Vrije Universiteit Brussel - Belgium
Faculty of Sciences

In Collaboration with
Universidad Nacional de La Plata (Argentina)
and
Ecole des Mines de Nantes (France)

Towards a Scalable and Collaborative Information Integration Platform and Methodology

2007

A Thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
(Thesis research conducted in the EMOOSE exchange)

Author: Felix Van de Maele

Promoter: Prof. Dr. Robert Meersman (Vrije Universiteit Brussel)
Co-Promoter: Prof. Dr. Alicia Diaz (Universidad Nacional de La Plata)

Abstract

The need to integrate heterogeneous and distributed data sources and models is a problem that is faced in almost any area of computer science. In our global, real-time, and constantly changing information society this need is growing fast. In this thesis, we identify a number of limitation of existing work on information integration. Furthermore, the recent trend towards and acceptance of the need for collaboration have led us to propose a novel methodology and platform to address the increased need for information integration.

In this thesis, we present a two-step methodology that splits up the traditional integration process into two completely separated phases. In the first phase, the *Mapping phase*, heterogeneous models are matched and mappings are created between the corresponding entities. The mappings are stored on a remote server in an application-neutral manner. We further introduce a community and contextual dimension for each mapping. In the second phase of our methodology, the *Commitment phase*, a final, application-specific alignment is created in a certain integration format or model.

We argue that this methodology enables a more scalable, efficient and collaborative integration process. We develop a platform that allows a user to integrate heterogeneous models using this methodology. We furthermore present a preliminary evaluation of our work by going through a real use case: Using our tool, we integrate two real ontologies by following our methodology.

Acknowledgments

First and foremost I would like to thank Professor Diaz for giving me the opportunity to conduct my research and thesis at Lifia and making my experience abroad so enjoyable. I also want to thank her for the numerous insights and the many proof readings and corrections. Without them, this thesis would never be what it is now. My thanks goes also out to all members of Lifia for being so friendly and making my stay in Argentina an unforgettable and invaluable one.

I would also like to thank Professor Meersman and the other members of STARLab for supporting me and giving me the opportunity to finish my thesis in such a fun and exciting environment.

A special thanks goes out to my family and my girlfriend Isabelle for believing in me and supporting my studies. My apologies for the many many lonely days when I spent all my time working on this thesis.

Contents

List of Figures

List of Tables

List of Definitions

1
Introduction

In our so-called information society we constantly demand and require access to available and high-quality information. This information is stored in all corners of our society, be it on the World Wide Web, in government agencies, in public libraries, in the information systems of companies or implicitly in the computer programs we use. These information sources are stored and maintained by different stakeholders, and are heterogeneous and distributed by nature. However, as the need for high-quality information continues to grow, the need to integrate these enormous amounts of distributed and heterogeneous data is becoming ever more important. Furthermore, while these information sources have been developed and deployed with a certain intended use in mind, they may now or in the future be used for very different purposes, for example:

- for governance, to comply with governmental regulations (e.g. Sarbanes-Oxley);
- for intelligence, to be able to make well-informed decisions (e.g. Business Intelligence);
- for engineering new kinds of systems (e.g. context-aware and mobile applications).

1.1 Problem Statement and Motivation

The need to integrate heterogeneous and distributed data sources and models is a problem that is faced in almost any area of computer science. Since long, a lot of research has been done to allow us to cope with the enormous overflow of information around us. However, in our ever more connected society, the needs for information integration have shifted. Nowadays, a much more dynamic, adaptive and agile approach is needed to support the rapidly changing requirements of the different stakeholders. Within enterprises, existing solutions like data-warehousing are no longer sufficient for the real-time and extended enterprise. These trends have led enterprises to increasingly adopt a Service Oriented Architecture (SOA) to address the agile and dynamic nature of our society. However, while such an architecture does enable efficient application integration, it does not provide an efficient and scalable way to integrate information. The same trends can be witnessed in the general software engineering field: More and more systems are developed using a model-driven approach in order to cope with the fast changing requirements and (un)intended use of these systems. Furthermore, the recent trend towards and acceptance of the need for collaboration and what is often referred to as "The Wisdom of Crowds"[65] and "Collective Intelligence"[1] have led us to believe new approaches should be explored to address the increased need for information integration.

We believe that more research is needed to explore these new techniques and trends in order to address the increased and changed integration needs. Our earlier experience with semantic integration frameworks, algorithms, and methodologies has helped us to identify certain shortcomings in existing approaches. More specifically, the lack of a scalable methodology and platform that enables the integration of a very large amount of heterogeneous data sources by a very large number of stakeholders made us question the efficiency and effectiveness of these approaches for real-world scenarios, now and in the future. The lack of support for collaboration and evolution in existing works has furthermore motivated us to propose a novel approach and methodology for data integration.

1.2 Proposed Solution

Given our previous experience with semantic integration frameworks and our interest in the power of collaboration between a large number of actors and communities has led us to this thesis in which we ask ourselves if and how we can address the shortcomings of previous work. Therefore, we propose and present in this thesis an ***integration approach that adopts a uniform, context-aware and collaborative methodology enabling efficient reuse to come to a scalable integration platform***.

1.2.1 Research Questions & Objectives

In this thesis we ask ourselves the following questions which are also the objectives we want to tackle:

[1] *"Collective intelligence is the capacity of human communities to evolve towards higher order complexity and harmony, through such innovation mechanisms as differentiation and integration, competition and collaboration."* [55]

1. What kinds of information heterogeneity exist and where does this heterogeneity come from?
2. What is the existing work on integration methodologies, algorithms and frameworks?
3. Can we propose a methodology based on the notions of collaboration, context-awareness and reuse to attain a scalable integration approach?
4. What would the advantages and disadvantages of such an approach be?
5. Can we develop an platform that implements and support this methodology?

These five questions define the scope of this thesis.

1.2.2 Research Methodology

To answer these research questions and reach our objectives we took a methodological approach. These are the steps we followed:

1. start a literature study on the origin of heterogeneity, the role of semantics and ontologies, and the existing work on information integration;
2. propose a solution by presenting our methodology with its specific and distinct properties and advantages;
3. construct a platform that implements this methodology and do a preliminary evaluation of this platform.

1.3 Outline of this Thesis

The outline of this thesis closely follows our research methodology:

Chapter 2: Background This chapter gives an introduction to semantics in general, the Semantic Web and the OMG's model-driven-architecture initiative. Furthermore it discusses the different types of heterogeneity, the terminology used in the literature on semantic integration and the different methodologies, algorithms and frameworks that have been proposed in the current literature.

Chapter 3: Approach & Methodology In this chapter we present our methodology and approach for a scalable and collaborative integration process. We propose our two-step methodology and discuss the specific and distinct properties of this approach. Next, we present the reader with the semantics of the elements used in our methodology. We conclude this chapter by reviewing the advantages our methodology brings to the integration process.

Chapter 4: The Platform In chapter 4 we present the integration platform implementing the methodology and conceptual framework we proposed in the previous chapter. We discuss how we have mapped our two-step methodology on the platform and discuss the two layers of the platform, the remote and client layer, in more detail. We conclude this chapter with a preliminary evaluation of the platform and show how it can be used to integrate two ontologies using our proposed methodology.

Chapter 5: Conclusions Chapter 5 is the last and final chapter that concludes the main contributions of this thesis by discussing the research questions and answers. Based on the work in the previous chapters a number of issues are discussed and a number of opportunities for future research are highlighted.

2 Background

In this chapter, we give an introduction to semantics in general, the Semantic Web and the OMG's model-driven-architecture initiative. Furthermore we discuss the different types of heterogeneity, the terminology on semantic integration used in the literature and the different methodologies, algorithms and frameworks that have been proposed in existing works.

2.1 Semantics

Semantics is usually defined as *the study of meaning*. Depending on the field in which it is used, a more restricted definition may be used. For example, in linguistics, semantics is often defined as *the study of the meaning of words*. On the other hand, in computer science semantics usually refers to the *meaning of programs or functions*. We will use the broader definition throughout this thesis.

It is interesting to mention two terms that are related to semantics: *Syntax* and *Pragmatics*. The three find themselves adjacent to each other in the domain of metaphysics which is the branch of philosophy concerned with explaining the nature of reality, being, and the world. Figure 2.1 depicts the different branches in metaphysics. Syntax, as a philosophical study, is concerned with first-order logic, or how to construct very basic grammars. In linguistics, syntax refers to the rules that make op the structure of the sentence. In computer science, syntax may refer to the structure and grammar of the language in which programs are written. Pragmatics, on

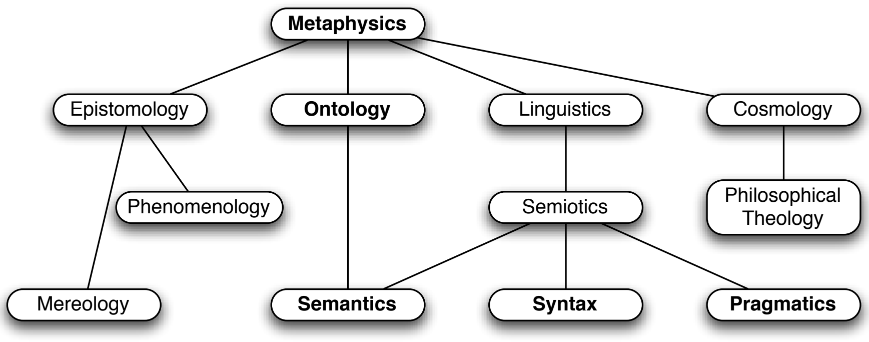


Figure 2.1: Semantics in relation to other branches of metaphysics

the contrary, is a branch of semiotics concerned with the relationship between language and the people using it. In other words, it is concerned with bridging the explanatory gap between *sentence meaning* and *speakers meaning*. A crucial concept in pragmatics is *context*. Pragmatics studies how context influences the interpretation. As we will see, pragmatics are very important in applying semantic technologies in practice. For these reasons, the notion of *context* will also play a central role in our integration methodology and platform.

In the following sections, we will discuss the different aspects and applications of semantics and semantic technologies. Some of the questions we will try to answer include: Where do semantics come from? In what form are the used and constructed? What is the origin of semantic heterogeneity? Why does semantics matter and may it be applied in software engineering? How can it be applied to the web and what role do they play on the web? How can we successfully manage semantic information?

Throughout the discussion of these topics, we will come across several scenarios that show the need for and importance of semantic integration. During this introduction, we will point out these problematic cases in which semantic integration is needed. We will also point out the role of context and community in the field of semantics and more specifically in semantic integration. This introduction to semantics and its uses can therefore be seen as the motivation for this thesis: a scalable, uniform, context-aware, collaborative semantic integration methodology and platform.

2.1.1 From Vocabulary to Ontology

Before we discuss the different forms of structured semantics, we must first introduce the concept of terminology. Terminology is concerned with the naming of things or concepts. Different terms exist for the same concept, and similar terms can refer to different concepts or things depending on the context in which they are used. For example, the term ”bank” can refer to ”a financial institution” or it can refer to ”the land alongside or sloping down to a river or lake”, depending on the context in which the term is used. Terms and facts are semantic constructs.

They are meant to have meaning that is shared by anyone who uses them. Unless producers and consumers of information agree, the information will be meaningless.

Semantic information comes in various forms. The first distinction can be made between unstructured and structured information. Every unstructured form of data has a form of semantics attached to it, albeit implicitly in the human interpretation of this data. In structured or semi-structured data sources, the form of semantics attached to this data range form implicit to explicit as the expressiveness and complexity of the data increases. The simplest and most used form of structured semantics is as a vocabulary which is simply an agreement on terms. The most complex form of semantics are found in ontologies which allow relations between concepts and tries to conceptualize the world. We will now give a brief overview of the different forms of semantics, from their simplest form as vocabularies to their most complex form as ontologies.

