSE and Prometheus. Sections 6.3.3 to 6.3.5 briefly explain the results of matching between both metamodels.

6.3.1 MaSE

MaSE is an organizationally based multiagent system engineering process framework (Garcia-Ojeda et al., 2008). In MaSE, the *Organization* is composed of five entities *Goals*, *Roles*, *Agents*, *Domain Model* and *Policies* (Figure 6.13). A *Goal* defines the overall function of the organization and a *Role* defines a position within an organization with behaviour that is expected to achieve a particular goal or set of goals.

6.3.2 Prometheus

Prometheus is a general-purpose methodology for multiagent systems to support BDI (Belief, Desire, Intention) systems. The methodology has three phases: system specification, architectural design and

detailed design (Winikoff and Padgham, 2004). Each phase includes models, focusing on the dynamics of the system. The details of the concepts included in Prometheus are shown in Figure 6.14.

6.3.3 Syntactic Similarity of MaSE and Prometheus

Table 6.14 presents those concepts from both metamodels that have syntactic similarity of 50% or more. There are eight different concepts in both metamodels that are exactly the same while three other concepts (the last three rows of Table 6.14) have 50% or more syntactic similarity. It is important to note here that the syntactic similarity between MaSE and Prometheus is higher than the syntactic similarity between ADELFE and Gaia (Table 6.9) in the previous example.

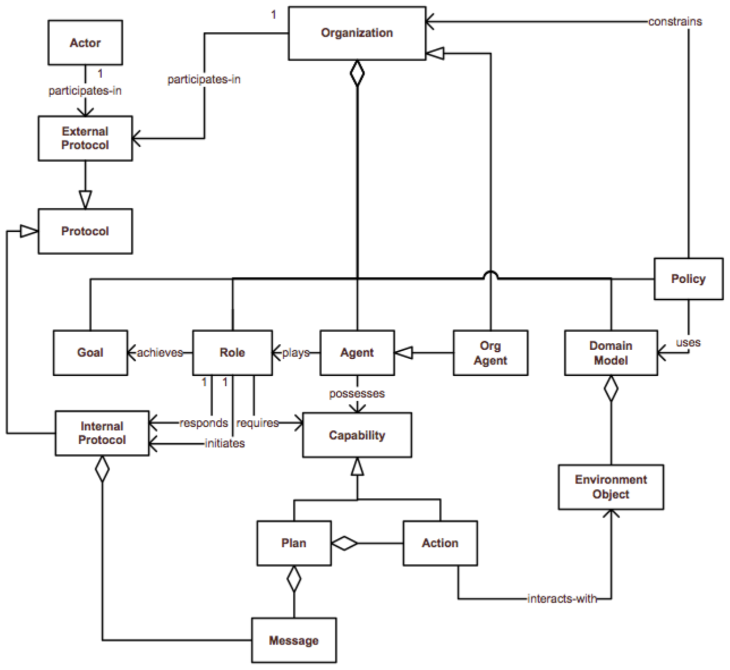

Figure 6.13: MAS metamodel adopted in OMaSE

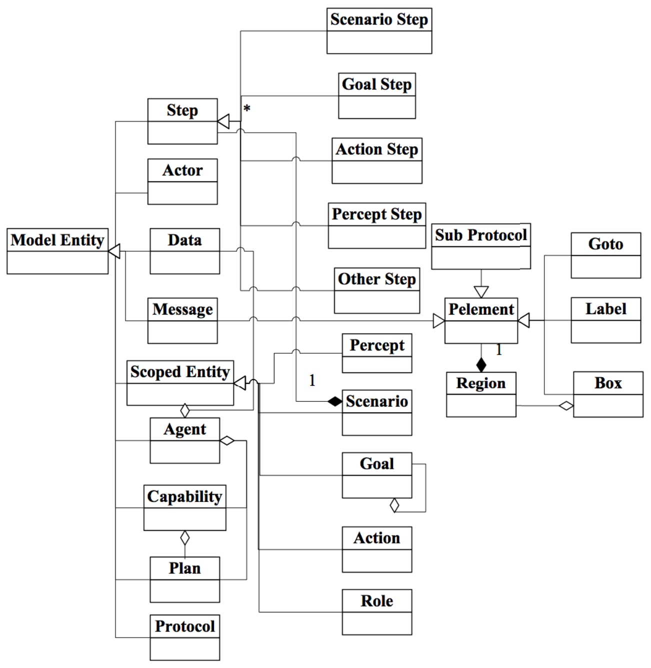

Figure 6.14: MAS metamodel adopted in Prometheus

Table 6.14: SSM between MaSE and Prometheus

MaSE Concept	**Prometheus Synonym**	**ED**	**SSM**	**Result**
Message	Message	0	1	Exactly the same
Goal	Goal	0	1	Exactly the Same
Capability	Capability	0	1	Exactly the Same
Agent	Agent	0	1	Exactly the Same
Protocol	Protocol	0	1	Exactly the Same
Role	Role	0	1	Exactly the Same
Action	Action	0	1	Exactly the Same
Plan	Plan	0	1	Exactly the Same
Action	Actor	2	0.6	Similar
Action	Region	3	0.5	Similar
Protocol	SubProtocol	4	0.5	Similar

The next section elaborates the semantic similarity between MaSE and Prometheus.

6.3.4 Semantic Similarity of MaSE and Prometheus

Semantic similarity between the elements of MaSE and Prometheus was calculated using synonyms from WordNet (Miller, 1995), as discussed previously. Table 6.15 presents the synonyms found for different MaSE concepts. Most of the concepts in MaSE and Prometheus are *exactly the same* (Table 6.14); thus we have presented only the synonyms of the MaSE concepts to avoid redundancy.

Table 6.15: Synonyms for concepts in MaSE

MaSE Concept	**MaSE Synonym**
Actor	Worker, Doer, Player
Goal	End, Finish, Destination
Role	Character, Function, Office Part
Agent	Broker, Factor
Capability	Capacity, Potentiality
Plan	Program, Design, Contrive
Action	Activity, Process
Organization	Arrangements, Administration, Constitution

Table 6.16: Candidate concepts of MaSE and Prometheus for merging

MaSE Concept	**Prometheus Concept**
Agent	Agent
Role	Role
Action	Action
Capability	Capability
Protocol	Protocol
Internal Protocol	Sub Protocol
External Protocol	Sub Protocol
Goal	Goal
Plan	Plan
Message	Message

The semantic similarity between concepts and/or their synonyms in both metamodels (MaSE and Prometheus) does not reveal any new matchings. This means that we could not find any concept in either of the metamodels that is represented with a different name (semantics) in the other metamodel. Table 6.16 presents the candidate concepts from both metamodels that can be merged or mapped with each other. These concepts from both metamodels were further evaluated by human intervention. Some of the concepts were found not to be similar in the metamodels; we have to accommodate them as well while structuring the merged metamodel. Some of these concepts are discussed in the following subsections.

6.3.5 Ontological Analysis

After identifying syntactic and semantic similarities between the concepts of both metamodels, we examined them for their semantics as defined in their specifications . Most of the conceptual elements are *exactly the same* (Table 6.14) and belong to the same domain (Multi-agent Systems); hence their semantics in both metamodels are almost the same. These conceptual elements are discussed below.

Organization and Domain Model

MaSE is an organization-based methodology for multiagent systems. Each key concept including the *Domain Model* in MaSE is a part of the *Organization* (Figure 6.13). In Prometheus, every key concept is a subclass of the *Model Entity* (Figure 6.14). We have combined these in the merged metamodel as *Organization*, which has a *Domain Model*, whilst different *Model Entities* are part of the *Domain Model*

(Figure 6.15).

Agent

In MaSE, an *Agent* is a part of the *Organization* and a specialized class of *Environment Object* (Figure 6.13), while an *Agent* in Prometheus (Figure 6.14) is a subclass of *Model Entity.* We have combined them as *Agent*, being a specialized class of *Model Entity*, whilst *Model Entity* is a part of *Domain Model* and *Domain Model* itself is a part of *Organization.* It is important to note that *Agent* in MaSE has a subclass *OrgAgent* that represents the organizational agent. We have maintained this relationship in the merged metamodel (Figure 6.15) with *OrgAgent* as a specialized class of *Agent.*

Action

Action was syntactically matched as being exactly the same in both metamodels but also has similarity with *Actor* and *Region. Action* is a part of *Capability* in MaSE (Figure 6.13) and *Capability* is a subclass of *Model Entity* in Prometheus (Figure 6.14); we have treated *Action* as a single concept in our merged metamodel (Figure 6.15) that is a part of the *Capability* class and *Capability* itself as a subclass of *Model Entity.*

Goal and Role

Goal and *Role* are both treated as a part (aggregation) of the *Organization* class in MaSE. In contrast, these two classes are represented as subclasses of *Scoped Entity* in Prometheus, where *Scoped Entity* is a subclass of *Model Entity* (Figure 6.14). In the merged metamodel (Figure 6.15), we have treated them as a subclass of *Model Entity.*

Model Entity **is a part of *Organization* (through *Domain Model*) and every subclass of *Model Entity* is consequently also a part of *Organization*.**

Protocol

MaSE has a class *Protocol* with two specialized classes *Internal Protocol* and *External Protocol* (Figure 6.13), while Prometheus has a *Sub Protocol*, which is a specialized class of the *Pelement* class (Figure 6.14). *Sub Protocol* can be treated as a specialized subclass of *Protocol*. We have organized these classes from both metamodels as a *Protocol* with subclasses of *Internal Protocol* and *External Protocol* (Figure 6.15). Table 6.16 shows the finalized conceptual elements from both metamodels for merging/mapping.

Figure 6.15 shows the merged metamodel for MaSE and Prometheus. Most of the concepts in Table 6.16 were a straightforward merger into a single concept. Both Figures 6.16 and 6.17 depict the mapping of concepts from MaSE and Prometheus to a merged metamodel. The dashed lines show the merging/mapping of concepts from both

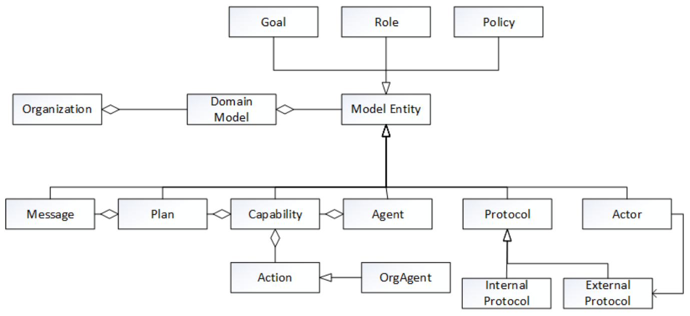

Figure 6.15: Merged metamodel for MaSE and Prometheus

metamodels into the merged metamodel.

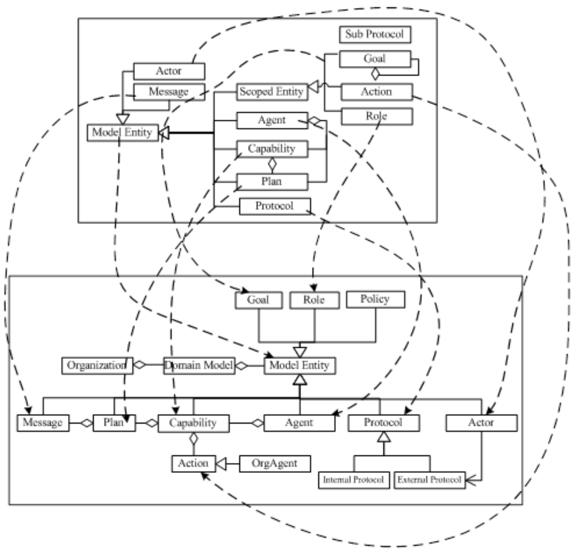

Figure 6.16: Mapping from MaSE to merged metamodel

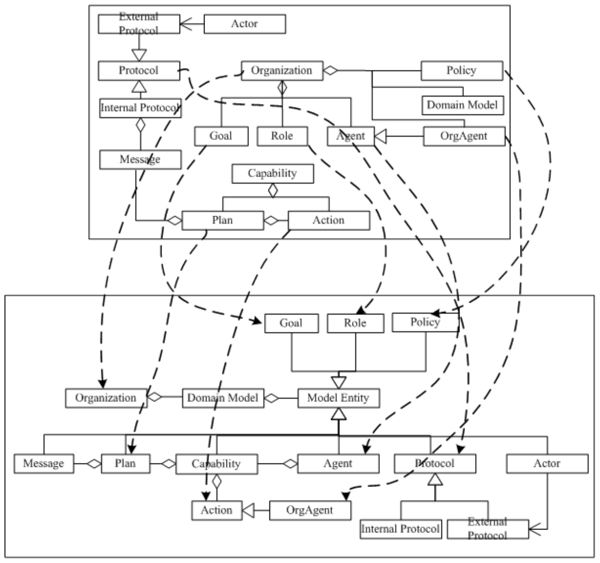

Figure 6.17: Mapping from Prometheus to merged metamodel

in the next section we will demonstrate the results of another example that applies the interoperability framework to merge Ecore and UML metamodels. For this example we will match Ecore with the class diagram metamodel of UML 2.0.

6.4 Example 4: Ecore and UML

6.4.1 Ecore and UML

Ecore is a metamodel for EMF (Eclipse Modelling Framework). EMF is a modelling framework and code-generation facility for building tools and other applications based on a structured data model. The Ecore metamodel contains information about the defined classes. Ecore allows to define different elements, e.g. *EClass*, *EAttribute*, *EReference* and *EDataType*. *EClass* represents a class, with zero or more references. *EAttribute* represents an attribute which has a name and a type. *EReference* represents one end of an association between two classes. It has a flag to indicate whether it represent a containment and a reference class to which it points. *EDataType* represents the type of an attribute.