Vocabulary

A Vocabulary is the set of symbols to which a given language has ascribed a shared meaning. Each community (in the sense of ”a group of people”) creates its own vocabulary. On the highest level, the group of people speaking the same language have created a vocabulary for that language (note however that in average each of us understands only between 2% and 4% of the vocabulary we supposedly share [47]). Underneath the language level, several sub-communities create their own vocabulary on top of this. For example, each industry traditionally has its own vocabulary. Similarly, each business within this industry typically creates its own vocabulary which is added to the vocabulary of its industry.

Communities have several techniques to construct new vocabularies [47]:

1. **Overloading of common words:** Different communities can overload existing words and give them a special meaning specific to that community. This is a large source of misunderstanding between communities and creates a lot of the integration problems we address in this thesis. Similar terms that have a different meaning are called *homonyms*. Lets illustrate this with an example. Take the term *”router”*. Depending on the community, this term means: A device that carves rounded edges on molding for woodworkers, a person who lays out a circuit board in the electronics industry, an electronic device that dispatches packets in the data communications industry, and a person who stocks vending machines in the snack food industry.

   The problem of homonyms is where pragmatics and more specifically *context* comes in to play. The context in which a term is used is crucial to be able to infer to which concept or thing the term refers. As we will see, in most of the existing semantic technologies, the notion of context does not exist, at least not explicitly.

2. **Creating new words:** To reduce the number of overloaded words, communities also create new words to denote community-specific concepts. Created words come in different

forms, many are acronyms, some are technical terms, some are borrowed from foreign languages, etc. Interestingly, many technical terms are borrowed from dead languages such as Greek and Latin. This trend is also community-dependent, for example, in the medical and law domain many terms come from Latin, while in the computer science domain many acronyms are used.

3. **Nonword identifiers:** A technique that is typically seen in industries is the use of nonword identifiers. In industry and company-specific vocabularies, many concepts or things are referred to by an identifier. This technique is used more in small communities. For example, companies often refer to their products using the product identifiers. Only when these concepts are widely used outside the company will they become more commonly used.

4. **Compounds:** Another tricky area is the use of compounds. Compounds are two or more existing words combined to create a new meaning. Similarly to the technique of overloading existing words, this can create the illusion of understanding where there is none. The problem, both for humans and for systems attempting to interpret language, is determining when a phrase is really a word or, more correctly, determining when several words together stand for one semantic thought[47]. For example, a *"claim check"* appears to mean "review of a claim", but it actually means "a piece of paper that lets you retrieve your property". For a more detailed treatment of this subject, we refer to [61].

A traditional way to specify vocabularies and their meanings is through definitions. A community may create a glossary to define the terms in a vocabulary and how they will be used in a particular system. However, there are several issues with capturing semantics using only definitions [47]: First, glossaries may suffer from "design time" artefacts: A design time artefact is one that is used while the system is being developed but generally ignored by users and maintainers of the system. Secondly, the glossary may suffer because it is created by inexperienced workers. Third, the glossary may suffer for the same reason that more general-purpose glossaries fall short. Glossaries and vocabularies are not designed to distinguish or group closely related concepts; they are designed to "define". As a result, most are full of ambiguity and overlap. In effect, this leaves much of the defining to the reader.

Taxonomy

A taxonomy is the next step from vocabulary towards more explicit and formal semantics. It is a form of organized vocabulary. It is generally composed of a hierarchical parent-child classification.

Ontology

The notion of *ontology* was first used in the field of Philosophy. The term *Ontology* (with an uppercase 'O') was later used by AI practitioners [1]. Ontologies are becoming increasingly popular in the domain of computer science. Recently they have been moving from the realm of AI-laboratories to the desktops of knowledge experts and are now one of the fundaments of the Semantic Web.

Several definitions have been proposed in literature [2]. One of the most cited definitions of an ontology is defined by Gruber [32]

> **Definition 1 (Ontology by Gruber) :**
> An ontology is a specification of a shared conceptualization.

Another, closely related definition by Ushold and Gruninger in [66] is:

> **Definition 2 (Ontology by Ushold and Gruninger) :**
> Ontologies are agreements about shared conceptualizations

The latter definition is less strict: In this definition, ontologies and conceptualisations are kept clearly distinct. An ontology in this sense is not a specification of a conceptualisations, it is a (possibly incomplete) agreement about a conceptualisation [63].

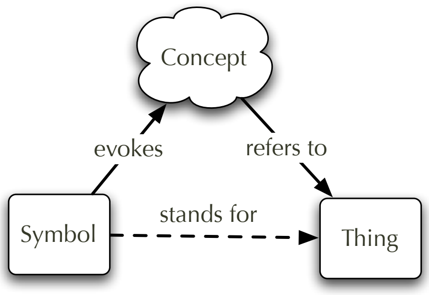


Figure 2.2: The meaning triangle [53]

An ontology always includes a vocabulary of representational concept labels to describe a shared domain. These concept labels are usually called terms (lexical references represented by symbols) and are associated with entities (non lexical referents – the concepts) in the universe of discourse. The meaning triangle [53], depicted in Figure 2.2, is a figure that is used to explain the relationship between Concepts, Symbols and Objects or Things. Formal axioms are also introduced to constrain their interpretation and well-formed use.

[1] AI researchers use ontologies mostly for building Knowledge Bases
[2] See [33] for more definitions

An ontology is in principle a formalisation of a shared understanding of a domain that is agreed upon by a number of agents [63]. In order for this domain knowledge to be shared amongst agents, they must have a shared understanding of the domain and therefore, agreement must exist on the topics about which to communicate. This raises the issue of ontological commitment which Gruber [32] describes as *"the agreements about the objects and relations being talked about among agents"*. In other words, in order to facilitate meaningful communication, an agent must commit to the semantics of the terms and relationships in the common ontology [56]. This includes axioms about properties of objects and how they are related, also called the semantic relationships of the ontology.

2.1.2 The Dogma Approach

DOGMA[3] is a research initiative of VUB STARLab where various theories, methods, and tools for ontology engineering are studied and prototypes developed. Our description of the DOGMA framework has been inspired by several previous papers such as [10, 58, 63, 17, 21]. A full formalisation of DOGMA can be found in De Leenheer, Meersman, and de Moor [20, 19].

The DOGMA approach to ontology engineering aims to satisfy real world needs by developing a useful and scalable ontology engineering approach. A DOGMA ontology is inspired by a classical model-theoretic perspective [60] and articulates an ontology in two distinct layers. This is called the principle of *double articulation* [64]. DOGMA is a representation model and framework that separates the specification of the *conceptualisation* (i.e. the lexical representation of concepts and their interrelationships, see the ontology definitions 1 and 2) from its *axiomatisation* (i.e. the semantic constraints).

In DOGMA's double articulation principle, an ontology is decomposed into a *lexon base* and a *commitment layer*

The Lexon Base

The Lexon Base is an uninterpreted, extensive, and reusable pool of elementary building blocks for constructing an ontology [18]. These building blocks are defined as Lexons, formalised in definition 3.

Definition 3 (Lexon) :
A lexon is an ordered 5-tuple of the form $< \gamma, \zeta, t_1, r_1, r_2, t_2 >$ with $\gamma \in \Gamma$, $\zeta \in Z$, $t_1 \in T$, $t_2 \in T$, $r_1 \in R$ and $r_2 \in R$. Where:

- T and R are sets of strings;
- t_1 is called the head-term of the lexon and t_2 is called the tail-term of the lexon;

[3]Developing Ontology-Grounded Methods for Applications

- r_1 is the role of the lexon and r_2 is the co-role of the lexon;
- γ is the context in which the lexon holds;
- ζ is a code that refers to the natural language in which the lexon is defined.

A lexon $< \gamma, \zeta, t_1, r_1, r_2, t_2 >$ is a fact that *might* hold in a domain, expressing that within the context γ and for the natural language ζ, an object of type t_1 might *plausibly* play the role r_1 in relation to an object of type t_2. On the other hand, the same lexon states that within the same context γ and for the same language ζ, an object of type t_2 might play the role r_2 in (the same) relation to an object of type t_1. This description of lexons shows that they represent *plausible binary fact types* (e.g., Person drives/is_driven_by Car).

Definition 4 (The Lexon Base) :
The Lexon Base Ω as a structure $< T, R, \Gamma, Z, D, \Lambda >$ where:

- $T \subseteq \mathcal{T}$ is a non-empty finite set of terms that occur in the Lexon Base,
- $R \subseteq \mathcal{R}$ is a non-empty finite set of role names that occur in the Lexon Base,
- $\mathcal{D}$ is a non necessarily finite document corups,
- Γ is a finite set of context identifiers,
- $Z \subseteq \mathcal{Z}$ is a non-empty finite set of natural language codes,
- Λ is a finite set of 5-tuples: $\Lambda \subseteq \Gamma \times Z \times T \times R \times RT \subseteq \mathcal{L}$. These 5-tuples are called lexons.

Logically, since lexons represent *plausible* fact types, this database of lexons can become very large. To guide an ontology engineer through this database, *contexts* impose a meaningful grouping of the *lexons*. The context of a lexon refers to the source it was extracted from. This source could be terminological or human domain experts.

It is important to note that this lexon base is uninterpreted. For example, while a lexon like $< \gamma$, manager, is-a, subsumes, person $>$ might intuitively express a specialisation relationship, the interpretation of a role/co-roe label pair as being a *part-of* or *specialisation* relation, is postponed to the commitment layer, where the semantic axiomatisation takes place.

The Commitment Layer

The Commitment Layer can be reified as a separate layer, mediating between the lexon base and the applications that commit to the lexon base. Committing to the Lexon Base means selecting a meaningful set Σ of lexons from the Lexon Base that approximates well the intended[4] conceptualisation, and subsequently putting semantic constraints on this subset. The result (i.e., Σ

[4]With respect to the application domain.

plus a set of constraints), called an *ontology commitment*, is a logical theory that intends to model the meaning of this application domain [18]. The set of constraints in the ontology commitment are specific to an application (*intended conceptualisation*) using the ontology.

The Concept Definition Server

As we have introduced in the previous sections, a lexon is basically a lexical representation of a plausible conceptual relationship between two concepts, though there is no one-to-one mapping between a lexical representation and a concept. Therefore, a higher conceptual level is introduced [64]. As a result not only do we have a double articulation between the Lexon Base and the Commitment layer, we also have two different kind of levels in the DOGMA ontology, namely the Language level (where the Lexon Base is located) and the Conceptual Level.

On the Conceptual Level, we have the Concept Definition Server (CDS). The idea for a Concept Definition Server was first mentioned in [10] and is inspired by lexical databases such as Wordnet [50]. In line with these lexical databases, for each (lexical) term, a set of senses is kept (comparable to *synsets* in Wordnet). A concept definition is unambiguously explained by a gloss (i.e. a natural language (NL) description) and a set of synonymous terms. Consequently we identify each concept definition in the CDS with a concept identifier $c \in \mathcal{C}$. The following definition specifies the CDS:

Definition 5 (Concept Definition Server) :
We define a Concept Definition Server Υ as a triple $< T_\Upsilon, \mathcal{D}_\Upsilon, concept >$ where:

- *T_Υ is a non-empty finite set of strings (terms),*
- *$\mathcal{D}_\Upsilon$ is a non-empty finite document corpus,*
- *concept : $\mathcal{C} \longmapsto \mathcal{D}_\Upsilon \times \wp(T_\Upsilon)$ is an injective mapping between concept identifiers $c \in \mathcal{C}$ and concept definitions.*

Further, we define $conceptdef(t) = \{concept(c) \,|\, concept(c) \ = < g, sy > \ \wedge t \in sy\}$,
where gloss $g \in \mathcal{D}_\Upsilon$ and synset $sy \subseteq T_\Upsilon$.

Going from the language level to the conceptual level corresponds to articulating lexons into meta-lexons. The articulation of a (lexical) term to a Concept in the Concept Definition Server is also called the *lift-up* of a term. In figure 2.3, we illustrate the two levels in DOGMA: on the left - the lexical level, lexons are elicited from various contexts. On the right, there is the conceptual level consisting of a concept definition server. The meaning ladder in between illustrates the articulation (or lift-up, therefore the "ladder" terminology) of lexical terms into concept definition.

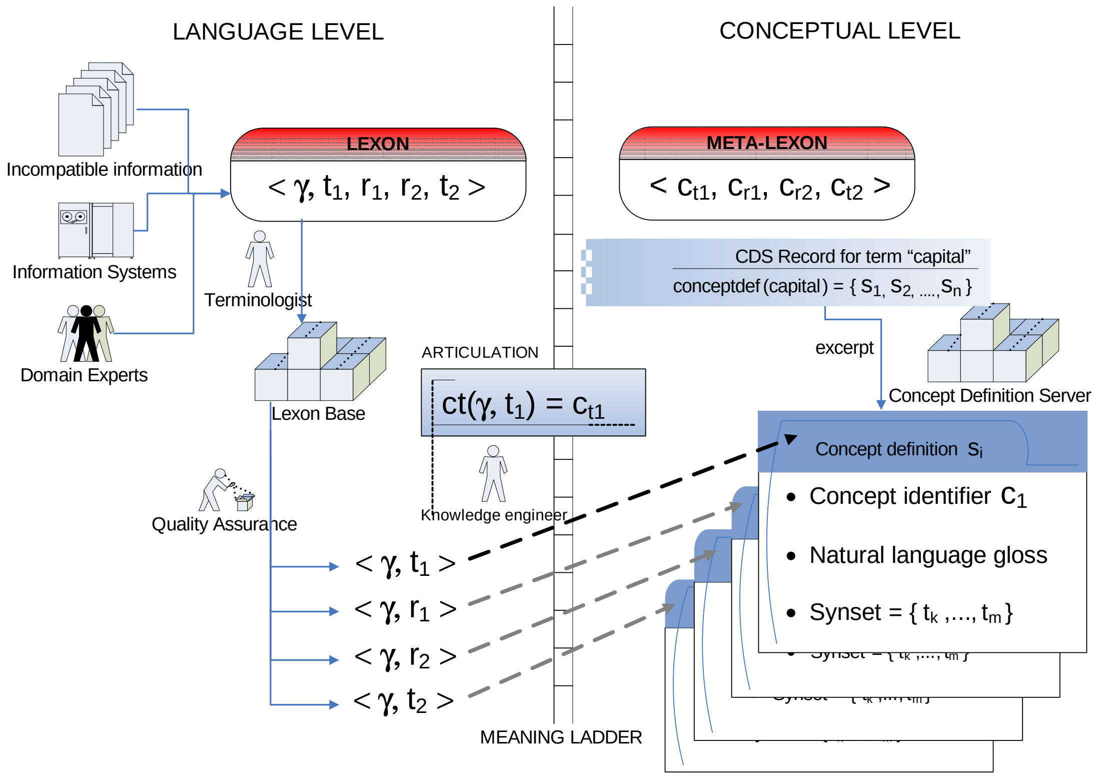


Figure 2.3: Illustartion of the two levels in DOGMA ontology. (Reproduced from [20])

Definition 6 (Meta-Lexon Base) :
Given a Lexon Base Ω and a total articulation mapping $ct : \Gamma \times T \cup R \rightarrow \mathcal{C}$, a Meta Lexon Base $M_{\Omega,ct} = \{m_{l,ct} | l \in \Omega\}$ can be induced.

Example: As an illustration of the defined concepts, consider Figure 2.9. The term "bank" in two different contexts can be articulated to different concept definitions in the CDS. The terms are part of some lexons residing in the Lexon Base. The knowledge engineer first queries the CDS Υ for the various concept definitions of the term: $concepdef(bank) = S_{bank} \subseteq \mathcal{D}_{\Upsilon} \times \wp(T_{\Upsilon})$. Next, he articulates each term to the concept identifier of the appropriate concept definition:

- The term "bank" was extracted from a seaport navigation document, and is articulated to a concept identifier c_1 that corresponds to concept definition (or meaning) $s_1 \in S_{capital}$ (as illustrated on the right of Figure 2.9). A gloss and set of synonyms (synset) is specified for s_1.
- The term "bank" was extracted from a financing book, due to the different context it was extracted from, it is articulated to another concept identifier c_2 that is associated with a concept definition $s_2 \in S$.

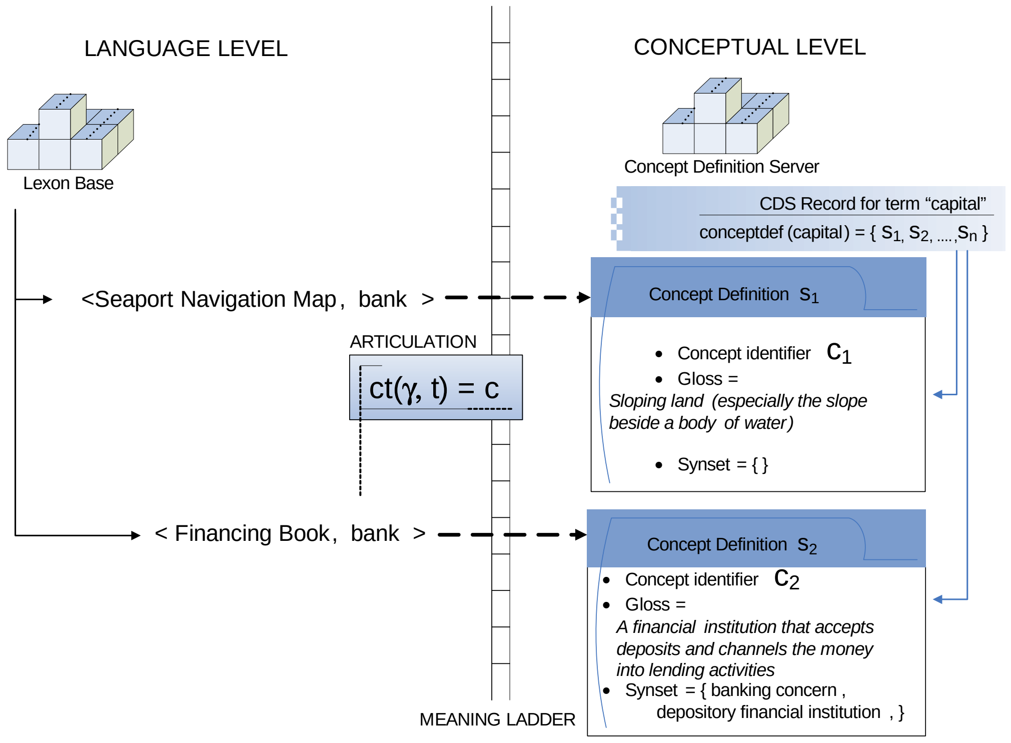

LANGUAGE LEVEL
CONCEPTUAL LEVEL
Lexon Base
Concept Definition Server
CDS Record for term "capital"
conceptdef (capital) = { S1, S2, ..., Sn }
<Seaport Navigation Map, bank >
< Financing Book, bank >
ARTICULATION
ct(γ, t) = c
Concept Definition S1
Concept identifier C1
Gloss =
Sloping land (especially the slope beside a body of water)
Synset = { }
Concept Definition S2
Concept identifier C2
Gloss =
A financial institution that accepts deposits and channels the money into lending activities
Synset = { banking concern , depository financial institution , }
MEANING LADDER

Figure 2.4: Illustration of two terms (within their resp. contexts), being articulated (via the mapping ct) to their appropriate concept definition. (Inspired by [20])

2.2 Semantics on the Web: The Semantic Web

The advent of the World Wide Web (WWW) has taken the availability of information to an unprecedented level. The rapid growth of the web poses new problems. Anyone can easily publish a new document or add a link to a site with no restrictions on the structure or validity of the new content. The lack of restrictions have been partly the reason for the success of the web. It has kept the expertise necessary to create content for the web low and thereby originated the amount of available content. The overabundance of unstructured and possibly faulty information has made it difficult for the users of the web to find relevant information easily. It also poses scaling difficulties on current web crawlers and search engines.

These difficulties have given rise to the successor of the current web: the Semantic Web. The Semantic Web is a project that intends to create a universal medium for information exchange by giving meaning (semantics), in a manner understandable by machines, to the content of documents on the Web. Currently under the direction of the Web's creator, Tim Berners-Lee of the World Wide Web Consortium (W3C), the Semantic Web extends the World Wide Web through the use of standards, markup languages and related processing tools[5]. With the Semantic Web

[5] *http : //en.wikipedia.org/wiki/Semantic_web*

we not only receive more exact results when searching for information, but also know when we can integrate information from different sources, know what information to compare, and can provide all kinds of automated services in different domains from future home appliances and digital libraries to electronic business and health services[8].

2.2.1 The Semantic Web Technology Stack

Currently, the World Wide Web consists primarily of documents written in HTML. This makes the the Web readable for humans, but since HTML has limited ability to classify the blocks of text apart from the roles they play, the Web in its current form is very hard to understand for computer agents. The purpose of the Semantic Web is to add a layer of descriptive technologies to web pages so they become readable and can be reasoned about by computer agents.

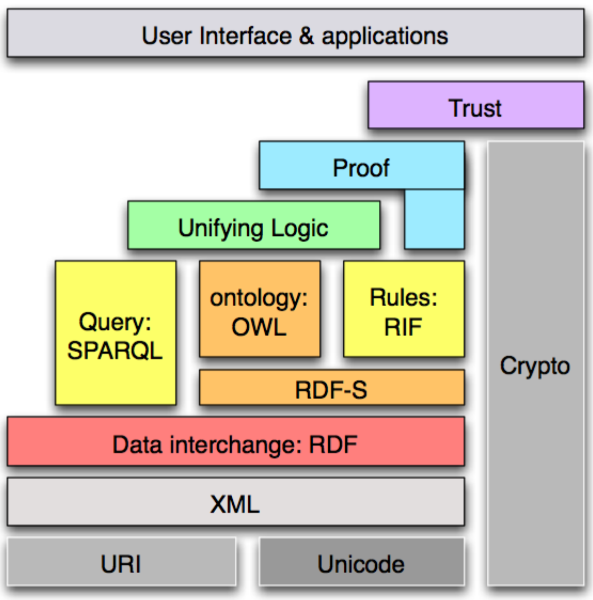


Figure 2.5: The Semantic Web layers

The Semantic Web principles are implemented as a technology stack consisting of layers of Web technologies and standards. The layers are presented in Figure 2.5 and described as follows:

- The *Unicode* and *Uniform Resource Identifier (URI)* layers make sure that we use international characters sets and provide means for identifying the objects in the Semantic Web, respectively. The most popular URI's on the World Wide Web are Uniform Resource Locaters (urls).

- The *XML layer* with *namespace* and *schema* definitions enables the integration of the Semantic Web definitions with the other XML based standards. XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documentes. XML Schema is a language for restricting the structure of XML documents.
- The *ontology* layer supports the evolution of vocabularies as it can define relations between the different concepts. The structure is simple: knowledge is expressed as descriptive statements, stating some relationship exists between one thing and another. The technologies to represent that structure are already in place[6]:
  - The Resource Description Framework (RDF) is a simple data model for referring to objects ("resources") and their relations to each other. An RDF-based model can be represented in XML syntax classes or in n3-notation.
  - RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.
  - The Web Ontology Language (OWL) adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.
  - SPARQL [57] is a query language for RDF and is a W3C candidate recommendation since 2006. It has a similar syntax as the well-known SQL but operates on triples instead of on relational tables.
  - Early 2005, the W3C formed the Rule Interchange Format (RIF) working group[7] with the task of standardizing the rules that propel data across the Web, regardless of format. RIF will provide a way to allow rules written for one application to be published, shared, merged and re-used in other applications and by other rule engines.
- The top layers *Logic, Proof* and *Trust*, are currently being researched and simple application demonstrations are being constructed. The Logic layer enables the writing of rules while the Proof layer executes the rules and evaluates together with the Trust layer mechanism for applications whether to trust the given proof or not.