The Ecore model has a root *Object* representing the whole model. This model has children which represent the packages, their children represent the classes, while the children of the classes represent the attributes of these classes. Figure 6.18 shows the Ecore metamodel. We have intentionally omitted the letter *E* before class names to make the figure more readable.

The UML metamodel (class diagram) has 35 classes with *Element* as a root class. *Class* is a specialised class of *Namespace* which itself is a specialisation of *Element*. Figure 6.19 shows the metamodel for the class diagram of UML 2.0. We will refer to this metamodel as the UML2CD metamodel (Metamodel of UML 2 Class Diagram).

6.4.2 Syntactic Similarity of ECore and UML

Table 6.17 shows the results of our syntactic similarity analysis between Ecore and UML2 CD. The table shows only those concepts which have more than 50% similarity (0.50). Nine concepts were identified as *exactly the same* having SSM = 1, while six concepts of Ecore were found to be similar (have 50% or more similarity) to concepts of UML2CD.

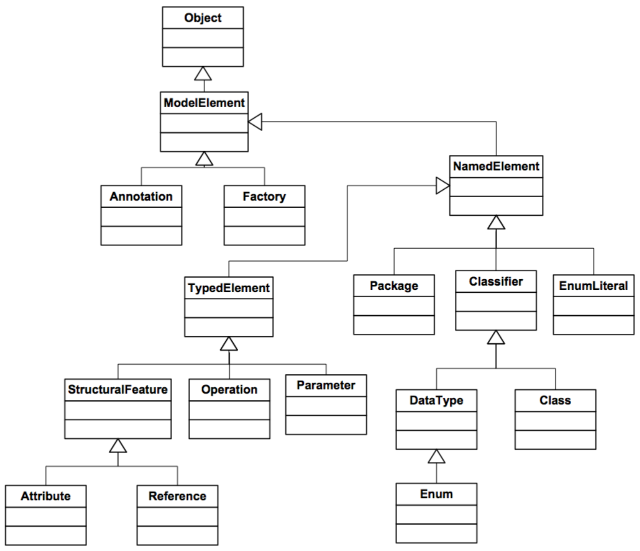

Figure 6.18: Ecore metamodel

The concept *ModelElement* in Ecore was found similar to both *NamedElement* and *TypedElement* in UML. *Annotation* in Ecore was found

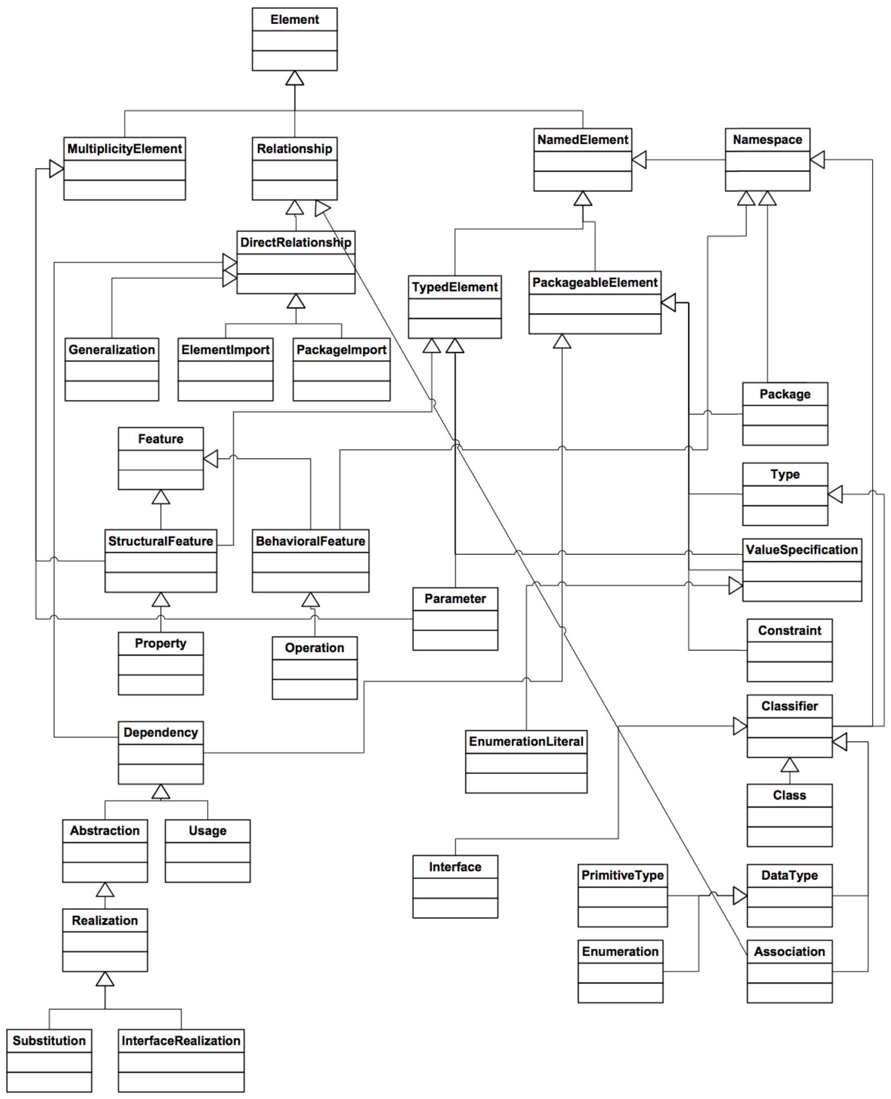

Figure 6.19: UML class diagram metamodel

similar to *Association, Abstraction and Enumeration* in UML2CD. These concepts require further ontological analysis for the best match.

6.4.3 Semantic Similarity of ECore and UML

The synonyms of all 17 concepts of Ecore and 35 concepts of UML2CD were found using WordNet. Each concept and its synonyms in Ecore were then compared with all the concepts and their synonyms in UML2CD. Table 6.18 shows the concepts from both metamodels

Table 6.17: Syntactic similarity between Ecore and UML2CD

Ecore	**UML2CD**	**SSM**
Parameter	Parameter	1
Class	Class	1
Package	Package	1
DataType	DataType	1
TypedElement	TypedElement	1
Operation	Operation	1
StructuralFeature	StructuralFeature	1
Classifier	Classifier	1
NamedElement	NamedElement	1
ModelElement	NamedElement	0.66
ModelElement	TypedElement	0.66
Annotation	Association	0.60
StructuralFeature	BehavioralFeature	0.58
Operation	Enumeration	0.55
Annotation	Abstraction	0.50
Annotation	Enumeration	0.50

Table 6.18: Semantic similarity between Ecore and UML2CD

Ecore	**UML2CD**	**SemSim**
Parameter	Parameter	0.15
Reference	Relationship	0.14
Package	Package	0.13
Parameter	Constraint	0.13
Reference	Constraint	0.13
Reference	Comment	0.10
Parameter	Relationship	0.10
Reference	Association	0.10

which have 10% or more semantic similarity. It is apparent from the table that the overall similarity between the synonyms of the concepts in both metamodels is very low. It is also to be noted that the concepts *Parameter* and *Package* in Table 6.18 have already been identified as the same (Table 6.17), so we will treat them as *same*. The concept *Reference* has a semantic similarity with *Relationship, Constraint, Comment* and *Association.* We need to further analyse it for suitable mappings to any of these concepts.

6.4.4 Structural Similarity of ECore and UML

Table 6.19 shows the structural similarity between the concepts of Ecore and UML2CD. The table shows concepts having more than 40% structural similarity. Based on the results of all three activities *Syntactic Analysis*, *Semantic Analysis* and *Structural Analysis*, we can filter the candidate concepts for unification in both metamodels.

Table 6.20 shows the candidate concepts from both metamodels that can be either merged, bridged or mapped. We will conduct an *ontological analysis* of these concepts to find their mappings.

6.4.5 Ontological Analysis

In this section we will analyse the identified concepts ontologically. Ontological analysis involves the definitions of these concepts from their specifications. We describe this ontological analysis differently from those for the previous example, where we analysed each concept separately. In this section we have divided the analysis based on whether the concepts can be merged, bridged or mapped.

Merging

From Table 6.20, we can see that the concepts *NamedElement*, *TypedElement*, *Package*, *Classifier*, *StructuralFeature*, *Operation*, *Param-*

Table 6.19: Structural similarity between Ecore and UML2CD

Ecore	**UML2CD**	**StructSim**
Class	Class	0.45
DataType	DataType	0.44
DataType	Interface	0.44
Class	Interface	0.44
TypedElement	TypedElement	0.42
Classifier	Namespace	0.40
EnumLiteral	PackageableElement	0.39
Classifier	PackageableElement	0.39
Package	Namespace	0.39
EnumLiteral	Namespace	0.39
Package	PackageableElement	0.39
TypedElement	Namespace	0.36
Classifier	TypedElement	0.36
TypedElement	PackageableElement	0.35
EnumLiteral	TypedElement	0.35
Package	TypedElement	0.34
Class	DataType	0.34
Attribute	Property	0.33
Reference	Property	0.33
Enum	PrimitiveType	0.33
DataType	Class	0.33
Enum	Enumeration	0.33
Annotation	Namespace	0.30
Factory	Namespace	0.30
Annotation	PackageableElement	0.30
Factory	PackageableElement	0.30
Package	Package	0.65
DataType	DataType	0.64
TypedElement	TypedElement	0.64
Operation	Operation	0.62
StructuralFeature	StructuralFeature	0.62
Classifier	Classifier	0.61
NamedElement	NamedElement	0.61
NamedElement	TypedElement	0.47
TypedElement	NamedElement	0.46
ModelElement	NamedElement	0.40
ModelElement	TypedElement	0.40

eter, *Datatype* and *Class* were found to be the same. These concepts can be merged easily as their definitions in the metamodels specifications are also the same. Figure 6.20 shows each of these concepts as a single concept in the merged metamodel.

The concept *Enum* in Ecore was matched to *PrimitiveType* and *Enumeration* in UML2CD (Table 6.20). The ontological analysis reveals that *Enum* is the same as *Enumeration* in UML2CD. *Enum* in Ecore was therefore merged with *Enumeration* in UML2CD (Figure 6.20). Both *Enum* in Ecore and *Enumeration* in UML2CD are specialised classes of a class *Datatype* in their respective metamodels (Figures 6.18 and 6.19), so this relationship is preserved in the merged metamodel (Figure 6.20).

Table 6.20: Candidate concepts between Ecore and UML2CD for interoperability

Ecore	**UML2CD**
Parameter	Parameter
Class	Class
Package	Package
DataType	DataType
TypedElement	TypedElement
Operation	Operation
StructuralFeature	StructuralFeature
Classifier	Classifier
NamedElement	NamedElement
ModelElement	NamedElement, TypedElement
Annotation	Association, Abstraction, Enumeration
Reference	Relationship, Constraint, Comment, Association
Factory	PackageableElement
EnumLiteral	PackageableElement, NameSpace, TypedElement
Attribute	Property
Enum	PrimitiveType, sEnumeration

The concept *Attribute* in Ecore was matched to *Property* in UML2CD (Table 6.20). Ontological analysis of both these concepts in their respective metamodels shows that they have the same meaning and can be merged easily. The merged metamodel (Figure 6.20) therefore shows them as a single concept *Property.* Both *Attribute* and *Property* were specialised classes of *StructuralFeature*, so the same relationship is maintained in the merged metamodel (Figure 6.20).

EnumLiteral in Ecore was matched to the *PackageableElement*, *NameSpace* and *TypedElement* in UML2CD (Table 6.20). The ontological analysis of *Enum* with all these concepts does not qualify it as same or similar to any of these concepts. Rather, a more suitable match is *EnumerationLiteral* in UML2CD, which is exactly the same ontologically as *EnumLiteral* in Ecore. Both concepts were therefore merged to the single concept (*EnumerationLiteral*) in the merged metamodel (Figure 6.20).

Object in Ecore and *Element* in UML2CD do not match syntactically or semantically, but their ontological analysis shows that these two concepts can be merged. Both are root concepts in their respective metamodels, having all other concepts as their specialised classes. Although *Object* in Ecore was meant to be the same as *Object* in the Java language (The superclass of every class), the purpose of *Object* in Ecore is the same as the purpose of *Element* in UML2CD. Figure 6.20 shows them as a single merged concept (*Element*).

Alignment

Some of the concepts in Table 6.20 cannot be merged straightforwardly. These concepts require *alignment* (mapping or bridging). *ModelElement* in Ecore was matched to *NamedElement* and *TypedElement* in UML2CD. On the other hand, both of these concepts have their exact matches in the other metamodel. Thus, matching these concepts with *ModelElement* in Ecore is not reasonable. We have to accommodate the *ModelElement* as it is (i.e. a mapping) to maintain the structure of the resulting metamodel. Figure 6.20 shows that *ModelElement* has been added as a subclass of *Element* in the merged metamodel, while *ModelElement* has specialised classes *Annotation* and *Factory* as in the original metamodel (Figure 6.18). In the merged metamodel, *ModelElement* also has a special class *NamedeElement*, which was a special class of *ModelElement* in Ecore (Figure 6.18) and a special class of *Element* in UML2CD (Figure 6.19). This arrangement makes *ModelElement* a bridge between *NamedElement* and *Element* in the merged metamodel (Figure 6.20).