RDF

RDF is an abstract data model that defines relationships between resources. These resource can be web-pages or surrogates for real world objects (e.g. persons). Meaning (semantics) is thus encoded in sets of 'triples': in a graphical representation the source of the relationship is called the *subject*, the labeled *arc* is the *property* and the relationship's destination is the *object*.

On its conception the most important design goals RDF is intended to meet are [40]:

[6] *http : //en.wikipedia.org/wiki/Semantic_web*
[7] http://www.w3.org/2005/rules/

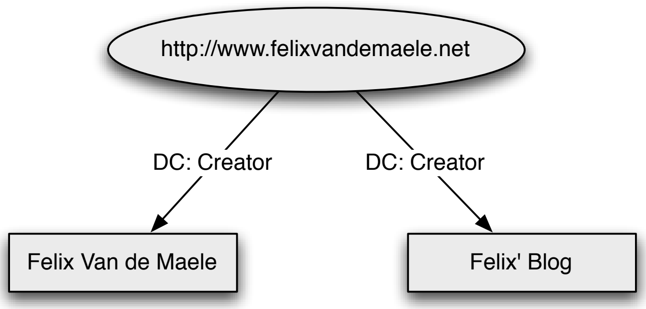


Figure 2.6: An RDF example.

- A *simple data model*. RDF has a simple data model that is easy for applications to process and manipulate. Note that the term "model" used here has a completely different sense than that in the term "model theory", where a model is a satisfying interpretation [34].
- *Formal Semantics and Inference*. Because the Semantic Web is all about machine processable content, RDF has a formal semantics which provides a dependable basis for reasoning about the meaning of an RDF expression and allow inferencing.
- *Extensible URI-based vocabulary*. Based on URI it allows for naming all kinds of things in RDF. Besides the URIs there is one other kind of value that appears in RDF data which is a literal.
- Anyone can make *statements* about any resource.

In the RDF data model we distinguish between *resources*, which are objects represented by URIs, and *literals* which are strings. These resources may be related to each other or to literal values via *properties*. Such a relationship may also be considered as a resource, which makes it possible to make statements about statements.

The Dublin Core is one standard for a set of descriptors (such as the title, publisher, subjects, etc.) that are used to catalog a wide range of networked resources, such as digitized text documents, photographs and audiovisual media. The following is an RDF example that makes use of The Dublin Core[8].

```
<rdf:RDF>
  <rdf:Description rdf:href=Òhttp://www.felixvandemaele.netÓ>
      <dc:Creator>Felix Van de Maele</dc:Creator>
      <dc:Title>Felix’ Blog</dc:Title>
  </rdf:Description>
</rdf:RDF>
```

Figure 2.6 gives a graphical representation of this example. For more details about RDF we refer to [42].

[8]http://www.dublincore.org/

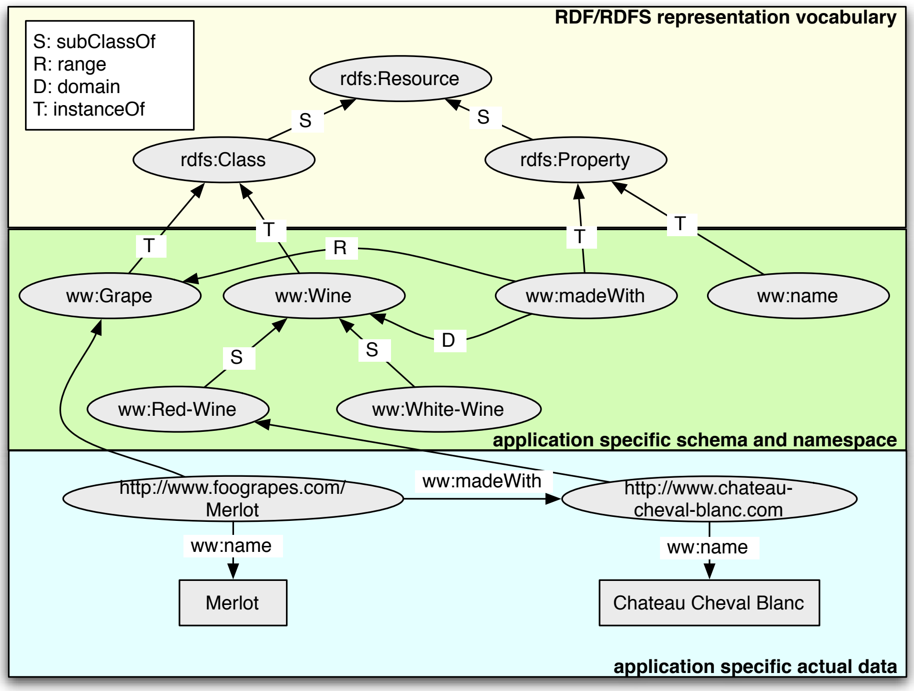


Figure 2.7: An RDF-Schema example.

RDFS

RDF-Schema is an RDF application that introduces an extensible type system to RDF. With RDFS we can define class hierarchies and domain and range restrictions for properties.

Figure 2.7 shows an example ontology (wine-world) modeled with RDFS[9].

- The most general class is `rdf:Resource`, which has two subclasses `rdfs:Class` and `rdf:property`. When specifying a domain specific schema for RDF(S), the classes and properties defined in this schema will become instances of these two resources.
- The resource `rdfs:Class` denotes the set of all classes in an object-oriented sense. This means that the classes `ww:Grape` and `ww:Wine` are instances of the meta-class `rdfs:Class`.
- Likewise, each property defined in an application specific RDFS is an instance of `rdf:Property` (e.g. `ww:madeWith`).
- RDFS provides `rdfs:subClassOf`, which is a special property, that defines the subclass relationship between classes. Likewise there is rdfs:subPropertyOf that defines a hierarchy of properties.

[9]The figure falsely indicates that Chateau Cheval Blanc consists entirely of Merlot. Correct would be an encepagement of 50% Cabernet Franc and 50% Merlot [46].

- RDFS allows to define domain and range restrictions associated with properties. In our example only `ww:Wine`s can only be made with `ww:Grape`s.

The following is a simple illustration of the RDFS syntax. It declares three classes. The classes "red wine" and "white wine" are declared to be subclasses of the class "wine".

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xml:base="http://www.wine-world.uto/wines">

<rdfs:Class rdf:ID="wine" />

<rdfs:Class rdf:ID="red wine">
<rdfs:subClassOf rdf:resource="#wine" />
</rdfs:Class>

<rdfs:Class rdf:ID="white wine">
<rdfs:subClassOf rdf:resource="#wine" />
</rdfs:Class>
</rdf:RDF>
```

RDF Schema provides basic capabilities for describing RDF vocabularies, but still some important semantic aspects are missing (this list is not exhaustive):

- *cardinality* constraints on properties
- *uniqueness* constraint on properties of a class
- *range* constraints
- ability to describe new classes as a *union* or *intersection* of other classes

For a deeper discussion on RDFS we refer to `http://www.w3.org/TR/rdf-schema/`

OWL

OWL stands for Web Ontology Language and extends RDFS. OWL facilitates greater machine interpretability of Web content by providing additional vocabulary along with a formal semantics. OWL provides three increasingly expressive sub-languages:

- *OWL Lite* supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1.
- *OWL DL*[10] allows some extra constraints, but guarantees computational completeness and decidability. OWL DL includes all OWL language constructs, but they can be used only

[10]Description Logics

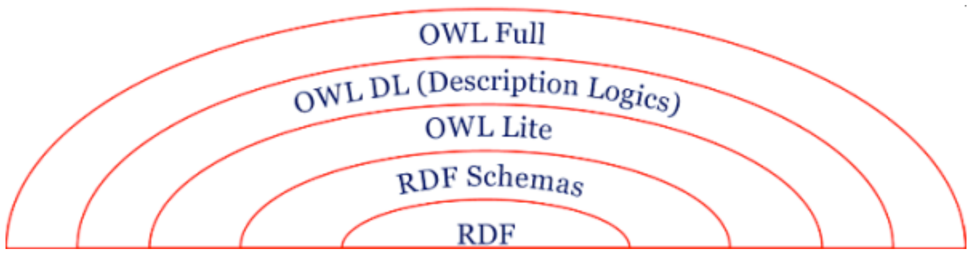


Figure 2.8: The Onion Model.

under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class).

- *OWL Full* allows maximum expressiveness and syntactic freedom, but without the guarantee of computational completeness and decidability. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right.

It is hard to implement a full ontology management system and it may be superfluous for some applications. Therefore W3C suggests the Onion Model of increasingly complex specs (cfr. Figure 2.8).

On `http://www.w3.org/TR/owl-features/` the interested reader can find an in-depth explanation of OWL.

SPARQL

SPARQL is an RDF query language; its name is a recursive acronym that stands for SPARQL Protocol and RDF Query Language. It is undergoing standardization by the RDF Data Access Working Group (DAWG) of the W3C. SPARQL enables the querying of rdf graphs in a similar way like SQL queries relational databases. Its syntax is also very similar to that of SQL. At this point, only read queries are possible.

For example:

The data: an RDF Object - Predicate - Subject triple

```
<http://example.org/book/book1>
<http://purl.org/dc/elements/1.1/title>
"SPARQL Tutorial"
```

The SPARQL query:

```
SELECT ?title
WHERE {
    <http://example.org/book/book1>
    <http://purl.org/dc/elements/1.1/title>
    ?title .
}
```

The result:

title
"SPARQL Tutorial"

Table 2.1: The SPARQL query result

2.3 Semantics in Software Engineering

2.3.1 The OMGs Model Driven Architecture

The Object Management Group (OMG), a standardization consortium for various aspects of software engineering including the well-established Unified Modeling Language (UML, [4]) has introduced, not so long ago, the Model Driven Architecture (MDA, [14]) paradigm. This approach has the ability to create (using metamodeling) a family of languages [25] that are defined in a similar way as UML.

Since the definition of UML, there has been a new wave of proposals at the OMG which is evidence of a new era and a new vision. At the center of this vision is the Meta-Object Facility (MOF, [3]), a unique and self-defined meta-meta-model. It acts as a framework to define and use meta-models [28, 16]. The need for MOF resulted from the fact that UML was only one of the meta-models in the software development landscape. Because of the risk posed by the presence of a variety of different, incompatible meta-models being defined and evolving independently (data warehouse, workflow, software process, etc.) there was an urgent need for a global integration framework for all the meta-models in the software development industry [9].

The MDA defines an approach whereby you can separate the system functionality specification from its implementation on any specific technology platform. That way you can have an architecture that will be language, vendor and middleware neutral. For creating MDA-based applications, the first step will be to create a Platform Independent Model (PIM), which you should express in UML. Such a PIM can then be mapped to a Platform Specific Model (PSM) to target platforms like the CORBA Component Model (CCM), Enterprise JavaBeans (EJB) or Microsoft Transaction Server (MTS). Standard mappings should allow tools to automate some of the conversion. Such a PSM, again expressed in UML, can then be actually implemented on that particular platform.

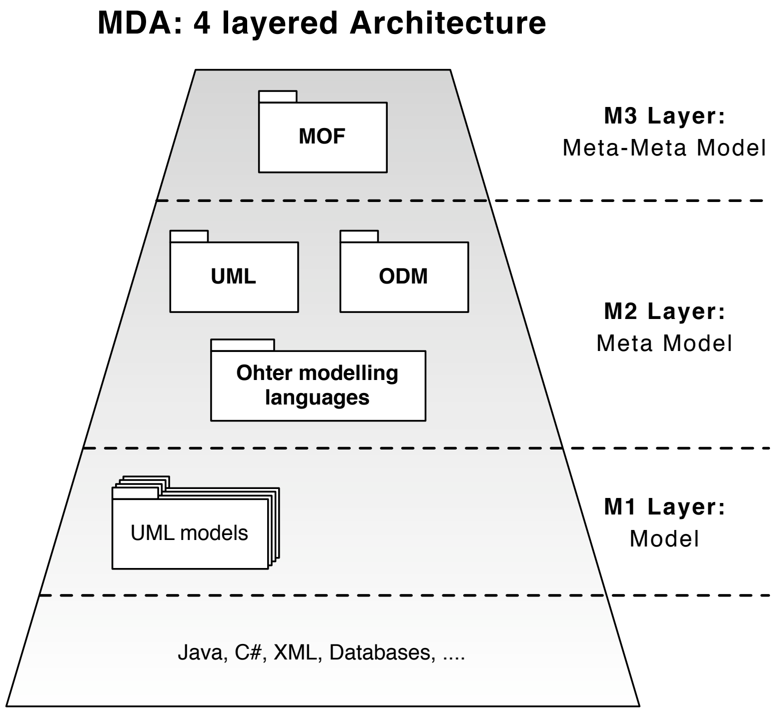

MDA: 4 layered Architecture
MOF
M3 Layer:
Meta-Meta Model
UML
ODM
Ohter modelling languages
M2 Layer:
Meta Model
UML models
M1 Layer:
Model
Java, C#, XML, Databases, ...

Figure 2.9: Illustration of the 4-layers of the OMGs Model Driven Architecture

The analogies between the OMGs Model Driven Architecture approach, and the approach to ontology engineering by STARLab's DOGMA initiative is remarkable. Figure 2.10 shows a conceptual comparison of the two paradigms.

2.3.2 The Ontology Definition Model

As we have discussed before, the standardization of the Web Ontology Language (OWL, [23]) by the World Wide Web Consortium (W3C) and the compelling vision of the Semantic Web [8] contributed heavily to the wide-spread use of ontologies. In order to overcome the gap between software engineering practitioners and the formal ontology practitioners coming from the AI and Knowledge Management field, the OMG replied to this evolution by issuing a Request for Proposal for an Ontology Definition Metamodel (ODM, [1]).

The intention was to provide a MOF based metamodel to support the development of ontologies using UML modeling tools and the two-way transformation between ontologies written in a specific ontology representation language and ontologies modeled using a dedicated UML syntax. Since that time, a submission team has developed a submission (see [15] for a concise overview) which has undergone several revisions, based on comments solicited not only of the OMG but from the W3C, ISO and Semantic Web communities as well.

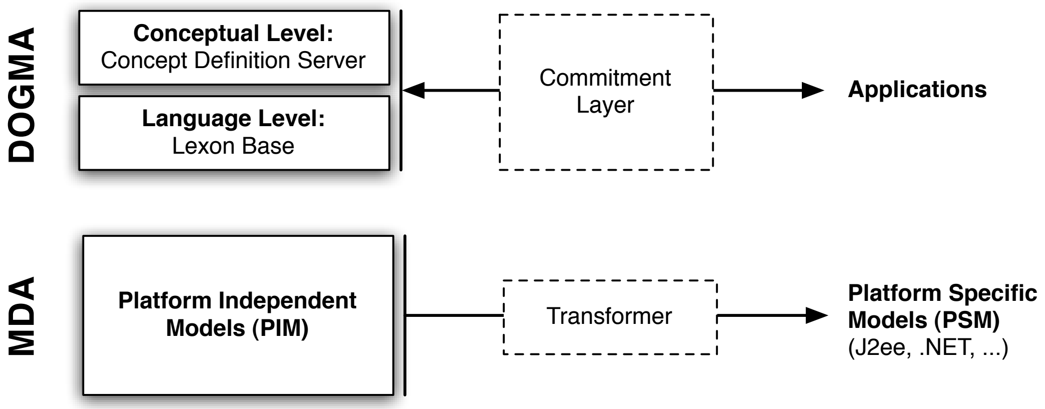

DOGMA
Conceptual Level: Concept Definition Server
Language Level: Lexon Base
Commitment Layer
Applications
MDA
Platform Independent Models (PIM)
Transformer
Platform Specific Models (PSM) (J2ee, .NET, ...)

Figure 2.10: The remarkable analogies between the OMGs MDA approach and STARLab's DOGMA approach to ontology engineering.

Besides the now very popular RDFS [13] and OWL [23] languages from the Semantic Web, there is a considerable body of legacy expressed in UML and the Entity-Relationship (ER) model, and an active ISO standard process Topic Maps (TM) [2]. However, none of these metamodels support a fully declarative first-order predicate calculus language for expressing predicates. Therefore, the ODM includes a metamodel for Simple Common Logic (SCL) [36] for this purpose. The modular structure of MOF makes it straightforward for third parties to extend and enhance the metamodel.

To avoid an n-squared set of mappings, the ODM includes a further metamodel, a weakly-constrained abstract formulation of Description Logic (DL), to be the target of bi-directional mappings from the other metamodels. To map a legacy application from say UML to say OWL, one would first map it to DL then from DL to OWL. The DL metamodel is not intended to be used for ontology development in its own right.

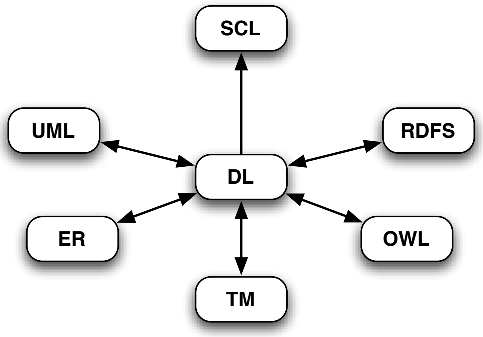


Figure 2.11: The structure of the ODM

SCL is an exception to this strategy. SCL is much more expressive than the other metamodels, so could be used to represent ontologies, but is therefore much more difficult to map into the other metamodels. The ODM intends SCL to be used to implement predicates which cannot be expressed in the other less-expressive metamodels. It is intended that a predicate be specified in a primary metamodel, say OWL, and implemented in SCL. The relevant elements of the M1 model expressed in the primary metamodel will be mapped into SCL. So only a uni-directional mapping from DL to SCL will be included. This mapping could also be used to migrate an application from one of the other metamodels to SCL for further development. The ODM does not provide a mechanism to get back from SCL to the other metamodels. Figure 2.11 depicts the structure and possible mappings of the ODM.

2.4 Semantic Integration

As we have discussed in the previous sections, ontologies have gained popularity as means for establishing explicit formal vocabulary to share between applications. Therefore, one can say that one of the goals of using ontologies is not to have the problem of heterogeneity at all. It is of course unrealistic to hope that there will be an agreement on one or even a small set of ontologies. While having some common ground either within an application area, a sector of organisations, or for some high-level general concepts could alleviate the problem of semantic heterogeneity, we will still need to map between ontologies, or heterogeneous semantic models in general. Not only do we need to map between different ontologies, we also need to map these ontologies to other models and applications, as is for example being done in OMG's Model Driven Architecture and by STARLab's DOGMA approach.

Semantic integration is the process of providing an integrated access to different, distinct and heterogeneous data sources that contain some form of semantics, implicitly or explicitly. For an overview of the different sorts of (implicit or explicit) semantics, we refer to our previous discussion in section 2. In [67], Wache et. al. have identified three possible scenario's in which ontologies can help integration heterogeneous information systems. Figure 2.12 shows the three scenarios.

The first scenario is the **single ontology approach** where one global ontology provides a shared model for the specification of the semantics. In this scenario, each object of each source must be related to the global domain model. This process is equal to the commitment of an application to an ontology in the DOGMA approach: It maps the objects from each source to the global ontology. This kind of mapping problem is one of the problems our platform addresses.

The first approach becomes very difficult if the granularity of the domains of each source is significantly different (we will discuss the different levels of granularity of ontologies later). This brings us to the second scenario: using **multiple ontologies**. In this approach, each information source is described by its own ontology. To relate the different objects for each source, the user needs to map the local ontologies. Here too is the problem of semantic integration which our platform tries to solve.

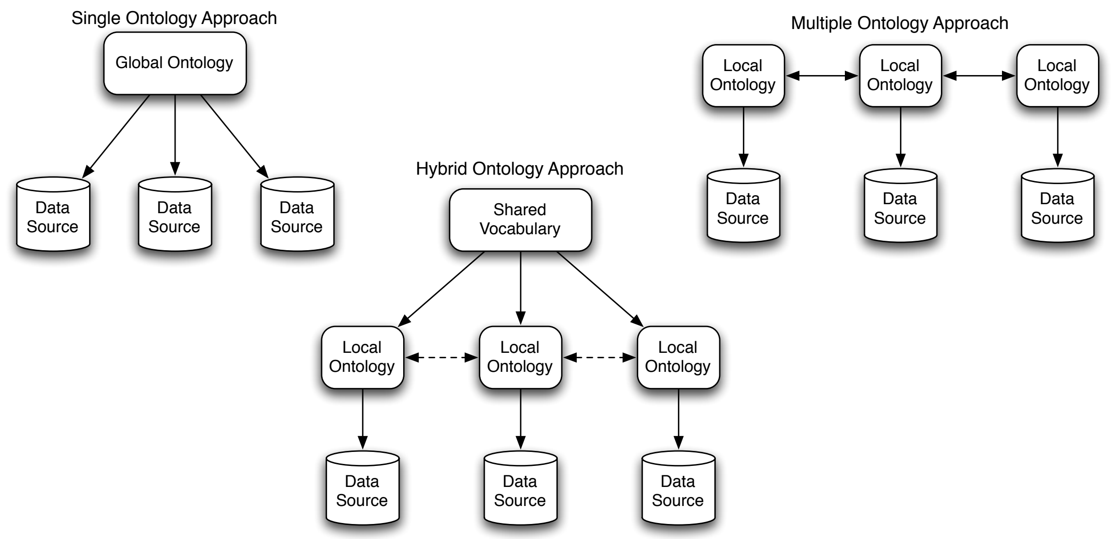


Figure 2.12: Three possible approaches for integrating heterogeneous sources using ontologies.

To overcome the drawbacks of the single or multiple ontology approaches, **hybrid approaches** were developed. Similar to the multiple ontology approaches, the semantics of each source is described by its own ontology, but in order to make the source ontologies comparable to each other, they are built upon one global shared ontology. This approach is often described as the Upper Ontology. One example of an upper ontology proposal is the Suggested Upper Merged Ontology (SUMO) [51].

Now that we have seen one of the many scenarios and problems for semantic integration, we will discuss the background on semantic integration and give a brief state of the art in this section.

2.4.1 Terminology

Before continuing this chapter, it is important that we first provide a clear and unambiguous terminology concerning the different aspects of ontology integration. We have opted to reuse the integration terminology that is adopted the most often in the current literature. We have borrowed the term definitions from [12].

Mapping: a formal expression that states the semantic relation between two entities belonging to different ontologies. When this relation is oriented, this corresponds to a restriction of the usual mathematical meaning of mapping: a function.

Ontology Alignment: a set of correspondences between two or more (in case of multi-alignment) ontologies. These correspondences are expressed as mappings.

Ontology Coordination: broadest term that applies whenever knowledge from two or more ontologies must be used at the same time in a meaningful way (e.g. to achieve a single goal).

Ontology Transformation: a general term for referring to any process which leads to a new ontology o' from an ontology o by using a transformation function t.

Ontology Merging: the creation of a new ontology o_m from two (possibly overlapping) source ontologies o' and o''. This concept is closely related to that of *integration* in the database community.

Ontology Reconciliation: a process that harmonizes the content of two (or more) ontologies, typically requiring changes on one of the two sides or even on both sides.

Meaning Negotiation: the protocol through which two agents (either human or artificial) agree on the changes required to reconciliate their ontologies.

2.4.2 Semantic Heterogeneity

In the distributed and open systems that is IT today, heterogeneity can not be avoided. Different actors have different interests and habits, use different tools, and use knowledge at different levels of detail. These various reasons for heterogeneity lead to different forms of heterogeneity that are considered below.