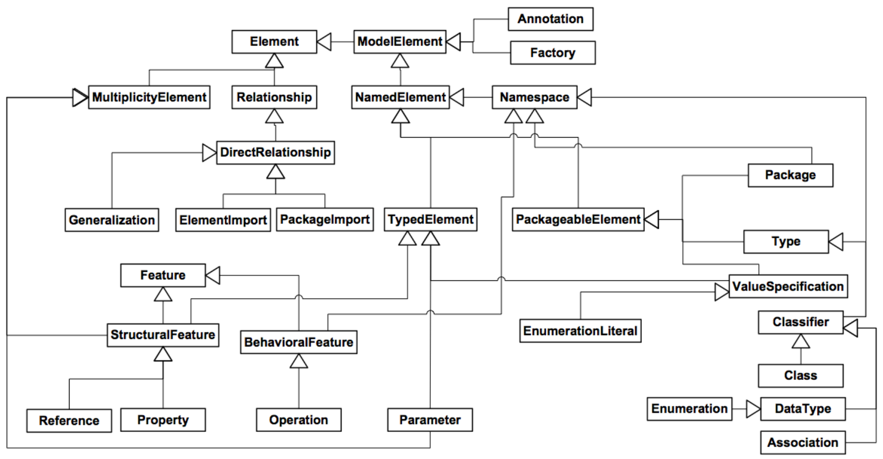

Figure 6.20: Merged metamodel for Ecore and UML2CD

The merged metamodel (Figure 6.20 accommodates all the concepts of Ecore and UML2CD. Now, models already created using either of these metamodels are conformant to the new merged metamodel. This interoperable metamodel also reduces the size of metamodels in terms of the number of classes and associations, which we will discuss in the next chapter in detail.

6.5 Example 5: SPEM and BPMN

This section shows the partial application of the interoperability framework to another pair of metamodels: BPMN and SPEM (OMG, 2009b). This example shows scenario in which interoperability between metamodels is required as the scenario cannot be modelled using one metamodel. We will explore the interoperability of SPEM and BPMN.

6.5.1 SPEM and BPMN

SPEM (Software and System Process Engineering Metamodel) is an OMG standard (OMG, 2009b). SPEM is used to define software and systems development processes and their components. It provides specific structures to enhance generic behaviour models that describe development processes. The SPEM metamodel is structured into seven main metamodel packages. The structure divides the model into logical units. Each unit extends the units it depends upon, providing additional structures and capabilities to the elements defined below it. Figure 6.21 depicts the partial metamodel for SPEM.

The details of BPMN are not repeated here as we have already discussed it in Section 6.1.1. The partial metamodel of BPMN used in this example is shown in Figure 6.22.

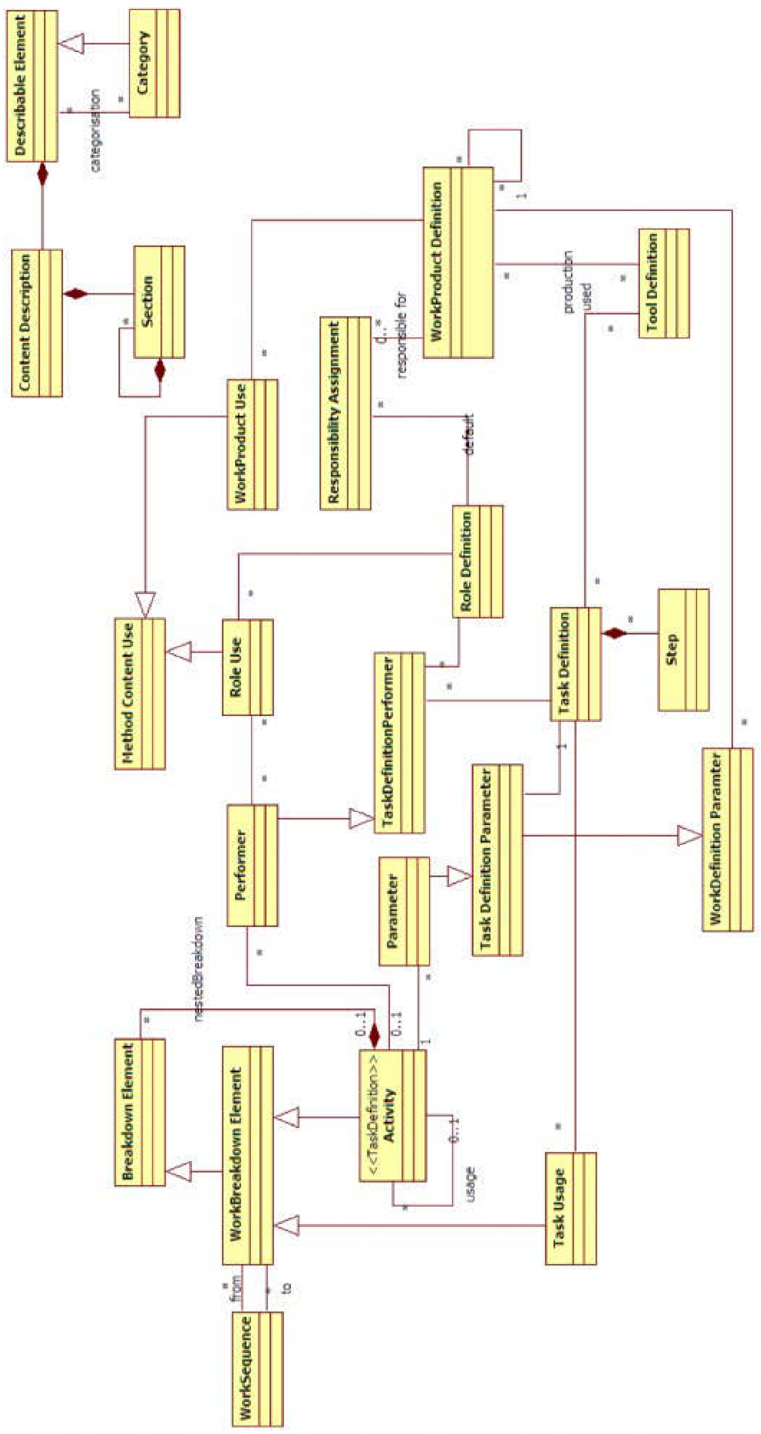

Figure 6.21: SPEM metamodel (partial)

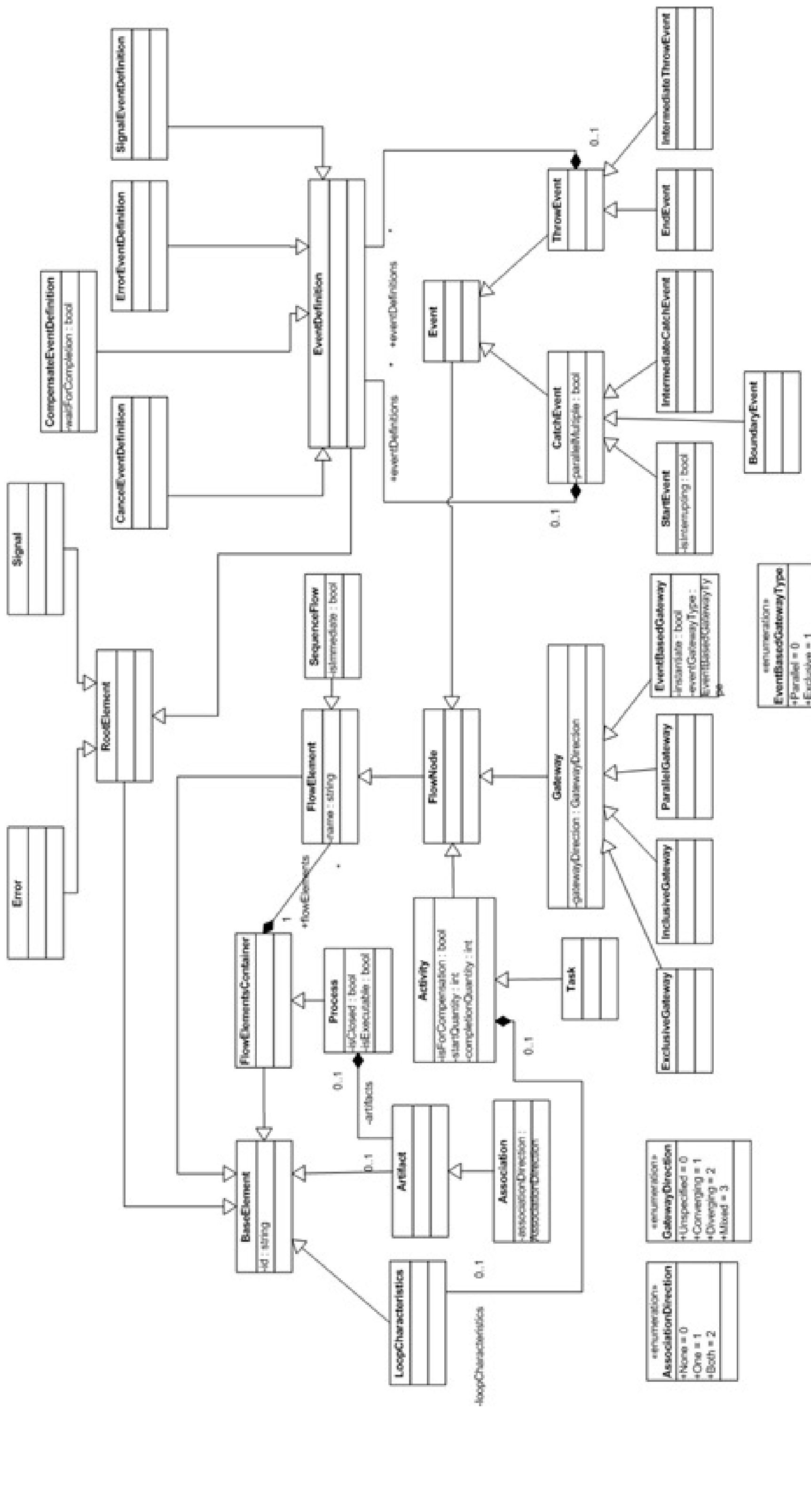

Figure 6.22: BPMN metamodel (partial)

6.5.2 The Scenario

Below is an example of an iterative software process model. This model consists of some sequential activities such as *Define Initial Requirements* and *Initial Planning*. The remaining activities are performed in iterative fashion, such as *Plan Iteration*, *Define Team*, *Build*, *Test* and *User Evaluation*. The activity *Product Release* is performed based on whether the product passes a *User Evaluation*. During an iteration, *Maintain Change Log* is initiated if any change request occurs. Each of these activities produces some products or artefacts. Some of the activities also maintain some associated metrics.

This example process model depicts the scenario in which both BPMN and SPEM are used to model a process and it was not possible to model the process using either metamodel alone. All the elements in Figure 6.23 are marked with the name of the originating metamodel to distinguish the elements of both metamodels. For example, *SPEM-Work Product Initial Requirements* depicts an instance of the *Work Product* class in SPEM. The activities can be modelled using the *Activity* class, available in both BPMN and SPEM. The artefacts

Table 6.21: Similarities between BPMN and SPEM

BPMN	**SPEM**	**SSM**	**Struct**	**Sem**	**Overall**
Activity	Activity	1	0	0.12	0.63
EventDefinition	ToolDefinition	0.64	0	0	0.38
EventDefinition	RoleDefinition	0.64	0	0	0.38
EventDefinition	TaskDefinition	0.64	0	0	0.38
ErrorEventDefinition	WorkProductDefinition	0.6	0	0.02	0.36
CancelEventDefinition	WorkProductDefinition	0.52	0	0.02	0.31
SignalEventDefinition	WorkProductDefinition	0.52	0	0.02	0.31

produced by these activities can be modelled using the *Work Product* class in SPEM or the *Artifact* class in BPMN. *Defining Team for Iteration* is the activity which produces the team structure for that iteration. This team structure can best be modelled using the *Team Profile* class of SPEM, as BPMN does not provide any class to accommodate this requirement, even though teams are commonly formed in business processes. Some of the activities calculate metrics, e.g. Errors/KLOC. These metrics can easily be modelled using the *Metric* class available in SPEM. BPMN does not have an equivalent class.

On the other hand, some activities, for example *Build* have to cater for any change in the requirements during the activity. In this case, the activity must trigger another activity, *Maintain Change Log*. Such a situation cannot be accommodated by SPEM and we have shown it in the example below by attaching different events with activities which are actually instances of the *Intermediate Event* class of BPMN. These events are marked with their BPMN name, e.g. *BPMN Event Change Request*.

Last but not least, the *Product Release* activity depends on the successful completion of the *User Acceptance* activity. This situation depicts the decision-making for which BPMN provides the *Gateway* class, which can be instantiated at the model level to accommodate different types of decisions. SPEM lack any such constructs.

As the purpose of this example is not to show the matching process in detail, Table 6.21 shows only those concepts of SPEM and BPMN that have more than 30% overall similarity.

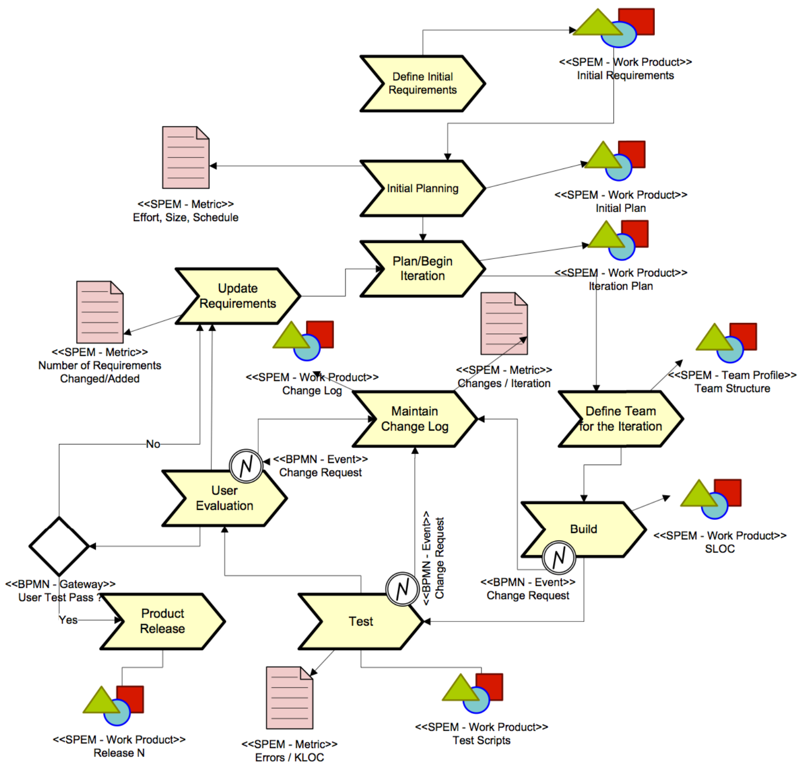
Define Initial Requirements
<<SPEM - Work Product>> Initial Requirements
Initial Planning
<<SPEM - Metric>> Effort, Size, Schedule
<<SPEM - Work Product>> Initial Plan
Plan/Begin Iteration
<<SPEM - Work Product>> Iteration Plan
Update Requirements
<<SPEM - Metric>> Number of Requirements Changed/Added
<<SPEM - Work Product>> Change Log
<<SPEM - Metric>> Changes / Iteration
<<SPEM - Team Profile>> Team Structure
Maintain Change Log
Define Team for the Iteration
No
<<BPMN - Event>> Change Request
User Evaluation
Build
<<SPEM - Work Product>> SLOC
<<BPMN - Event>> Change Request
<<BPMN - Event>> Change Request
<<BPMN - Gateway>> User Test Pass ?
Yes
Product Release
Test
<<SPEM - Work Product>> Release N
<<SPEM - Metric>> Errors / KLOC
<<SPEM - Work Product>> Test Scripts

Figure 6.23: An example scenario using the merged metamodel

So far, we have seen the application of the interoperability framework to some members of OMG's family of metamodels. These metamodels are generally based on object orientation, having classes and instances of classes. However, the interoperability framework is generic in nature and can be applied to any pair of metamodels.