Heterogeneity may occur at many different levels, and a detailed list of all the forms of possible mismatches is beyond the scope of this document. We refer to [31, 7, 39, 37] for a more complete discussion on this matter. However, for the sake of the definition of our conceptual framework, we provide a classification into four main levels, based on [12].

The syntactic level

At the syntactic level, we encounter all forms of heterogeneity that depend on the choice of the representation format. Indeed, the semantic models come in several formats (UML, RDF, OWL, DB-Schema, ...), and each of them is based on a different syntax. In this thesis, we are not strongly concerned about the syntactic level. In general, transformations on the syntactic level are well understood in the computer science domain (e.g. XSLT).

The terminological level

The terminological level addresses all forms of mismatches that are related to the process of naming the entities (e.g. individuals, classes, properties, relations) that occur in models. Naming is the process of associating a linguistic object from a public language (or vocabulary – the one that is used to exchange information with other parties) to entities described in the ontology. In section 2.1.1 we have addressed how communities come up with new vocabularies, namely by (i) overloading common words, (ii) creating new words, (iii) non-word identifiers and (iv) compounds. This results in typical terminological mismatches:

- different words are used to name the same entity: synonyms;
- the same word is used to name different entities: polysemy and homonyms;
- words from different languages are used to name entities;
- syntactical variations of the same word (different spelling, abbreviations, prefixes and suffixes, etc.).

These terminological mismatches may occur in situations where the models or entities are conceptually equivalent. While mismatches at the terminological level are not as deep as those occurring at the conceptual level (see below), they are very common and real in most cases and therefore this level is just as important as the other one.

The conceptual level

At the conceptual level, we encounter mismatches that have to do with the content of the models. The practical forms in which metaphysical differences can rise are countless. However, following the artificial intelligence literature on this topic (in particular [7]), we follow the suggestion by [12] and visualised by figure 2.13 to cluster them in three abstract types:

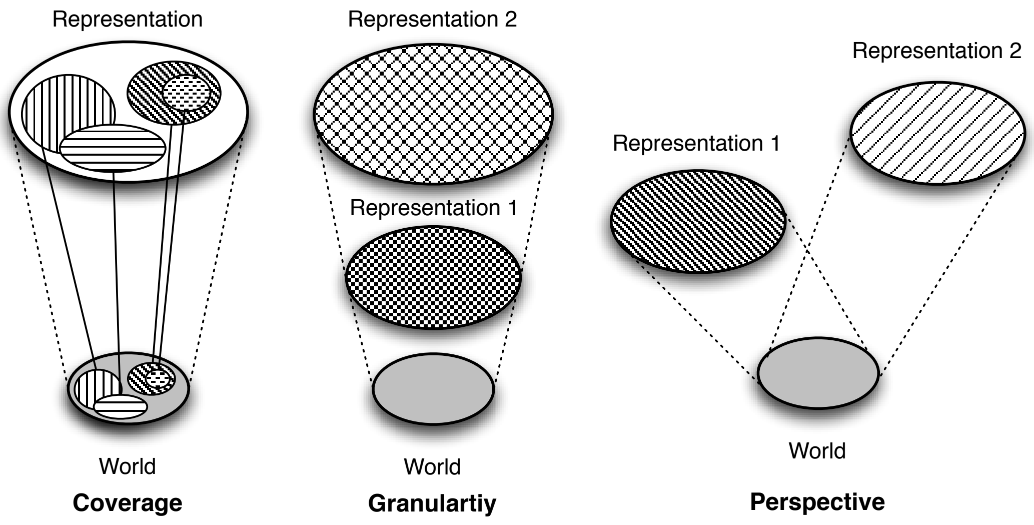


Figure 2.13: The three dimensions of heterogeneity at the conceptual level, reproduced from [12]

Coverage : an ontology may differ from another as they cover different portions - possibly overlapping - of the world or domain. For example, an ontology on planes may contain properties of fighter jets while another ontology might only cover passenger planes.

Granularity : an ontology may differ from another as the first provides a more (or less) detailed description of the same entities. For example, an ontology concerned with accounting and taxes, or delivery, would only consider the generic concept of document, while an ontology for libraries or scholars would distinguish between types of documents.

Perspective : an ontology may provide a viewpoint on some domain which is different from the viewpoint adopted in another ontology. For example, two ontologies may represent the same domain at the same level of coverage and granularity, but as different points in time (which means that the same property can hold at the time when the first ontology was designed and do not hold at the time when the other was designed.

2.4.3 Ontology Matching

Schema matching has become a much debated topic in today's research agenda, especially with the advent of the Semantic Web. Its roots, however, can be found in the database literature of the eighties. Due to the widespread importance of integration and matching, many disparate communities have tackled this problem.

In a nice survey of the different matching approaches up to 2001, Rahm et. al. [59] have introduced the *Match* operator. They define the match operator as a function that takes two schemas (models) S1 and S2 as input and returns a mapping between those two shemas as output, called the *match result*. We have depicted our interpretation of the Match operator in figure 2.14.

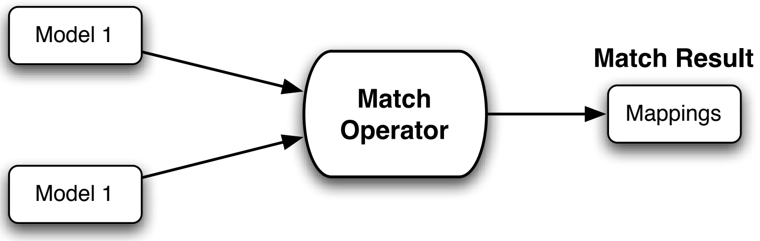


Figure 2.14: The Match Operator

Many different matching approaches have been proposed by several different communities. While the actual matching process is not the goal of our integration platform, it is a critical part of the full ontology integration process. Therefore, we present here a typology and classification of the most popular matching approaches and algorithms.

In a first classification of matching approaches, we present a typology of the different matching techniques. A second classification is based on the observation that a common trend in semantic integration is to progress from semantically-poor to semantically-rich solutions ([38]). This classification, described in [38], ranks the different techniques along a *semantic intensity spectrum*.

Typology of Matching Techniques

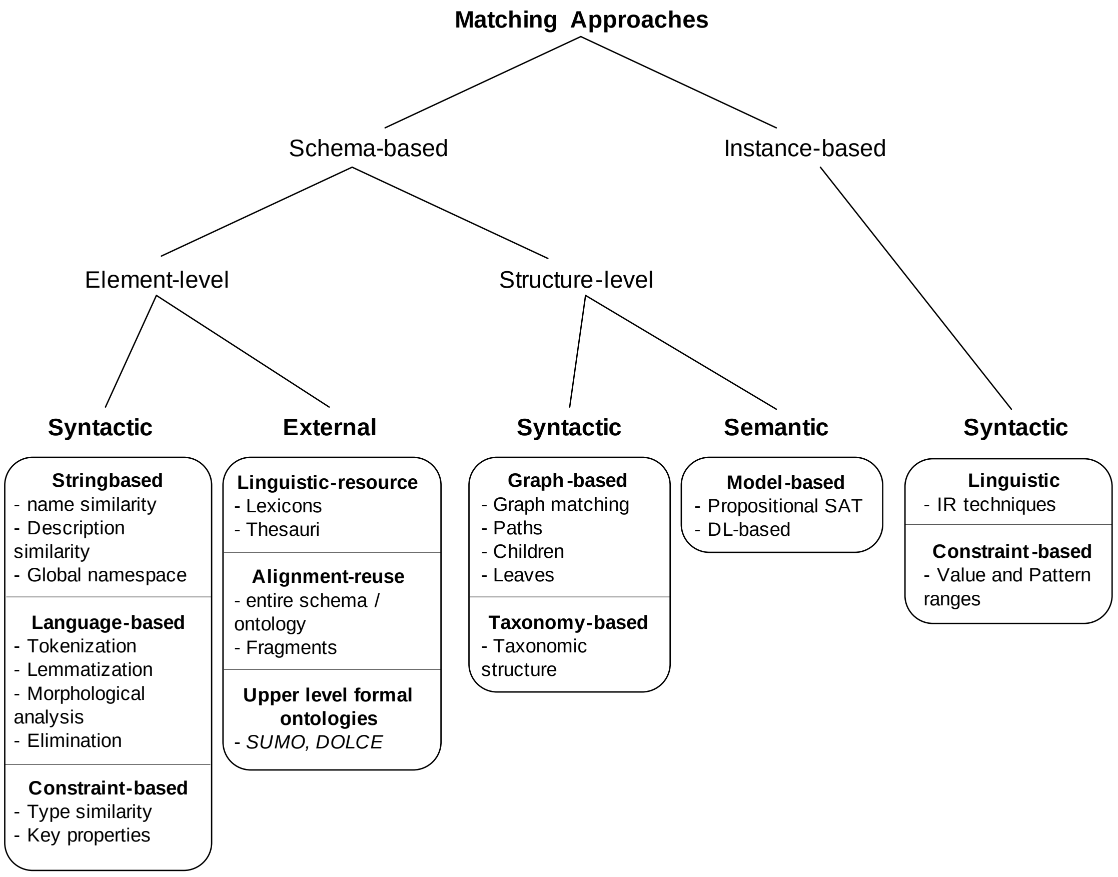


Figure 2.15: Matching typology, based on [59] and [62]

As we described earlier, most of the schema and ontology matching techniques are mutually beneficial. Therefore, it should be no surprise that classifications of schema matching techniques, which are targeted on schema matching approaches, have been used and revised for ontology-based matching systems. A well-cited classification that targets database schema matching approaches is the typology that was proposed by Rahm and Bernstein [59]. The classification of Rahm and Bernstein distinguishes between *elementary* (individual) and *combinations* of matchers. Elementary matchers comprise *instance-based* and *schema-based*, *element-* and *structure-level*, *linguistic-* and *constrained-based* matching techniques. Also *cardinality* and *auxiliary information* (e.g., thesauri like WordNet [50], global schemas) are taken into account.

This typology has been used and revised by Shvaiko and Euzenat in [62]. In their revision, they have introduced two synthetic classifications: a *Granularity/Input Interpretation* classification is based on (i) the granularity of a match, i.e., element- or structure-level, and then (ii) on how the techniques generally interpret the input information. The second classification is the *Kind of Input* which is concerned with the type of input considered by a particular technique.

The resulting typology can be seen in Figure 2.15.

Element-level vs Structure level Two alternatives can be distinguished for the granularity of a match: element-level and structure-level matching. This criterium was first introduced in [59]: *"For each element of the first schema, element-level matching determines the matching elements in the second input schema. [...] Structure-level matching, on the other hand, refers to matching combinations of elements that appear together in a structure.*

Syntatic vs external vs semantic This criteria is the *Input Interpretation* classification proposed by Shvaiko and Euzenat in [62]: *"The key characteristics of the syntactic techniques is that they interpret the input in function of its sole structure following some clearly stated algorithm. External are the techniques exploiting auxiliary (external) resources of a domain and common knowledge in order to interpret the input. These resources might be human input or some thesaurus expressing the relationships between terms. The key characteristic of the semantic techniques is that they use some formal semantics (e.g. model-theoretic semantics) to interpret the input and justify their results. In case of semantic based matching system, exact algorithms are complete."*

Semantic Intensity Spectrum

Another interesting classification is described in [38]. This classification is based on the observation that a common trend for DB and AI semantic integration practitioners is to progress from semantically-poor to semantically-rich solutions. This metaphor of semantic richness is used to classify works from both communities along a *semantic intensity spectrum*. Along this spectrum are several interim points to address string similarity, structure, context, extension and intension awareness as different layers of semantic intensity (see Figure 2.16).

This classification is interesting for us as, in general, semantically-rich solutions are more complex then semantically-poor solutions.

We will now give an overview of the different interim points on the spectrum, based on [38]. We refer to the full text ([38]) for a complete and detailed discussion.

String similarity , occupying the semantically-poor end of the spectrum, compares names of elements from different semantic models. These techniques consider strings as sequences of letters in an alphabet. They are typically based on the following assumption: The more similar the strings of the concepts or nodes, that is, the syntactic similarity, the higher their semantic similarity. Usually, the string is first normalized, then token-based distance functions are used to map the pair of strings to a real number. Some examples of such

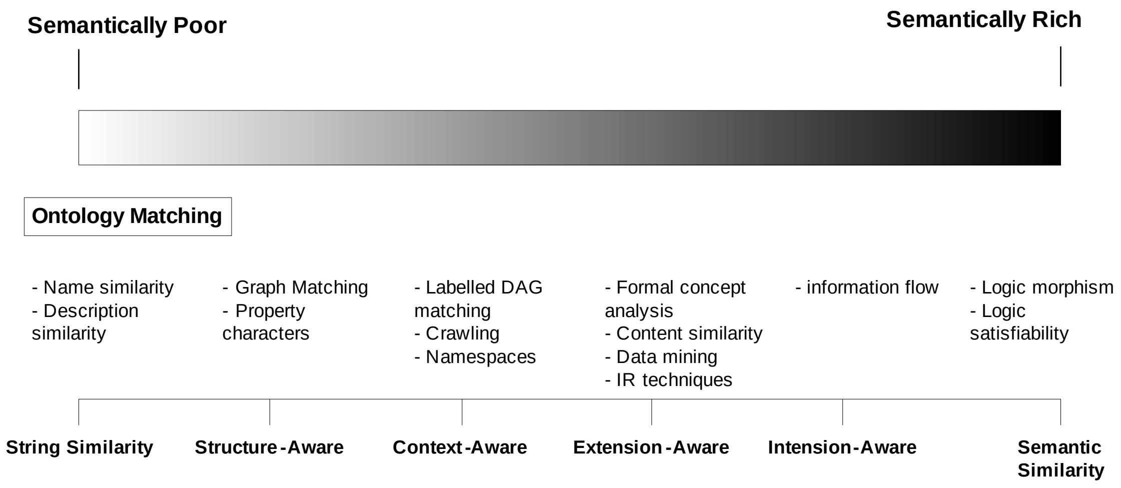


Figure 2.16: Semantic Intensity Spectrum, reproduced from [38]

techniques that are frequently used in matching systems are *prefix*, *suffix*, *edit distance* and *n-gram*.

Linguistic similarity , is just a little bit more to the right-end side of the semantic intensity spectrum then pure string-based similarity. In this case, names of concepts or nodes are considered as words of a natural language. Therefor, these techniques use Natural Language Processing (NLP) techniques. For example, instance, pronunciation and soundex are taken into account to enhance the similarity purely based on strings. Also, linguistic relations between words like synonyms and hypernyms (a word that is more generic than another word) will be considered based on generic and/or domain-specific thesauri, e.g. WordNet [50], Dublin Core.

Structure-aware , refers to approaches that take into account the structural layout of ontologies and schema data. Going beyond matching terms (strings), structural similarity considers the entire underlying structure. Although, in some interpretations, structure-level techniques include full graph matching algorithms, the interpretation of structure-aware techniques here, is the matching of ontologies that are represented as a hierarchical, partially ordered lattice. Therefor, in pure structural matching techniques, matching is equivalent to matching vertices of the two source graphs. Similarity between two such graphs G_1 and G_2 is computed by finding a subgraph of G_2 that is *isomorphic* to G_1 or vice versa.

Context-aware , in many cases, there is a variety of relations among concepts which makes it necessary to differentiate distinct types of connections among nodes. This gives rise to a family of matching techniques which are more semantically rich than *structure-aware* ones. In general, two different types of context-awareness can be identified.

In the simplest form, algorithms that compare nodes from two ontologies also traverse downwards several layers along the direction of edges from the node under considera-

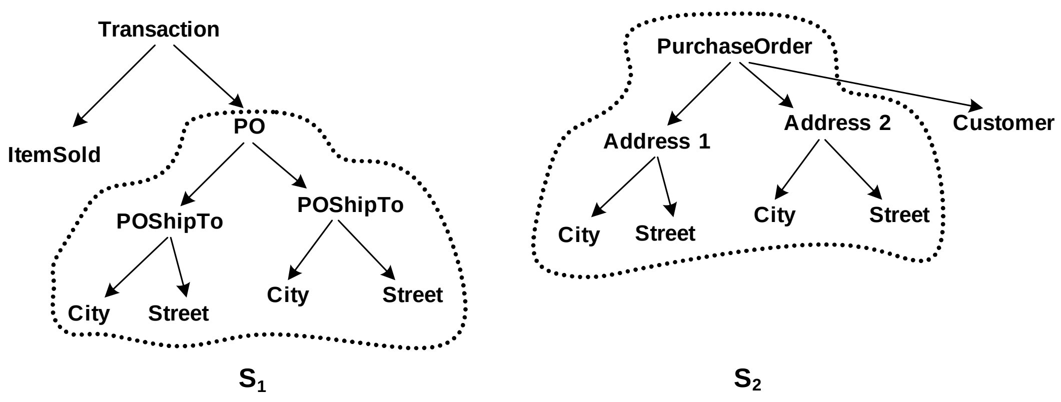


Figure 2.17: Structure Awareness

tion, or upwards against the direction of edges to the node under consideration. All the visited nodes, together with the information about edges connecting them (taxonomic relationships like part-of, subclass-of, is-a, etc) are evaluated as a whole to infer further mappings between nodes in the context.

Extension-aware , when a relatively complete set of instances can be obtained, the semantics of a schema or an ontology can be reflected through the way that instances are classified. A major assumption made by techniques belonging to this family is that instances with similar semantics might share features [45], therefor, an understanding of such common features can contribute to an approximate understanding of the semantics.

Formal Concept Analysis (FCA) [27] is a representative of instance-aware approaches. FCA is a field of mathematics emerged in the nineties that builds upon lattice theory and the work of Ganter and Wille on the mathematisation of concept in the eighties. A formal context is a triple $K = (O, P, S)$, where O is a set of objects, P is a set of attributes (or properties), and $S \subseteq OxP$ is a relation that connects each object o with the attributes satisfied by o.

The intent (the set of attributes belonging to an object) and the extent (set of objects having these attributes) are given formal definitions in [27]. A formal concept is a pair $< A, B >$ consisting of an extent $A \subseteq O$ and an intent $B \subseteq P$, and these concepts are hierarchically ordered by inclusion of their extents. This partial order induces a complete lattice, the concept lattice of the context. FCA can be applied to semi-structured domains to assist in the main structure most the mapping systems work with.

Intension-aware refers to the family of techniques that establish correlations between relations among extent and intent. Such approaches are particulary useful when it is impossible or impractical to obtain a complete set of instances to reflect the semantics. A mathematical theory that goes beyond *extension-awareness* towards the tick marked by *intension-awareness* is *Information Flow*, proposed by Barwise and Seligman. We refer to [6] for a detailed description.

Semantic Similarity , very close to the semantically-rich end lays the family of *logic satisfiability* approaches which focus on the logic correspondences. The idea behind techniques in this category is to reduce the matching problem to one that can be solved by resorting to logic satisfiability. Concepts in a hierarchical structure are transformed into well-formed logic formulae (wffs). These notions are also used in the DOGMA Framework: the lexons in DOGMA are all well-formed formulae.

2.4.4 Ontology Alignment & Merging

For our purposes, aligning two or more ontologies or models is a process that produces a set of mappings across entities which allow ontology coordination (see section 2.4.1 describing the terminology). Ontology merging, on the other hand, is a more specialised case which corresponds to the problem of generating an ontology $\mathcal{O}_m$ which mediates between two heterogeneous ontologies $\mathcal{O}$ and $\mathcal{O}'$. Ontology alignment & merging are only possible after the ontologies are matched (by a matching framework or by human actors). However, in most ontology integration approaches no explicit difference is made between the matching and alignment phase.

We back this claim by presenting the definition of the alignment process proposed in [12]:

Definition 7 (Alignment Process by Bouquet et. al. [12]) :
The alignment process can be seen as a function f which, from a pair of ontologies o and o′ to align, an input alignment A, a set of parameters p, a set of oracles and resources r, returns a new alignment A′ between these ontologies:

$$A' = f(o, o', A, p, r)$$

This definition can be represented as in figure 2.18.

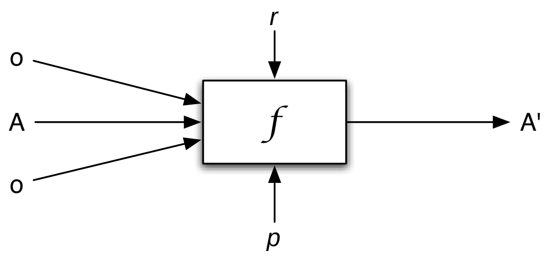


Figure 2.18: The alignment process

This definition and the figure shows that the alignment process by Bouquet et. al. is almost equal to the Match operator defined by Rahm et. al. As we will discuss in detail in chapter 3, our approach separates these two processes. We will also argue that this two-step methodology has many benefits to the existing single-step approach.

2.4.5 Existing Integration Frameworks

In light of our classification and typology of matching approaches, we discuss here a short overview of different matching systems that implement one or more of these algorithms. The big difference with these systems and our mapping platform, is that the systems below actually compute the mapping using a combination of matching algorithms. The goal of our platform on the other hand, is to store language independent mappings (which might come from the systems described here) and put them within a certain context and community.

This is a brief overview of the best-known matching systems, we refer to [62] for a more in-depth overview and description:

Cupid Cupid [44] implements a hybrid matching algorithm comprising linguistic and structural schema matching techniques, and computes similarity coefficients with the assistance of a domain specific thesauri.

COMA COMA (COmbination of MAtching algorithms) [24] is a composite schema matching tool. It provides an extensible library of matching algorithms; a framework for combining the obtained results, and a platform for the evaluation of the effectiveness of the different matchers. The features that set COMA apart from CUPID are a more flexible architecture and a possibility of performing iterations in the matching process.

Anchor-PROMPT Anchor-PROMPT [52] is an ontology merging and alignment tool with a sophisticated prompt mechanism for possible matching terms. In this respect, this system is more likely to ours compared to the other matching systems discussed here, in that it allows human suggestions and feedback. The other systems here are completely automatic.

S-Match S-Match [29, 30] is a schema-based matching system. It takes two graph-like structures and returns semantic relations (e.g. XML schemas or ontologies) and returns semantic relations (eg. equivalence, subsumption) between the nodes of the graphs that correspond semantically to each other.

3 Approach & Methodology

In this chapter, we will discuss our approach and methodology towards semantic integration. In the first section, we will briefly summarise, from the previous chapter, the need for semantic integration and why the existing approaches are not sufficient. In the methodology section we discuss and elaborate on our specific, multi-step approach to semantic integration. In the following section we describe in more detail our approach. More specifically, we discuss the uniformness of our proposed mapping platform, its context-awareness and its support for the influences of different communities. Furthermore, we provide an overview of the specific semantics of our integration platform. We elaborate on the specific meaning of a community, a context, a mapping, etc. within our framework by providing the reader with a set of definitions. In the last section of this chapter we provide the reader with the motivation for our choices. We provide a number of significant advantages of our approach compared to existing platforms and we discuss how our proposed methodology achieves the goals for this thesis.

3.1 Introduction

As we have discussed in detail in section 2.4 on semantic integration, integrating two heterogeneous models corresponds to finding a set of mappings between the entities of the first model and the entities of the second model that enable the first model to be translated into the other, and/or the other way around. In the current literature, the integration process of finding a final set of mappings between two models is considered one process within a well-defined time span. If we recap from the background chapter: Some authors use the notion of *Match Operator* to describe the mapping process (e.g. Rahm et. al. [59]) while another group of authors talks about an alignment process (e.g. Bouquet et. al. [12]). Both approaches are depicted next to each other in Figure 3.1.

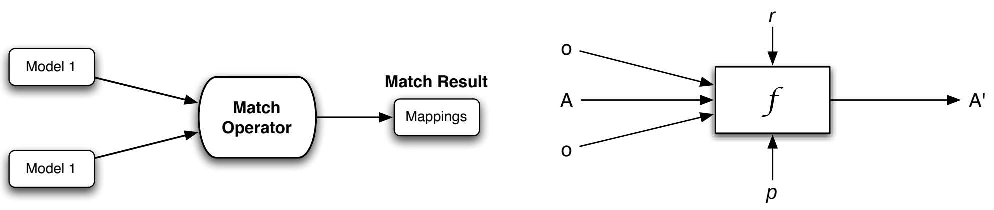


Figure 3.1: The Match Operator by [59] *(left)* and the Alignment Process by [12] *(right)*

The *Match Operator* receives 2 models as input and uses a set of matching algorithms to compute the similarity between the entities of both models. As a result, the match operator outputs a list of mappings between the entities of both models.

Similarly, the *Alignment Process* receives two models as input and an alignment (initially this might be an empty alignment or an instance training set). The output is an updated alignment, which exists of a set of mappings between the entities of both models. Because the alignment process also receives an alignment as input, these alignment processes can be chained.

Both approaches show a one-step process (or several chained one-step processes) that do not have a persistent state between the start and the end of the process. We argue that these approaches come with several drawbacks as we will show in the following sections.

3.1.1 Mapping Reuse

The idea behind this thesis is to propose a conceptual framework to allow efficient and scalable mapping reuse. This is one of the promising directions in semantic integration. A rational behind mapping and alignment reuse is that many ontologies or models to be matched are similar to already matched ontologies and models, especially if they are describing the same application domain [59, 62]. Once an alignment has been determined, it can be saved, and further reused. Thus, a (large) repository of mappings has a potential to increase the effectiveness of matching systems by providing yet another source of domain specific knowledge. Knowledge is contained within mappings between heterogeneous data sources. It is important that this knowledge can be shared, governed and agreed upon by the community of interest. Just like ontologies, which are **shared** conceptualisations, we argue that mappings should just as well represent the shared vision and domain knowledge of a community. Enabling actors to establish and reuse mappings can, especially within their specific communities, create an adequate and timely domain representation, facilitate knowledge exchange, etc.

Most of the existing integration frameworks do not explicitly support mapping reuse. As we have presented the advantages of having a shared and agreed upon set of mappings, the limitations and drawbacks of integration platforms and framework that lack such an approach are evident: Currently, a mapping is the interpretation of a single individual for a certain integration problem. There are no processes in place for the mapping to represent the interpretation of a group of individuals that are affected by the particular integration problem. There is no way the community of interest can represent its confidence or thrust in the mappings. These are very important issues that are overlooked in the current integration frameworks. Furthermore, as mappings are not shared, every individual must redo the entire mapping process for integration scenarios that may be very similar to already solved problems. Because of these limitations, the integration problem is still a very labour-intensive problem that does not scale well. As it happens, the efficiency and scalability of mapping frameworks will become ever more important while we are moving more and more towards model-driven architectures and methodologies in a lot of application areas of computer science and Information Technology, as we have shown in the Semantic Web and OMGs Model Driven Architecture initiatives.