6.6 Conclusion

In this chapter, we provide applications of our metamodel interoperability framework (presented in Chapter 5). These examples show the approach defended in this thesis for syntactic, semantic and structural interoperation of metamodels. We demonstrated the application of the framework with five different examples. In Section 6.1, we showed the application of the framework to BPMN and OSM, both proposed by the same organisation (OMG). In the next sections, we extended the application of the framework to some multi-agent systems (MAS) metamodels and demonstrated the merging of (ADELFE, Gaia and PASSI) and then (MaSE and Prometheus). The merging of the UML class diagram metamodel with ECore was presented in Section 6.4. Finally, to show the usefulness of the merged metamodels, Section 6.5 shows a small example using BPMN and SPEM where one metamodel alone was not sufficient.

In the next chapter, we will discuss some quality aspects of the results we have achieved from these examples.

Chapter 7

Quality Assessment

7.1 Introduction

This chapter assesses the quality of the interoperability framework and the results we have achieved using this framework. The objective of this assessment is to analyse the performance of the framework, in comparison with some other techniques using standard precision and recall measures. We will also analyse the precision of this framework in the context of the different metamodels to which it was applied.

The next section describes our quality assessment calculations, while the following section presents our complexity analysis. Section 7.4 concludes the chapter.

7.2 Quality Assessment

The main activity within the framework was *matching* to find the *same* or *similar* conceptual elements in a pair of metamodels. There-

fore, our major focus is to analyse the quality of the matching results. We have used the standard measures of *Precision* and *Recall*, from the field of information retrieval. These measures are based on the notions of true positives *(tp)*, false positives *(fp)* and false negatives *(fn)*. True positives *(tp)* are values which were correctly matched while false positives *(fp)* are those values which were incorrectly identified.

Precision measures the share of relevant matches among all the matches given by any technique. Recall gives the frequency of relevant matches compared with the set of relevant matches. F-measures use both precision and recall to overcome some over- or underestimation of the two measures. F-measure can be considered an average of precision and recall measures.

In an ideal scenario, the precision and recall should be the same and *precision = recall = 1*. However, they also depend on the context in which they are used. In our case, we were expecting high precision and recall as our framework uses *ontological analysis* to reduce the number of false negatives *(fn)*, which should allow high values of precision and recall.

The F-Measure is a statistical measure for the accuracy of test results. It is calculated as the harmonic mean of precision and recall. The F-measure provides a way of combining recall and precision into a single measure which falls between recall and precision. Recall and precision can have relative weights in the calculation of the F-measure, giving it the flexibility to be used for different applications. These measures

are defined in (Salton, 1987) as

$$Precision = \frac{|tp|}{|tp| + |fp|} \tag{7.1}$$

$$Recall = \frac{|tp|}{|tp| + |fn|} \tag{7.2}$$

$$F - Measure = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{7.3}$$

We applied the above measures to some pairs of metamodels (OSM/BPMN, GAIA/PASSI and Ecore/UML2CD), that we used in our examples. Table 7.1 shows the precision and recall for these metamodels when matched using old techniques that used only syntactic and structural matching based on Equations 5.1 and 5.3. These techniques are used in the domain of ontology merging and schema matching, e.g. (Maedche and Staab, 2001). On the other hand, Table 7.2 shows the values of precision, recall and F-measure using our technique. These values are based on the results we achieved during our examples after ontological analysis (Tables 6.5 and 6.20). We can see that there are higher precision and recall values for all three pairs of metamodels. Figure 7.1 shows the improvement in precision and recall values.

Table 7.1: Precision and recall of matching using old techniques

Pair of Metamodels	Precision	Recall	F-Measure
OSM and BPMN	0.090	0.090	0.090
Ecore and UML2CD	0.687	0.578	0.628
Ecore and EER	0.571	0.666	0.615

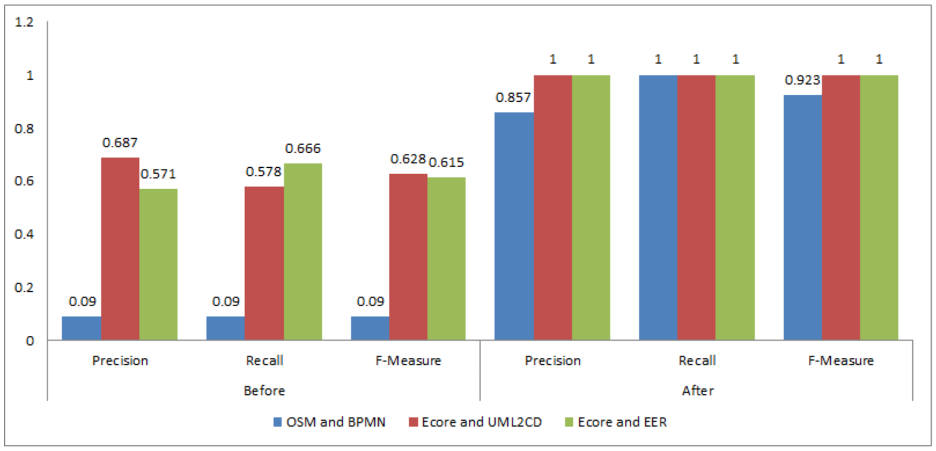

Figure 7.1: Improvements in precision

We can see from Tables 7.2 and 7.1 that there is a great increase in the precision and recall values for BPMN/OSM pair. These values were approximately 9% (0.09) (Table 7.1) using the old techniques and increased to almost 85% (0.85) (Table 7.2) using our approach. Both metamodels, although belonging to the same domain and developed by the same organization (OMG), are different in terms of conceptual elements. Tables 7.2 and 7.1 show the improvement (in terms of precision and recall) in matching results due to the application of our framework. Similarly, we have better precision and recall values for the other two pairs (GAIA/PASSI and Ecore/UML2CD) using our framework.

Table 7.2: Precision and recall of matching using the interoperability framework

Pair of Metamodels	Precision	Recall	F-Measure
OSM and BPMN	0.857	1	0.923
Ecore and UML2CD	1	1	1
Ecore and EER	1	1	1

These results also confirm our argument that to have better matching elements, we must analyse them ontologically, which is missing in most of the ontology merging and schema matching techniques discussed earlier. The next section elaborates some measures related to the size and complexity of these metamodels.

7.3 Size and Complexity Measures

This section presents the effects of interoperability of different metamodels on the size and complexity of the resulting metamodel. These results were calculated using the size and complexity measures proposed by (Rossi and Brinkkemper, 1996). The rationale for using these measure was twofold: first, they have already been used to measure the size and complexity of some process models (Siau and Cao, 2001) and (Recker et al., 2009a); second, they were also used to measure the size and complexity of some metamodels during the initial stage of our research (Henderson-Sellers et al., 2012). The purpose of this analysis was to discover whether the interoperability of metamodels makes any difference to the size or complexity of those metamodels.

Rossi and Brinkkemper (1996) presented a set of 17 metrics, based on the metamodels vocabulary (i.e. concepts and properties). Specifically, these are the aggregate measurement of object types, their attributes/properties, their relationships with each other and roles. Formally, a model M of a technique T is defined as $M = O, P, R, X, r, p$**, where:**

- O is a finite set of object types. An object is defined as a *thing* that exists independently. For example, *Class*, *Association* and *Classifier* are different object types in the software and systems process engineering metamodel (SPEM) (OMG, 2008a).

- P is a finite set of property types or attributes that are the characteristics associated with the object types.
- R is a finite set of relationship types. A relationship is an association between two or more object types. Different relationship cardinalities are treated differently. For example, *Composition 1 to 1* and *Composition 1 to ** are treated as two relationship types, as suggested by Rossi and Rossi and Brinkkemper (1996).
- X is a finite set of role types, a role being the name of the connection between an object type and its association.
- The variables r and p represent mappings from role types to relationship types and object types; and from nonproperty types to property types, respectively. Neither is used in our metrics calculations.

The complexity of a metamodel is calculated using Equation 7.4 :

$$C(M_T) = \sqrt{n(O_T)^2 + n(P_T)^2 + n(R_T)^2} \tag{7.4}$$

We applied these metrics to evaluate the size and complexity of some of the metamodels used in our examples (OSM, BPMN, GAIA, PASSI, Ecore and UML2CD). The figures representing these metamodels (Chapter 6) contain partial information and some properties and relationships may be missing, e.g. *Communication* of Gaia in Figure 6.10 shows no attributes but actually it has three attributes

according to the specification of Gaia. We have used only those objects, properties and relationships that are present in those figures for simplicity.

Table 7.3 shows the results of the calculation based on the partial metamodels as represented in Chapter 6. The last column of Table 7.3 shows the combined complexity when two metamodels are used together during software development. The third row in each section shows the complexity of the merged/interoperable metamodel. We can see the difference in both values for all three pairs of metamodels. The complexity of the merged metamodels is less than the combined complexity of both metamodels when used together.

A threat to the validity of results that we have achieved could be that these results cannot be compared with some generally accepted merged versions of these metamodels. Unfortunately, such a benchmark does not exist, except in the case of ADELFE, Gaia and PASSI metamodels, which were compared to the similar results achieved by (Bernon et al., 2004). This threat to the validity of the results could be minimized if the merged metamodels were reviewed by some expert in the field. However, a major obstacle in such a review was finding the expert who has expertise in all metamodels that are harmonized.

We can contrast the values of C(MT): the sum of the two metamodels (e.g. the OSM/BPMN value of 33.23) as compared with the value of 26 for Merged (BPMN and OSM). Similarly, the combined complexity of Ecore/UML2CD was calculated as 52.03 which is reduced to 39.01 (the last row) when these metamodels were merged.

Table 7.3: Size and complexity improvement

	n(OT)	n(PT)	n(RT)	C(MT)
OSM	11	0	6	12.52
BPMN (Partial)	19	8	2	20.71
Combined				33.23
Merged(BPMN and OSM)	24	8	6	26
GAIA	18	15	7	24.45
PASSI	18	12	6	22.44
Combined				46.89
Merged (GAIA and PASSI)	30	25		39.50
Ecore	17	0	1	17.02
UML2CD	35	0	1	35.01
Combined				52.03
Merged (Ecore and UML2CD)	39	0	1	39.01

Method engineers and analysts can use these merged metamodels to model systems that previously required both metamodels, e.g. the scenario explained in Section 6.5. This reduced joint complexity facilitates the user of metamodels as one (merged) metamodel rather than using two different metamodels at the same time.

7.4 Conclusion

We have shown that our technique for finding the interoperability of metamodels produces merged metamodels with considerably reduced size and complexity in comparison with the original metamodels.

This should make the merged metamodels easier for users to understand and apply, thereby improving productivity and efficiency in software development. Also, we have shown that our proposed framework results in better precision and recall as compare to old matching techniques.

This chapter marks the final stage of our research presentation. The following chapter presents the conclusions we have drawn from the overall project.

Chapter 8

Conclusions

If we knew what it was we were doing, it would not be called research, would it?

Albert Einstein

This chapter discusses some final considerations about the work presented in this thesis. In each of the previous chapters, we have already provided a chapter summary. However, in this chapter we focus on the major stages - *syntactic analysis*, *semantic analysis*, *structural analysis* - of the interoperability framework (presented in Chapter 5) and shed light on the lessons we have learned by applying the interoperability framework to different metamodels (Chapter 6). We treat them as possible opportunities and limitations during those stages. We, then, discuss our plans for future research in Section 8.2.

8.1 Lessons Learned

During the course of this research we found that the interoperability of a pair of metamodels may result in either *merging*, *mapping* or *bridging* (Section 4.1). We also showed that *matching* is the most fundamental and important operation during the interoperability of metamodels (Section 4.1.1). Matching a pair of elements requires their *syntactic*, *semantic* and *structural* analysis. The following sections summarise these stages with some discussion on the limitations of that stage and the opportunities we see.

8.1.1 Syntactic Matching

Syntactic matching between a pair of metamodels, used in the framework (Chapter 5), is based on a comparison between the names of the conceptual elements within those metamodels. The Edit Distance (ED) technique, among many other such techniques available in the literature, was used to calculate the syntactic similarity. ED counts the number of token insertions, deletions and substitutions needed to transform one lexical string S1 to another S2, viewing each string as a list of tokens. Some other techniques for string comparison are also available that can be used for such comparison, e.g. Ngram, morphological analysis (Stemming), and stopword elimination.

We have observed that these techniques are useful for comparing conceptual elements within the same domain, e.g. domain ontologies, where elements with the same name are likely to have the same meaning. The problem with applying these techniques to metamodels is that they are not effective when applied standalone. The results of this matching in our studies (Chapter 6) show that considering only syntactic similarity measures, isolated from their semantics, creates misunderstanding by expressing the same meanings in different terms. We faced this problem repeatedly during the comparison of metamodels (Chapter 6). Many conceptual classes appear to be the same in two different metamodels because of their names but have totally different semantics. For example, *Participant* in OSM and *Participant* in BPMN are syntactically the same but have quite different meanings. We recommend that it is not sufficient to evaluate being very careful while evaluating the similarity of metamodel elements based solely on their names.

8.1.2 Semantics Matching

Metamodels, as definitions of modelling languages, are designed (mostly) for specific domains. Consequently, most metamodels contain conceptual elements to define the language and elements to define the domain of interest. Therefore, we believe that to compare the semantic similarity of metamodel elements, it is important to consider both the linguistic and ontological (domain) perspectives. The linguistic semantics involves checking the semantics of the metamodel elements from that modelling language's perspective, e.g. their properties (attributes), types of attributes and to some extent their behaviour as well. On the other hand, ontological semantics means finding elements that have the same meaning but may have been presented with different names in different metamodels.

For linguistic semantics, techniques available for comparing class diagrams (Girschick, 2006) can be utilized to find the similarities be-

tween metamodel elements, especially for metamodels that are represented using object-oriented classes (metaclasses), e.g. OMG's family of metamodels. Ontological semantics can be compared using different approaches in computational linguistics and natural language processing, e.g. finding the synonyms of a given conceptual element of one metamodel and looking for those synonyms in the second metamodel. Synonyms can be found using any lexical database, e.g. a dictionary. We used WordNet (Miller et al., 1998) to find synonyms and word senses in our examples.

We have observed that finding ontological semantic similarity is very important as there are so many similar conceptual elements in metamodels which are presented with different names. For example, *Person* in OSM (OMG, 2009b) can be semantically matched with the *Human Performer* in BPMN (OMG, 2011a), even though they have low syntactic similarity. Beside synonyms, hyponyms (sub-name, e.g. sugar-maple and maple), hypernyms (super-name, e.g. step and footstep) can also be used to find semantically relevant elements, but none of these have previously been considered in any technique. Meronyms (part-name, e.g. ship and fleet) and holonyms (whole-name, e.g. face and eye) can also be useful to find semantic similarities. Another challenge in this regard is how to combine both linguistic and ontological semantic similarity for a pair of conceptual elements. Which one of them is more important and how much weight should be assigned to each of them is still unaddressed in the literature.

8.1.3 Comparing Structures

As we have discussed earlier, a metamodel is just another model. For a better similarity comparison between any pair of conceptual models, not only their syntax and semantics but also their structure should be compared. Different techniques have been proposed in the literature for analysing the structural similarity of conceptual models. Some of these (Dumas et al., 2009; Ehrig et al., 2007) compare the structure of business process models, whilst others (Voigt and Heinze, 2010a) match the structures of conceptual models based on graph theory.

The techniques used to compare the structure of business process models (e.g. (Dumas et al., 2009; Ehrig et al., 2007)) cannot be generalized to metamodels as business process models are behavioural models while metamodels represent structures. Similarly, converting conceptual models to graphs and then applying graph matching algorithms to find the structural similarity between them is not a trivial task. To apply such a graph matching technique, much care is required in converting a class diagram into a graph. True translation of relationships among classes (e.g. association, generalization, aggregation, composition) into relationships among nodes of a graph (e.g. directed/undirected, weighted/un-weighted) is not straightforward.

Another barrier to the application of such techniques is that most of the metamodels in the software engineering literature are specified as a combination of diagrams, tables and textual explanation. Creating a single class diagram for such a huge metamodel is not easy. Techniques based on planar graph theory like (Voigt and Heinze, 2010b) are also not feasible for metamodels because of the basic principle of

planar graphs (having no crossing edges). Metamodels with a rich set of constructs (classes) like UML can easily violate this rule as it is very difficult to convert class diagrams of these metamodels to graphs without any crossing edges. The complexity of these graph matching techniques (mostly of NP-hard complexity) is also mentioned by some authors (Dumas et al., 2009) as another barrier to their application to metamodels, hence making it difficult to apply in practice.

Alternatives to a graph matching technique are the schema matching techniques (Dumas et al., 2009), (Bernstein, 2003; Chukmol et al., 2005b; Lopes et al., 2006b; Sousa et al., 2009). We have used one such technique in our framework (Melnik et al., 2002). In this technique, the structural similarity of two conceptual elements C1 and C2 is calculated based on their structural neighbours: *ancestors*, *siblings*, *immediateChilds*, and *leaves*. These partial similarities are then calculated using mean values of all the neighbouring concepts (Equation 5.3). Another limitation of these techniques is that only the *Generalization* relationship is considered to determine the structural neighbours of conceptual elements. Most of the metamodels, especially metamodels based on object oriented classes, have other relationship types (e.g. Aggregation) that are important. These important relationship types could be a potential topic of research in future.

Based on the experience of applying these techniques to metamodels, we recommend that it is not necessary to compare the leaves of any conceptual element in a metamodel. Comparing the leaf classes of a given class (conceptual element) only results in low similarity. We also think that rather than comparing all the ancestors of a concep-

tual element, it is better to compare only the parent classes of that element.

Table 8.1 elaborates the difference that Equation 8.1 makes to the structural similarity of some metamodels. The table shows the number of concept pairs having a certain percentage of structural similarity between them. The percentage of structural similarity varies among different metamodels, e.g. the number of BPMN and OSM concepts having more than 5% structural similarity before (Using Equation 5.3) was 87, which was increased to 144 pairs of concepts after using Equation 8.1. Figure 8.1 shows the relative improvement in terms of finding more concepts with structural similarities.

An important analysis could be to analyse the effect of this changed equation on the values of overall *precision* and *recall* of the results achieved. Although this analysis is not covered in this thesis it could be a valuable contribution for future research in the same direction.

$$\begin{aligned} structSim(C_i, C_j) = parentSim(C_i, C_j) * parentCoef + \\ sibSim(C_i, C_j) * sibCoef + \\ immChSim(C_i, C_j) * immChCoef \end{aligned} \tag{8.1}$$

8.1.4 Automation

Considering the size and complexity of metamodels (Henderson-Sellers et al., 2012), it is very convenient to have tool support for determining the similarity of metamodels. However, our experience with the matching of metamodels shows that, although partial tool support is

very valuable, complete automated metamodel matching is not possible.

Automation of syntactic matching of metamodels elements can be achieved by implementing ED (Edit Distance) and SSM (Syntactic Similarity Measure) algorithms using available online calculators for ED (Garcia, 2013) and APIs (Mindrot, 2013). The ontological semantics of metamodel elements can be matched automatically using lexical databases like WordNet or MS Office Thesaurus.

Table 8.1: Effect of Equation 8.1 on structural similarity

Metamodel	**Structural Similarity**	**Before**	**After**
OSM and BPMN	>= 5%	87	144
BPMN and SPEM	>= 9%	31	829
Ecore and UML2CD	>= 10%	72	120
Mase and Prometheus	>= 3%	7	9
Ecore and EER	>= 5%	10	38

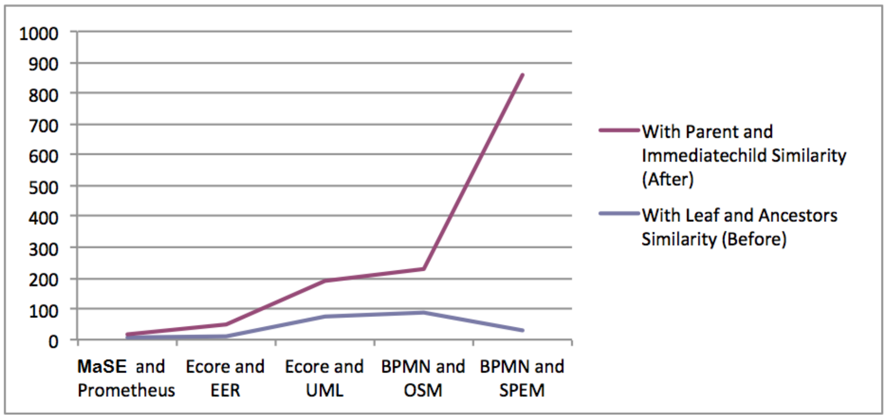

Figure 8.1: Number of concepts having structural similarity of some degree before using Equation 5.3 and after

Complete automation for metamodel similarity matching, especially for structural similarity, requires well-formed formal definitions of metamodels that can be used as an input to the automated tool. Unfortunately, besides XML definitions for some metamodels (OMG metamodels with XMI definitions), most metamodels lack formal specifications and are mostly specified using a combination of textual descriptions, tables and class diagrams.

Another important barrier to complete automation is that coefficients in Equations 5.3 and 5.5) do not have any fixed values and a domain expert must assign values for each matching. Also, the ontological semantic similarity analysis requires the expert's intellectual input to decide whether two conceptual elements are equal or not.

8.2 Future Research Plans

In this thesis, we have presented a framework (Chapter 5), that we have developed (based on ontology merging and schema matching techniques) for metamodel interoperability, and its application in some examples (Chapter 6). However, there are many other aspects that were not explored in this work but shall be pursued in future investigations. Some of the issues we have identified during our research require a considerable amount of effort to resolve, while others may be resolved by short-term research. Incorporating some other string-matching techniques or combination of these techniques, e.g. N-gram, morphological analysis (stemming), and stop-word elimination – as we discussed in Section 8.1.1 – can result in better syntactic matching. The application of these techniques in the framework needs a systematic analysis of its application and evaluation of the results.

We have used *synSets* (synonyms) of conceptual elements to evaluate their (linguistic) semantics. An interesting approach to pursue in this topic would be to incorporate different word senses, e.g. *hyponyms*, *hypernyms*, *meronyms* and *holonyms*. This is especially important from the perspective of *aggregation* and *generalization* relationships. Most of the conceptual elements in metamodels are bound to each other in such relationships, so that analysing the hyponyms and hypernyms may help to evaluate the similarity of conceptual elements in a *generalization* relationship. Similarly, analysis of meronyms and holonyms may help to evaluate the similarity of the concepts in *aggregation* relationship.

Structural similarity (Section 5.2.3) of conceptual elements was calculated using their structural neighbours. We observed during our research that immense effort is required to find structural neighbours of these elements if metamodels are not specified formally. Finding structural similarity of conceptual elements in a pair of metamodels can be greatly facilitated if we have the formal specifications of the metamodels. Developing some form of formal specifications of metamodels (e.g. XML definitions) could be a useful research topic in this context.

During the application of the framework, we observed that most metamodels have two orthogonal forms of conceptual elements: linguistic and ontological (as also highlighted by (Atkinson and Kühne, 2003)). The former represent the language definition while the latter

describe what concepts exist in a certain domain and what properties they have. These two types of elements are mingled with each other in most metamodels and there is no explicit boundary between them. An important consideration in metamodel matching is to separate these two types of elements; we call it *cleansing*. Metamodels must be first *cleansed* before matching can occur. This cleansing is required to remove the conceptual elements in metamodels that are not related to the domain of interest. Most such elements are linguistic (i.e. glue) and are present to maintain the structure of the metamodels. For example, *Resource Parameter Binding* and *ParallelGateway* in BPMN are related to the language definition of BPMN and are not worth matching with any other metamodel of the same domain since every metamodel has its own language definitional elements. Rather, it is better to match the conceptual elements that are related to the domain of interest, e.g. matching *Activity* in BPMN with *Activity* in SPEM, which are more related to the common domain of interest: Workflows and Processes.

Another contribution in the future could be to find good values for the coefficients used in Equations 5.3 and 5.5. This task is not simple and requires much empirical evidence for the analysis. One of our future research plans is recommend values after applying the framework to several metamodels and critically assessing the similarity results.

Last but not least, our future research plan includes the application of the framework to more pairs of metamodels and also to develop a tool that can be used to identify the similarities among the conceptual elements of metamodels.

The research presented in this thesis is not a landmark in the field of metamodel interoperability, but has set a basis that can be used for the future research in the same direction: metamodel interoperability.

References

Alonso, J. B. (2006). Ontology-based software engineering. Technical report, European Integrated Project IST-027819.

Aßmann, U., Zschaler, S., and Wagner, G. (2006). *Ontologies for Software Engineering and Software Technology*, chapter Ontologies, Meta-models, and the Model-Driven Paradigm, pages 249–273. Springer Berlin.

Atkinson, C. (1998). Supporting and applying the uml conceptual framework. In Bézivin, J. and Muller, P.-A., editors, *UML*, volume 1618 of *Lecture Notes in Computer Science*, pages 21–36. Springer.

Atkinson, C. and Kühne, T. (2002). Profiles in a strict metamodeling framework. *Sci. Comput. Program.*, 44(1):5–22.

Atkinson, C. and Kühne, T. (2002). Rearchitecting the uml infrastructure. *ACM Trans. Model. Comput. Simul.*, 12(4):290–321.

Atkinson, C. and Kühne, T. (2003). Model-driven development: A metamodeling foundation. *IEEE Softw.*, 20(5):36–41.

Atkinson, C. and Kuhne, T. (2005). Concepts for comparing modeling tool architectures. In *MODELS 2005*, volume LNCS 3713, pages 398–413. Springer-Verlag, Berlin.

Baclawski, K., Kokar, M. M., Kogut, P. A., Hart, L., Smith, J. E., Letkowski, J., and Emery, P. (2002). Extending the unified modeling language for ontology development. *Software and System Modeling*, 1(2):142–156.

Bernon, C., Cossentino, M., Gleizes, M. P., Turci, P., and Zambonelli, F. (2004). A study of some multi-agent meta-models. In Odell, J., Giorgini, P., and Müller, J. P., editors, *AOSE*, volume 3382 of *Lecture Notes in Computer Science*, pages 62–77. Springer.

Bernon, C., Gleizes, M.-P., Peyruqueou, S., and Picard, G. (2003). Adelfe: a methodology for adaptive multi-agent systems engineering. In *Proceedings of the 3rd International Conference on Engineering Societies in the Agents World III*, ESAW'02, pages 156–169, Berlin, Heidelberg. Springer-Verlag.

Bernstein, P. A. (2003). Applying model management to classical meta data problems. In *CIDR*, pages 17–32.

Bertrand, T. and Bézivin, J. (2000). *Ontological support for business process improvement*, pages 313–331. Artech House, London.

Bettin, J. and Clark, T. (2010). Advanced modelling made simple with the gmodel metalanguage. In *Proceedings of the First International Workshop on Model-Driven Interoperability*, MDI '10, pages 79–88. ACM.

Beydoun, G., Low, G. C., Henderson-Sellers, B., Mouratidis, H., Gómez-Sanz, J. J., Pavón, J., and Gonzalez-Perez, C. (2009). Faml: A generic metamodel for mas development. *IEEE Trans. Software Eng.*, 35(6):841–863.

Bézivin, J. (2004). In Search of a Basic Principle for Model Driven Engineering. *UPGRADE – The European Journal for the Informatics Professional*, 5(2):21–24.

Bézivin, J. (2005). On the unification power of models. *Software and Systems Modelling*, 4:171–188.

Bézivin, J., Devedzic, V., Djuric, D., Favreau, J.-M., Gasevic, D., and Jouault, F. (2006). *An M3-Neutral Infrastructure for Bridging Model Engineering and Ontology Engineering*, pages 159–171. Springer London.

Bézivin, J. and Gerbe, O. (2001). Towards a precise definition of the omg/mda framework. 872565 273.

Bézivin, J., Soley, R. M., and Vallecillo, A. (2010). Proceedings of the first international workshop on model-driven interoperability.

Blanc, X., Marie-Pierre, G., and Prawee, S. (2005). Model bus: towards the interoperability of modelling tools. In Aßmann, U., Aksit, M., and Rensink, A., editors, *Model Driven Architecture*, volume 3599 of *Lecture Notes in Computer Science*, pages 17–32. Springer Berlin Heidelberg.

Booth, T. (1967). *Sequential Machines and Automata Theory.* John Wiley and Sons New York.

Brambilla, M., Cabot, J., and Wimmer, M. (2012). *Model-Driven Software Engineering in Practice: Synthesis Lectures on Software Engineering.* 1. Morgan and Claypool.

Bresciani, P., Giorgini, P., Henderson-Sellers, B., Low, G., Winikoff, M., Guizzardi, G., and Wagner, G. (2005). *Towards Ontological Foundations for Agent Modelling Concepts Using the Unified Fundational Ontology (UFO)*, volume 3508 of *Lecture Notes in Computer Science*, pages 410–410. Springer Berlin / Heidelberg.

Bruijn, J. d., Ehrig, M., Feier, C., Maríns-Recuerda, F., Scharffe, F., and

Weiten, M. (2006). Ontology mediation, merging, and aligning. *Semantic Web Technologies*.

Brunet, G., Chechik, M., Easterbrook, S., Nejati, S., Niu, N., and Sabetzadeh, M. (2006). A manifesto for model merging. In *Proceedings of the 2006 international workshop on Global integrated model management*, GaMMa '06, pages 5–12, New York, NY, USA. ACM.

Bunge, M. (1977). Ontology i: The furniture of the world. *Treatise on Basic Philosophy*, 3.

Caire, G., Coulier, W., Garijo, F. J., Gómez-Sanz, J. J., Pavón, J., Leal, F., Chainho, P., Kearney, P. E., Stark, J., Evans, R., and Massonet, P. (2001). Agent oriented analysis using message/uml. In Wooldridge, M., Weiß, G., and Ciancarini, P., editors, *AOSE*, volume 2222 of *Lecture Notes in Computer Science*, pages 119–135. Springer.

Calero, C., Ruiz, F., and Piattini, M. (2006). *Ontologies for software engineering and software technology*. Springer.

Castro, J., Kolp, M., and Mylopoulos, J. (2001). A requirements-driven development methodology. In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering*, CAiSE '01, pages 108–123, London, UK, UK. Springer-Verlag.

Cernuzzi, L. and Zambonelli, F. (2011). Improving comparative analysis for the evaluation of aose methodologies. *Int. J. Agent-Oriented Softw. Eng.*, 4(4):331–352.

Chan, P. P.-S. (1976). The entity-relationship model–toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36.

Chomsky, N. (2000). *New horizons in the study of language and mind.* Cambridge University Press.

Chukmol, U., Rifaieh, R., and Benharkat, A.-N. (2005a). Exsmal: Edi/xml semi-automatic schema matching algorithm. In *CEC*, pages 422–425. IEEE Computer Society.

Chukmol, U., Rifaieh, R., and Benharkat, N. (2005b). Exsmal: Edi/xml semi-automatic schema matching algorithm. In *E-Commerce Technology, 2005. CEC 2005. Seventh IEEE International Conference on*, pages 422–425.

Cranefield, S. (2001). Uml and the semantic web. In Cruz, I. F., Decker, S., Euzenat, J., and McGuinness, D. L., editors, *SWWS*, pages 113–130.

Dam, H. K. and Winikoff, M. (2012). Towards a next-generation aose methodology. *Science of Computer Programming.*

de Sousa Jr, J., Lopes, D., Claro, D. B., and Abdelouahab, Z. (2009). *A Step Forward in Semi-automatic Metamodel Matching: Algorithms and Tool*, volume 24 of *Lecture Notes in Business Information Processing*, pages 137–148. Springer Berlin Heidelberg.

Decker, B., Eric, R., Rech, J., Klein, B., and Hoecht, C. (2005). Self-organized reuse of software engineering knowledge supported by semantic wikis. In *Workshop on Semantic Web Enabled Software Engineering (SWESE.*

Devedzić, V. (2002). Understanding ontological engineering. *Communications of the ACM*, 45(4):136–144.

Dijkman, R., Dumas, M., Dongen, B. v., Krik, R., and Mendling, J. (2011). Similarity of business process models: Metrics and evaluation. *Information Systems.*, 36(2):498–516. 1891237.

Dillon, T. S., Chang, E., and Wongthongtham, P. (2008). Ontology-based software engineering- software engineering 2.0. In *Australian Software Engineering Conference*, pages 13–23. IEEE Computer Society.

Dumas, M., Garcia-Banuelos, L., and Dijkman, R. (2009). Similarity search of business process models. *IEEE Data Eng. Bull*, 32(3):23–28.

ECIS (2011). Interoperability in model-driven development. Technical report, European Commission for Information Society.

Ehrig, M., Koschmider, A., and Oberweis, A. (2007). Measuring similarity between semantic business process models. In Roddick, J. F. and Hinze, A., editors, *APCCM*, volume 67 of *CRPIT*, pages 71–80. Australian Computer Society.

Ehrig, M. and Staab, S. (2004). Qom - quick ontology mapping. In McIlraith, S., Plexousakis, D., and Harmelen, F., editors, *The Semantic Web ISWC 2004*, volume 3298 of *Lecture Notes in Computer Science*, pages 683–697. Springer Berlin Heidelberg.

Ehrig, M. and Sure, Y. (2004). Ontology mapping: An integrated approach. In Bussler, C., Davies, J., Fensel, D., and Studer, R., editors, *The Semantic Web: Research and Applications*, volume 3053 of *Lecture Notes in Computer Science*, pages 76–91. Springer Berlin Heidelberg.

EIA (1994). Interchange format-overview. interim standard.

Eriksson, H.-E. and Penker, M. (1998). *UML Toolkit*. John Wiley and Sons, New York.

Eriksson, O. and Ågerfalk, P. J. (2010). Rethinking the meaning of identifiers in information infrastructures. *AIS*, 11(8):433–454.

Falbo, R. d. A., Natali, A. C. C., Mian, P. G., Bertollo, G., and Ruy, F. B. (2003). Ode: Ontology-based software development environment. In *CACIC 2003*.

Favre, J.-M. (2004). Foundations of model (driven) (reverse) engineering : Models - episode 1: Stories of the fidus papyrus and of the solarus. In Bézivin, J. and Heckel, R., editors, *Language Engineering for Model-Driven Software Development*, volume 04101 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

Fowler, M. and Scott, K. (2000). *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley.

Garcia, E. (2013). Mining the web through information retrieval.

Garcia-Ojeda, J. C., DeLoach, S. A., Robby, R., Oyenan, W. H., and Valenzuela, J. (2008). O-mase: a customizable approach to developing multiagent development processes. In *Proceedings of the 8th International Conference on Agent-Oriented Software Engineering VIII*, AOSE'07, pages 1–15, Berlin, Heidelberg. Springer-Verlag.

Garo, A. and Turci, P. (2003). Metamodel sources:gaia. Technical report, Foundations for intelligent physical agents.

Gašević, D., Kaviani, N., and Hatala, M. (2007). On metamodeling in megamodels. In Engels, G., Opdyke, B., Schmidt, D. C., and Weil, F., editors, *MoDELS*, volume 4735 of *Lecture Notes in Computer Science*, pages 91–105. Springer.

Gehlert, A. and Pfeiffer, D. (2005). A framework for comparing conceptual models. In Desel, J. and Frank, U., editors, *EMISA*, volume 75 of *LNI*, pages 108–122. GI.

Gilbreth, F. B. and Gilbreth, L. M. (1921). Process charts. *American Society of Mechanical Engineers*.

Girschick, M. (2006). Difference detection and visualization in uml class diagrams. technical report. Technical report, Technical University of Darmstadt.

González-Pérez, C. and Henderson-Sellers, B. (2006). *An ontology for software development methodologies and endeavours*, pages 123–152. Springer-Verlag, Berlin.

Gonzalez-Perez, C. and Henderson-Sellers, B. (2007). Modelling software development methodologies: A conceptual foundation. *Journal of Systems and Software*, 80(11):1778 – 1796.

González-Pérez, C. and Henderson-Sellers, B. (2009). *Metamodelling for Software Engineering*. John Wiley and Sons.

Greenfield, J. and Short, K. (2004). *Software Factories*. John Wiley and Sons.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220.

Guizzardi, G. and Halpin, T. (2008). Ontological foundations for conceptual modelling. *Appl. Ontol.*, 3(1-2):1–12.

Happel, H.-J. and Seedorf, S. (2006). Applications of ontologies in software engineering. In *2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006)*.

Haythorn, W. (1994). What is object oriented design? *Object Oriented Programming*, 7:67–78.

HeÃŸ, A. (2006). *An Iterative Algorithm for Ontology Mapping Capable of Using Training Data*, volume 4011 of *Lecture Notes in Computer Science*, pages 19–33. Springer Berlin Heidelberg.

Heiler, S. (1995). Semantic interoperability. *ACM Comput. Surv.*, 27(2):271–273. 210392.

Henderson-Sellers, B. (1994). Comma: an architecture for method interoperability. *Object Analysis and Design*, 1(3):25–28.

Henderson-Sellers, B. (2011). Bridging metamodels and ontologies in software engineering. *J. Syst. Softw.*, 84(2):301–313.

Henderson-Sellers, B. (2012). *On the Mathematics of Modelling, Metamodelling, Ontologies and Modelling Languages.* Briefs in Computer Science. Springer.

Henderson-Sellers, B. and González-Pérez, C. (2004). A comparison of four process metamodels and the creation of a new generic standard. *Information and Software Technology*, 47(1):49–65.

Henderson-Sellers, B. and González-Pérez, C. (2005). Connecting powertypes and stereotypes. *Journal of Object Technology*, 4(7):83–96.

Henderson-Sellers, B., Qureshi, M. A., and González-Pérez, C. (2012). Towards an interoperable metamodel suite: Size assessment as one input. *International Journal of Software and Informatics*, 6(2):111–124.

Hesse, W. (2006). More matters on (meta-)modeling: Remarks on thomas kühne's "matters". *Software and Systems Modeling*, 5(4):387–394.

Hevner, A. and Chatterjee, S. (2010). *Design Research in Information Systems.* Integrated Series in Information Systems. Springer.

Hitzler, P., Krotzch, M., Ehrig, M., and Sure, Y. (2005). What is ontology merging? a category-theoratical perspective using pushouts. In *First International Workshop on Contexts and Ontologies: Theory, Practice and Applications*, pages 104–107.

InterOP (2004). State of the art report: Ontology interoperability. Technical report, NoE WP8, Subtask 3.

ISO (1998). Cdif framework.

ISO (2011). Iso/tc 37/sc 3 systems to manage terminology, knowledge and content.

ISO/IEC-24744 (2007). Software engineering metamodel for development methodologies.

ISO/IEC19505 (2012). Information technology - object management group ling language (omg uml) – part 2: Superstructure.

Jean, S., Aït-Ameur, Y., and Pierra, G. (2011). A language for ontology-based metamodeling systems. In Catania, B., Ivanović, M., and Thalheim, B., editors, *Advances in Databases and Information Systems*, volume 6295 of *Lecture Notes in Computer Science*, pages 247–261. Springer Berlin Heidelberg.

Jiang, K., Zhang, L., and Miyake, S. (2007). An executable uml with ocl-based action semantics language. In *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*, pages 302–309.

Johnson, R. and Woolf, B. (1997). *The type object pattern*, volume 3, pages 47–65. Addison-Wesley, Boston.

Kappel, G., Kargl, H., Kramler, G., Schauerhuber, A., Seidl, M., Strommer, M., and Wimmer, M. (2006). Matching metamodels with semantic systems

- an experience report. In *Workshop on Model Management und Metadaten-Verwaltung*.

Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., and Wimmer, M. (2006). Lifting metamodels to ontologies: a step to the semantic integration of modeling languages. In *ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems*.

Kim, W., Park, S., Bang, S., and Lee, S. (2006). An ontology mapping algorithm between heterogeneous product classification taxonomies. In Shvaiko, P., Euzenat, J., Noy, N. F., Stuckenschmidt, H., Benjamins, V. R., and Uschold, M., editors, *Ontology Matching*, volume 225 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Knublauch, H. (2005). Ramblings on agile methodologies and ontology-driven software development. In *'Workshop on Semantic Web Enabled Software Engineering (SWESE)*.

Koch, N. and Kozuruba, S. (2012). Requirements models as first class entities in model-driven web engineering. In Grossniklaus, M. and Wimmer, M., editors, *Current Trends in Web Engineering*, volume 7703 of *Lecture Notes in Computer Science*, pages 158–169. Springer Berlin Heidelberg.

Kolovos, D. S., Paige, R. F., and Polack, F. A. (2006). Merging models with the epsilon merging language. In *MODELS 2006*, LNCS 4199, pages 215–229. Springer-Verlag Berlin Heidellberg.

Kosanke, K. (2006). Iso standards for interoperability: a comparison. In Konstantas, D., Bourrières, J.-P., Léonard, M., and Boudjlida, N., editors, *Interoperability of Enterprise Software and Applications*, pages 55–64. Springer London.

Kühn, H. and Murzek, M. (2005). Interoperability issues in metamodelling platforms. In *1st Int. Conf. on Interoperability of Enterprise Software and Applications (INTEROP-ESA'05)*.

Kühne, T. (2005). *What is a Model?* Internat. Begegnungs-und Forschungszentrum für Informatik.

Kühne, T. (2006a). Clarifying matters of (meta-) modeling: an author's reply. *Software and Systems Modelling*, 5(4):395–401.

Kühne, T. (2006b). Matters of metamodelling. *Software and System Modeling*, 5(4):395–401.

Lano, K. and Haughton, H. (1996). *Specification in B: An Introduction Using the B Toolkit*. WSCGM.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710.

Lin, J., Fox, M., and Bilgic, T. (1996). A requirement ontology for engineering design. *Concurrent Engineering: Research and Applications*, 4:279–291.

Lopes, D., Hammoudi, S., and Abdelouahab, Z. (2006a). Schema matching in the context of model driven engineering: From theory to practice. In Sobh, T. and Elleithy, K., editors, *Advances in Systems, Computing Sciences and Software Engineering*, pages 219–227. Springer Netherlands.

Lopes, D., Hammoudi, S., de Souza, J., and Bontempo, A. (2006b). Metamodel matching: Experiments and comparison. In *Software Engineering Advances, International Conference on*, pages 2–2.

Ludewig, J. (2003). Models in software engineering - an introduction. *Software and Systems Modelling*, 2(1):5–14.

Maedche, A., Motik, B., Silva, N., and Volz, R. (2002). Mafra - a mapping framework for distributed ontologies. In Gómez-Pérez, A. and Benjamins, V. R., editors, *EKAW*, volume 2473 of *Lecture Notes in Computer Science*, pages 235–250. Springer.

Maedche, A. and Staab, S. (2001). Comparing ontologies - similarity measures and a comparison study. Technical report, Institute AIFB, University of Karlsruhe, Internal Report.

Magnini, B. and Speranza, M. (2002). Merging global and specialized linguistic ontologies. In *Workshop Ontolex-2002 Ontologies and Lexical Knowledge Bases, LREC-2002*.

Melnik, S., Garcia-Molina, H., and Rahm, E. (2002). Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In Agrawal, R. and Dittrich, K. R., editors, *ICDE*, pages 117–128. IEEE Computer Society.

Meyer, B. (1988). *The Object-Oriented Software Construction*. Prentice Hall.

Miller, G. (1995). Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41.

Miller, G., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. (1998). *Five Papers on WordNet*. MIT Press.

Mindrot (2013). Python module to calculate edit distance (py-editist).

Mohsenzadah, M., Shams, F., and Teshnehlab, M. (2005). A new approach for merging ontologies. *Worlds Academy of science and technology*, 1(4):153–159.

Monarchi, D., Booch, G., Henderson-Sellers, B., Jacobson, I., Mellor, S., Rumbaugh, J., and Wirfs-Brock, R. (1994). Methodology standards: help or hindrance? *SIGPLAN Not.*, 29(10):223–228. 191115 Chairman - David Monarchi.

Morris, C. (1938). Foundations of a theory of signs. In *International Encyclopedia of Unified Science*, pages 77–138. Chicago University Press.

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., and Swartout, W. R. (1991). Enabling technology for knowledge sharing. *AI Mag.*, 12(3):36–56. 123775.

Neuhaus, M. and Bunke, H. (2004). H.: An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification. In *10th Int. Workshop on Structural and Syntactic Pattern Recognition. LNCS 3138*, pages 180–189. Springer.

Noy, N. F. and Musen, M. A. (1999). An algorithm for merging and aligning ontologies: Automation and tool support. In *Sixteenth National Conference on Artificial Intelligence (AAAI-99)*. AAAI Press.

Noy, N. F. and Musen, M. A. (2003). The prompt suite: interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 55(6):983–1024.

Odell, J. (1994). Power types. *Object Oriented Programming*, 7(2):8–12.

Oliveira, K., Villela, K., Rocha, A., Travassos, G., Calero, C., Ruiz, F., and Piattini, M. (2006). *Use of Ontologies in Software Development Environments Ontologies for Software Engineering and Software Technology*, pages 275–309. Springer Berlin Heidelberg.

OMG (1997). Uml semantics. Technical report, OMG.

OMG (2001). Model driven architecture (mda),.

OMG (2005). Unified modeling language specifications, ver 2.0, available at http://www.omg.org/spec/uml/ 2.0.

OMG (2008a). Software and systems process engineering meta-model specification, ver 2.0, available at http://www.omg.org/spec/spem/current.

OMG (2008b). Software metrics metamodel specification, ver 1.0, available at http://www.omg.org/spec/smm/ 1.0.

OMG (2009a). Ontology definition metamodel specification, ver 1.0, available at http://www.omg.org/spec/ odm/current.

OMG (2009b). Organization structure metamodel specification, 3rd initial submission, available at http://www.omg.org/spec/index.htm.

OMG (2011a). Business process model and notation specification, ver 2.0, available at http://www.omg.org/ spec/bpmn/2.0.

OMG (2011b). Meta object facility (mof) core, ver 2.4, available at http://www.omg.org/spec/mof/2.4.

OMG (2012). System modelling language specification, ver 1.3, available at http://www.omg.org/spec/ sysml/1.3/. Online.

Ouksel, A. M. and Sheth, A. P. (1999). Semantic interoperability in global information systems: A brief introduction to the research area and the special section. *SIGMOD Record*, 28(1):5–12.

Oxford (2012). *Oxford Online Dictionary*. Oxford.

Pardo, C., Pino, F. J., García, F., Piattini, M., and Baldassarre, M. T. (2010). A systematic review on the harmonization of reference models. In *ENASE 2010*, pages 40–47. SciTe Press.

Pidcock, W. (2003). What are the differences between a vocabulary, a taxonomy, a thesaurus, an ontology, and a meta- model? Technical report, available at infogrid.org/trac/wiki/Reference/PidcockArticle; originally posted at www.metamodel.com.

Pirotte, A., Zimanyi, E., Massart, D., and Yakusheva, T. (1994). Materialization: a powerful and ubiquitous abstraction pattern. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB 94*. Morgan Kaufmann.

Pisanelli, D. M., Gangemi, A., and Steve, G. (2002). Ontologies and information systems: the marriage of the century. In *Proceedings of LYEE Workshop*.

Pottinger, R. and Bernstein, P. A. (2003). Merging models based on given correspondences. In *29th VLDB Conference*.

Rahm, E. and Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350. 767154.

Rech, J. and Althoff, K.-D. (2004). Artificial intelligence and software engineering: Status and future trends. *Special Issue on Artificial Intelligence and Software Engineering*, 18(3):5–11.

Recker, J., Muehlen, M. Z., Siau, K., Erickson, J., and Indulska, M. (2009a). Measuring method complexity: Uml versus bpmn. In Nickerson, R. C. and Sharda, R., editors, *AMCIS*, page 541. Association for Information Systems.

Recker, J. C., Zur Muehlen, M., Siau, K., Erickson, J., and Indulska, M. (2009b). Measuring method complexity : Uml versus bpmn.

Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI*, pages 448–453. Morgan Kaufmann.

Rossi, M. and Brinkkemper, S. (1996). Complexity metrics for systems development methods and techniques. *Information Systems*, 21(2):209–227.

Ruiz, F. and Hilera, J. R. (2006). *Using Ontologies in Software Engineering and Technology*, chapter 2, pages 49–95. Springer-Verlag, Berlin.

Saldhana, J. A. and Shatz, S. M. (2000). Uml diagrams to object petri net models: An approach for modeling and analysis. In *Int. Conf. on Software Engineering and Knowledge Engineering (SEKE 2000)*, pages 103–110.

Salton, G. (1987). Expert systems and information retrieval. *SIGIR Forum*, 21(3-4):3–9.

Scharffe, F. and Bruijn, J. d. (2005). A language to specify mappings between ontologies. In Chbeir, R., Dipanda, A., and Yétongnon, K., editors, *SITIS 2005*, pages 267–271. Dicolor Press.

Seidewitz, E. (2003). What models mean. *IEEE Softw.*, 20(5):26–32. 942706.

Sendall, S. and Kozaczynski, W. (2003). Model transformation: The heart and soul of model-driven software development. *IEEE Softw.*, 20(5):42–45.

Shvaiko, P. and Euzenat, J. (2013). Ontology matching: State of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering*, 25(1):158–176.

Siau, K. and Cao, Q. (2001). Unified modelling language: A complexity analysis. *Journal of Database Management*, 12(1):26–34.

Siau, K. and Tian, Y. (2009). A semiotic analysis of unified modeling language graphical notations. *Requirements Engineering*, 14(1):15–26.

Simon, H. A. (1996). *The Sciences of the Artificial*. Springer.

Siricharoen, W. V. (2008). Merging ontologies for object oriented software engineering. In Kim, J., Delen, D., Park, J., Ko, F., and Na, Y. J., editors, *NCM (2)*, pages 525–530. IEEE Computer Society.

Solberg, A., France, R., and Reddy, R. (2005). Navigating the metamuddle. In *4th Workshop in Software Model Engineering*.

Sousa, José, J., Lopes, D., Claro, D., and Abdelouahab, Z. (2009). A step forward in semi-automatic metamodel matching: Algorithms and tool. In Filipe, J. and Cordeiro, J., editors, *Enterprise Information Systems*, volume 24 of *Lecture Notes in Business Information Processing*, pages 137–148. Springer Berlin Heidelberg.

Spivey, J. (1998). The z notation: A reference manual. Technical report, Programming Research Group University of Oxford.

Spyns, P., Meersman, R., and Jarrar, M. (2002). Data modelling versus ontology engineering. *SIGMOD Record*, 31(4):12–17.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2008). *Eclipse Modeling Framework: EMF*. Addison-Wesley Professional.

Stumme, G. and Maedche, A. (2001). Fca-merge: Bottom-up merging of ontologies. In *International Joint conference on Artificial Intelligence*.

Tetlow, P., Pan, J., Oberle, D., Wallace, E., Uschold, M., and Kendall, E. (2006). Ontology driven architectures and potential uses of the semantic web in systems and software engineering. Technical report, w3c.

Tolosa, J. B., García-Díaz, V., Sanjuán-Martínez, O., Fernández-Fernández, H., and García-Fernández, G. (2009). Towards meta-model interoperability of models through intelligent transformations. In Omatu, S., Rocha, M., Bravo, J., Fernández, F., Corchado, E., Bustillo, A., and Corchado, J., editors, *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, volume 5518 of *Lecture Notes in Computer Science*, pages 315–322. Springer Berlin Heidelberg.

Tran, Q.-N. N. and Low, G. (2008). Mobmas: A methodology for ontology-based multi-agent systems development. *Inf. Software Technol*, 50(7-8):697–722.

Uhrig, S. (2008). Matching class diagrams: with estimated costs towards the exact solution? In *Proceedings of the 2008 international workshop on Comparison and versioning of software models*, CVSM '08, pages 7–12, New York, NY, USA. ACM.

van Dongen, B. F., Dijkman, R. M., and Mendling, J. (2008). Measuring similarity between business process models. In Bellahsene, Z. and Léonard, M., editors, *CAiSE*, volume 5074 of *Lecture Notes in Computer Science*, pages 450–464. Springer.

Voigt, K. and Heinze, T. (2010a). Metamodel matching based on planar graph edit distance. In Tratt, L. and Gogolla, M., editors, *ICMT*, volume 6142 of *Lecture Notes in Computer Science*, pages 245–259. Springer.

Voigt, K. and Heinze, T. (2010b). Metamodel matching based on planar graph edit distance. In Tratt, L. and Gogolla, M., editors, *Theory and Practice of Model Transformations*, volume 6142 of *Lecture Notes in Computer Science*, pages 245–259. Springer Berlin Heidelberg.

Wegner, P. (1996). Interoperability. *ACM Comput. Surv.*, 28(1):285–287. 234424.

Welty, C. (2006). *Ontology-Driven Conceptual Modeling*, volume 2348/2006, page 3. Springer.

Werner, E., Andreas, G., and Grit, S. (2004). *Towards a Framework for Model Migration*, volume 3084, pages 545–577. Springer Verlag=Berlin.

Winikoff, M. and Padgham, L. (2004). *THE Prometheus Methodology*, chapter Methodologies and Software Engineering for Agent Systems. Springer.

Woolridge, M. (2001). *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY, USA.

Wouters, B., Deridder, D., and Paesschen, E. V. (2000). The use of ontologies as a backbone for use case management. In *European Conference on Object-Oriented Programming*.

Zambonelli, F., Jennings, N., and Wooldridge, M. (2003). Developing multiagent systems: The gaia methodology. *ACM Trans. Softw. Eng. Methodol.*, 12(3):317–370.

Zhong, J., Zhu, H., Li, J., and Yu, Y. (2002). Conceptual graph matching for semantic search. *Journal of Architecture*, 2393:92–196.

Appendix A

Table 8.2: BPMN Concepts List

BPMN Concepts List		
Definitions	Flow Node	Resource Parameter
Root Element	Event	Conversation
Interface	Data Input Association	Process Type
Callable Element	Data Input	Lane Set
Participant	Implicit Throw Event	Property
Choreography	Throw Event	Activity Resource
Process	Intermediate Throw Event	Performer
Global Task	End Event	Loop Characteristics
Global Choreography Task	Event Definition	Standard Loop Characteristics

Table 8.2: BPMN Concepts List

BPMN Concepts List		
Collaboration	Catch Event	Multi Instance Loop Characteristics
Conversation	Data Output Association	Resource Parameter Binding
Message	Data Output	Resource Assignment Expression
Message Flow	Start Event	Human Performer
Import	Intermediate Catch Event	Potential Owner
Relationship	Boundary Event	User Task
Base Element	Activity	Manual Task
Diagram	Expression	Script Task
Documentation	Flow Element Container	Business Rule Task
Extension	Auditing	Implementation
Extension Attribute Value	Monitoring	Rendering
Extension Definition	Data Object	User Task Implementation
Extension Attribute Definition	Sequence Flow	Subprocess
Relationship Direction	Choreography Activity	Adhoc Sub Process

Table 8.2: BPMN Concepts List

BPMN Concepts List

Element	Gateway	Transaction
Flow Elements Container	Sub Process	Adhoc Ordering
Artifact	Choreography Sub Process	Transaction Method
Association	Gateway Direction	Global User Task
Group	Exclusive Gateway	Global Manual Task
Text Annotation	Inclusive Gateway	Global Script Task
Association Direction	Parallel Gateway	Global Business Rule Task
Category Value	Complex Gateway	Complex Behavior Definition
Category	Event Based Gateway	Multi Instance Behavior
Flow Element	Event Based Gateway Type	Data State
Call Activity	Interaction Specification	Item Aware Element
Call Choreography Activity	Global Communication	Data Store
Input Output Specification	Item Kind	Data Store Reference
Input Set	Send Task	Assignment

Table 8.2: BPMN Concepts List

BPMN Concepts List

Output Set	Receive Task	Data Association
Input Output Binding	Service Task	Cancel Event Definition
Operation	Message Event Definition	Terminate Event Definition
Sub Conversation	Message Flow Node	Compensate Event Definition
Correlation Key	Task	Link Event Definition
Correlation Property	Choreography Task	Timer Event Definition
Correlation Property Binding	Message Flow Association	Conditional Event Definition
Correlation Subscription	End Point	Error Event Definition
Correlation Property Retrieval Expression	Participant Multiplicity	Escalation Event Definition
Formal Expression	Partner Role	Signal Event Definition
Conversation Association	Partner Entity	Escalation
Error	Participant Association	Signal

Table 8.2: BPMN Concepts List

BPMN Concepts List

Item Definition	Resource	Lane
Conversation Node	Correlation Property	Call Conversation
Conversation Container		

Table 8.3: ECore Concepts List

ECore Concepts List

Attribute	Annotation	Class
Classifier	DataType	Enum
EnumLiteral	Factory	ModelElement
NamedElement	Object	Operation
Package	Parameter	Reference
StructuralFeature	TypedElement	

Table 8.4: UML Class Diagram Concepts List

UML Class Diagram Concepts List

Element	Comment	Relationship
DirectRelationship	NamedElement	PackageableElement
Namespace	ElementImpiort	PackageImport

Table 8.4: UML Class Diagram Concepts List

UML Class Diagram Concepts List

Package	MultiplicityElement	ValueSpecification
TypedElement	Type	Constraint
Generalization	Classifier	Feature
StructuralFeature	BehavioralFeature	Parameter
Operation	Class	DataType
Property	Association	PrimitiveType
Enumeration	EnumerationLiteral	PackageMerge
Dependency	Abstraction	Usage
Realization	Substitution	Interface
InterfaceRealization	AssociationClass	RedefendableElement
GeneralizationSet	AggregationKind	ParameterDirection Kind
VisibilityKind		

Table 8.5: OSM Concepts List

OSM Concepts List

Participant	Property	Person
OrgRole	OrgRelationship	PersonSpec
RoleSpec	RelationshipSpec	Constraint
Spec	PropertySpec	

Table 8.6: Concepts List of Gaia, ADELFE and PASSI

Concept List of Gaia, ADELFE and PASSI

Organization	Classifier	FIPA-Platform Agent
Organizational Structure	Environment	FIPA-Platform Task
Safety Rule	Element	Service
Liveness Rule	Representation	Agent
Service	Cooperative Agent	Role
Agent Type	Skill	Task
Organizational Rule	Aptitude	AIP
Environment	Characteristics	Performative
Resource	Communication	Communication
Permission	AIP	Message
Role	Cooperation Rules	Ontology
Communication	NCS	Predicate
Action	Incomprehension	Action
Responsibility	Ambiguity	Concept
Activity	Incompetence	Scenario
Protocol	Unproductiveness	Requirement
Liveness Property	Concurrency	Non Func Requirement
Safety Property	Conflict	Resource
	Uselessness	

Table 8.7: OMaSE Concepts List

OMaSE Concepts List		
Actor	External Protocol	Protocol
Goal	Internal Protocol	Message
Role	Agent	Capability
Plan	Action	Org Agent
Domain Model	Environment Object	Environment Property
Policy	Organization	

Table 8.8: Prometheus Concepts List

Prometheus Concepts List		
Actor	Action	Percept
Role	Data	Goal
Scenario	Agent	Message
Capability	Model Entity	Step
Plan	Protocol	Scoped Entity
Sub Protocol	Pelement	Region
Goto	Label	Box
Scenario Step	Gola Step	Action Step
Percept Step	Other Step	

Table 8.9: BPMN Synonyms List

BPMN Concept	**Synonyms**
Definitions	Explanation, Distinctness
Interface	Surface, Program, Overlap, Port
Element	Component, Substance, Part
Participant	Associate, Contestant
Choreography	Dancing, Dance, Notation
Process	Activity, Cognition, Writ
Task	Work, Activity, Duty
Collaboration	Cooperation
Conversation	Speech
Message	Communication, Content
Flow	Stream, Flux
Import	Importation, Importee, Message
Relationship	Connectedness, Relation, State
Diagram	Plot, Drawing
Documentation	Certification, Support
Extension	Delay, Expansion, Propagation, Denotation, Elongation
Artifact	Artefact
Association	Organization, Connection, Interaction
Category	Division, Family
Operation	Procedure, Business, Planned Activity, Functioning, Process
Error	Fault, Misplay, Wrong Doing
Event	Case
Activity	Action, Behavior, Process
Expression	Look, Aspect, Manifestation, Formulation

Table 8.9: BPMN Synonyms List

Auditing	Scrutinize
Monitoring	Observing
Gateway	Entrance
Partner	Collaborator, Cooperator
Resource	Source, Aid, Wealth
Property	Geographical area, Possession, Attribute, Concept
Transaction	Dealing

Table 8.10: OSM Synonyms List

OSM Concept	**Synonyms**
Participant	Associate, Contestant
Property	Geographical area, Possession, Attribute, Concept
Person	Individual, Human body, Grammatical Category
Constraint	Confinement, Device, Restriction
PropertySpec	No Synonym
OrgRole	No Synonym
PersonSpec	No Synonym
RoleSpec	No Synonym
RelationshipSpec	No Synonym
Spec	No Synonym
OrgRelationship	No Synonym

Table 8.11: Semantic Similarity between BPMN and OSM

BPMN, OSM and their Synonyms

BPMN Concept-Synonym	**OSM Concept-Synonym**	**semSim Value**
Participant	Participant	0.208024691
Participant	Participant	0.208024691
Relationship	Constraint	0.118067151
Property	Property	0.098619329
Documentation	Constraint	0.083075655
Definitions	Constraint	0.078526686
Rendering	Constraint	0.072522848
Auditing	Constraint	0.071712018
Collaboration	Constraint	0.069130076
Participant	Constraint	0.067364976
Participant	Constraint	0.067364976
Auditing	Participant	0.066468254
Escalation	Constraint	0.063726551
Transaction	Constraint	0.063539305
Assignment	Constraint	0.062981046
Expression	Constraint	0.062897547
Association	Constraint	0.061209029
Transaction	Participant	0.057484568
Collaboration	Participant	0.056619769
Choreography	Constraint	0.052035663
Choreography	Constraint	0.052035663
Resource	Property	0.050549451

Table 8.11: Semantic Similarity between BPMN and OSM

BPMN, OSM and their Synonyms		
Performer	Participant	0.050086179
Association	Participant	0.049177718
Relationship	Participant	0.04871633
Task	Constraint	0.048006597
Task	Constraint	0.048006597
Extension	Constraint	0.047625557
Import	Participant	0.04755291
Import	Constraint	0.04736689
Documentation	Participant	0.046212121
Event	Participant	0.045763187
Event	Participant	0.045763187
Message	Constraint	0.043242925
Process	Participant	0.041678358
Process	Participant	0.041678358
Activity	Constraint	0.041342728
Rendering	Participant	0.041031345
Monitoring	Constraint	0.039034378
Expression	Participant	0.038061809
Group	Participant	0.037754375
Auditing	Property	0.037576313
Definitions	Property	0.037192732
Performer	Constraint	0.035754484
Signal	Participant	0.03569249
Activity	Participant	0.03537415

Table 8.11: Semantic Similarity between BPMN and OSM

BPMN, OSM and their Synonyms

Signal	Constraint	0.034222647
Extension	Participant	0.033760854
Documentation	Property	0.033150788
Group	Constraint	0.033017082
Message	Participant	0.032915464
Conversation	Participant	0.03290404
Operation	Participant	0.032422262
Assignment	Participant	0.031957102
Resource	Person	0.031105991
Process	Property	0.030938352
Process	Property	0.030938352
Activity	Property	0.030871708
Task	Property	0.030772561
Task	Property	0.030772561
Lane	Participant	0.030754302
Gateway	Property	0.030716902
Task	Participant	0.028968855
Task	Participant	0.028968855
Gateway	Constraint	0.028847881
Category	Person	0.028582137
Process	Constraint	0.028139387
Process	Constraint	0.028139387
Operation	Constraint	0.028026665
Interface	Property	0.027632368

Table 8.11: Semantic Similarity between BPMN and OSM

BPMN, OSM and their Synonyms

Escalation	Property	0.02711143
Choreography	Participant	0.026590308
Choreography	Participant	0.026590308
Category	Constraint	0.026582792
Relationship	Property	0.026136364
Gateway	Participant	0.024929568
Transaction	Property	0.024588837
Property	Person	0.02403233
Conversation	Constraint	0.023736343
Relationship	Person	0.023361932
Message	Property	0.023267209
Signal	Person	0.023220286
Category	Property	0.023195554
Message	Person	0.022980514
Rendering	Property	0.022737416
Association	Person	0.022628468
Group	Property	0.022555465
Error	Constraint	0.022417199
Escalation	Person	0.022330727
Conversation	Property	0.02220696
Lane	Constraint	0.022069905
Task	Person	0.021589396
Task	Person	0.021589396
Extension	Property	0.021219716

Table 8.11: Semantic Similarity between BPMN and OSM

BPMN, OSM and their Synonyms

Process	Person	0.02103
Process	Person	0.02103
Interface	Person	0.020706444
Event	Property	0.020689033
Event	Property	0.020689033
Monitoring	Property	0.020424234
Interface	Participant	0.02
Definitions	Participant	0.019920799
Expression	Person	0.019820917
Auditing	Person	0.019532221
Assignment	Person	0.019289671
Event	Constraint	0.019200079
Event	Constraint	0.019200079
Transaction	Person	0.019113622
Expression	Property	0.01903266
Operation	Property	0.019021909
Assignment	Property	0.018775523
Monitoring	Participant	0.018615644
Group	Person	0.018597604
Artifact	Constraint	0.018594104
Error	Participant	0.018430335
Performer	Property	0.018275058
Rendering	Person	0.018023871
Import	Property	0.018020273

Table 8.11: Semantic Similarity between BPMN and OSM

BPMN, OSM and their Synonyms

Definitions	Person	0.017906247
Participant	Person	0.017799927
Participant	Person	0.017799927
Escalation	Participant	0.017792508
Import	Person	0.017776597
Association	Property	0.017637521
Signal	Property	0.017611295
Error	Person	0.01737465
Performer	Person	0.017101429
Operation	Person	0.017033201
Gateway	Person	0.017024674
Resource	Constraint	0.017006803
Collaboration	Person	0.016740147
Extension	Person	0.016684933
Collaboration	Property	0.016601256
Activity	Person	0.016357618
Documentation	Person	0.016203301
Interface	Constraint	0.015986395
Event	Person	0.01551372
Event	Person	0.01551372
Participant	Property	0.01529859
Participant	Property	0.01529859
Property	Participant	0.01529859
Choreography	Property	0.015202715

Table 8.11: Semantic Similarity between BPMN and OSM

BPMN, OSM and their Synonyms

Choreography	Property	0.015202715
Conversation	Person	0.015093562
Property	Constraint	0.014994529
Error	Property	0.01478429
Lane	Property	0.014585569
Monitoring	Person	0.014190882
Artifact	Person	0.01378144
Artifact	Participant	0.013059163
Resource	Participant	0.012222222
Lane	Person	0.011889246
Artifact	Property	0.01017871
Choreography	Person	0.010087014
Choreography	Person	0.010087014
Category	Participant	0.004498106

Index