One framework, COMA++ [5] does support mapping reuse, but only privately: Access to the system is limited to individual users, who usually do not know each other, hence, they do not communicate with each other. While this private reuse solves some issues on the practical side of the integration problem, it does not address our proposition that the mappings should represent the shared knowledge of the community of interest. In this private reuse of mappings approach, shared knowledge of the community with its negotiation processes can not be fully leveraged.

3.1.2 Application-specific mappings

Not only does the current work on semantic integration not account for shared mapping reuse, it does also make no distinction between the conceptual meaning of a mapping and its application specific representation. Using the definition of semantic integration, mappings must allow one model to be translated into the other. To allow models to be translated into each other, the mappings between the entities of these models need to be described in a specific integration language or integration model that can be interpreted and executed by a specific application for a specific purpose. We give a few examples of how this issue can present itself:

Map an SQL schema to an ontology In this scenario, we want to map an SQL schema from a relation database to an ontology. Using this mapping, we want to be able to query the ontology for the content of the relational database.

Let's say that there are two applications that can translate queries on the concepts and properties from the ontology to standard SQL queries. To do that, both programs need a mapping file that precisely specifies how the ontology is translated to relation tables and columns. However, both these applications are developed by distinct software vendors and use their own proprietary language to describe these mappings. Currently, there is no integration framework that enables the actors to save the mappings from the ontology to the relational tables in an application-neutral format. This clearly shows the need to separate the conceptual meaning of the mapping (matching a relation table to a concept from the ontology for example) from the application- or language-specific representation of these mappings.

Translate instance data from one model into another model In this scenario, the actors need to translate instance data that is represented in one model into another given model. This is a very well known problem in the enterprise-IT market. One example in which this scenario often occurs is data warehousing. A data warehouse is a decision support database that is extracted from a set of data sources. The extraction process requires transforming data from the source format into the warehouse format. As shown in [54], the match operator is useful for designing transformations. However, these transformations again have to be interpreted and executed. For example, a *concatenate* transformation needs to be executed when the instance data is actually transformed into the other model, in this case the data warehouse.

Another drawback of directly working on language- or application-specific mappings is that these mappings can only be created by an actor that is familiar with the language or application. However, mappings between different models often require a more high-level and conceptual view of the data and models. The information that is needed to create valid mappings is often contained by users with a managerial role in the organisation while these users often have no knowledge of the specific, low level integration language used.

While these limitations are not so serious on integration platforms that don't support mapping reuse, on our platform, where we aim to provide the actors with a platform on which they can establish their shared and reusable mappings, this constraint is much more severe: To be able to establish such a shared knowledge, the involved actors must be able to communicate in a manner that all of them understand. Therefore, it is important that the shared mappings are expressed in a conceptual manner which also non-technical knowledge experts understand.

3.2 Methodology

As we have discussed in the previous section, semantic integration is traditionally a one-step process that, given two heterogeneous models, returns a set of mappings that can translate one model into the other. We have also argued that this approach comes with a number of drawbacks such as dealing with application-specific mappings, no conceptual framework to reason, discuss and approve the mappings, hard to reuse existing mappings, etc.

In this thesis we propose a possible solution for these problems: We introduce an integration approach that splits up the integration process into two distinct phases: the *Mapping Phase* and the *Commitment Phase*. In short, the mapping phase is similar to the Math Operator or Alignment Process in that it takes two heterogeneous models as input and returns a set of mappings between the entities of both models. The difference lies in the fact that these mappings are language-, application and paradigm-neutral or in other words, **uniform**. In the second step, the commitment phase, the actors select a meaningful subset from the large number of conceptual mappings. The mappings from this subset are context-, community-, and application-dependent and are represented in a language- or application-specific format.

Our two-step approach has been inspired by both the STARLab Dogma approach to ontology engineering and OMGs Model Driven Architecture paradigm. A very high-level comparison between the three is depicted in figure 3.2.

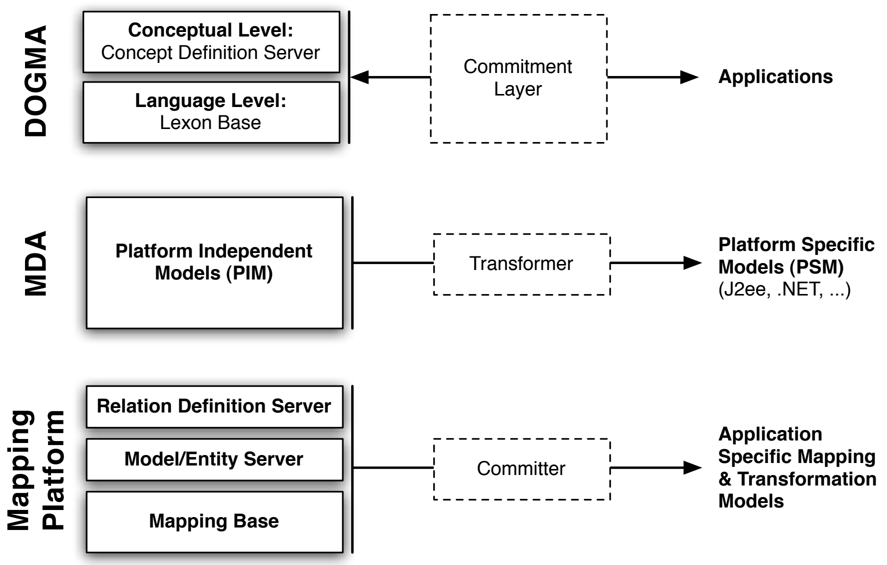

DOGMA
Conceptual Level: Concept Definition Server
Language Level: Lexon Base
Commitment Layer
Applications
MDA
Platform Independent Models (PIM)
Transformer
Platform Specific Models (PSM) (J2ee, .NET, ...)
Mapping Platform
Relation Definition Server
Model/Entity Server
Mapping Base
Committer
Application Specific Mapping & Transformation Models

Figure 3.2: A high-level view, from top to bottom, of Dogma's ontology engineering approach, OMGs model driven architecture paradigm, and on the bottom, our approach to semantic integration, our mapping platform.

In Dogma's approach to ontology engineering, the commitment layer is a separate layer, mediating between the lexon base and the applications that commit to the lexon base. Committing to the lexon base means selecting a meaningful set of lexons from the lexon base that approximates well the intended conceptualisation with respect to the application domain.

In the OMGs Model Driven Architecture paradigm, the envisioned applications are created using platform independent models, using for example UML, OCL, etc.. In a second phase, these platform independent models are transformed into platform specific models that deploy the application on a specific platform in a specific language, for example in Java J2EE or Microsoft .NET.

Similarly, on our mapping platform, application or platform independent mappings are created in the mapping phase. We call these mappings *plausible mappings* as they might be valid in a certain context and community, but not necessarily in another. In the second phase, a meaningful subset is chosen from this large set of conceptual, plausible mappings depending on the context, the involved community, and the application domain of the integration scenario in the commitment phase. We call this process ”committing” . For this process, platform or language-specific committers must be created that will help the actor to interpret the conceptual mappings and translate them to a language-specific, executable alignment. Next, we will go into more detail on both steps of our integration methodology.

3.2.1 Mapping phase

The goal of the mapping phase is to identify plausible mappings between entities from the heterogeneous models. The result of this process is a set of uniform, plausible mappings that are added to a large mapping store. This process is split up into two methodological steps:

1. Preintegration: In this step, the syntactical heterogeneities of the models are overcome by parsing the entities of both models into our uniform mapping data model. We will discuss our mapping data model in the coming sections of this thesis.
2. Matching: In the second step, the different entities are matched and mappings are created between similar entities. This matching process can be completely automated by using a set of matching algorithms we reviewed in our first chapter, but more realistically, matching algorithms will create a number of mappings after which a human actor or actors should validate these. The resulting mappings are stored in a large mapping repository.

Each mapping process is executed within a certain context and is executed by a certain community. Therefore, each mapping stored in the mapping store has a context and community identifier attached to it.

The context of the mapping models the environmental settings in which the mapping has been deducted. A mapping might be plausible in a certain context, while the same mapping might not be valid in another. The context also enables the actor to provide a well-defined and context-specific definition of the relation of the mapping. For example, a relation between two entities in one context might be transitive, while the same relation in another context might not be.

The community of a mapping contains the actors, processes and goals of the actors that have created the mapping. Again, a mapping might be relevant to one community but not to another. Furthermore, reuse of mappings created by different actors implies resolving, among others, such challenges as the *appropriateness* of mappings when using them in applications and *thrust* issues. We will review the context and community aspects of our integration platform in more detail in the coming sections.

The mappings that are created in the mapping phase should be viewed as conceptual mappings, that is, they are not meant to be used in applications, but should be used by the communities of interest to negotiate and share the implicit knowledge that they contain: How does one model or ontology relate to another within a certain given context through the eyes of a certain community. Because each mapping has a context and community aspect, these can be fully leveraged in the next phase, the commitment phase, of the integration process. To conclude: The result of the mapping process is a set of uniform, conceptual mappings with a well-defined relation that are plausible in a certain context and are created by a certain community.

3.2.2 Commitment phase

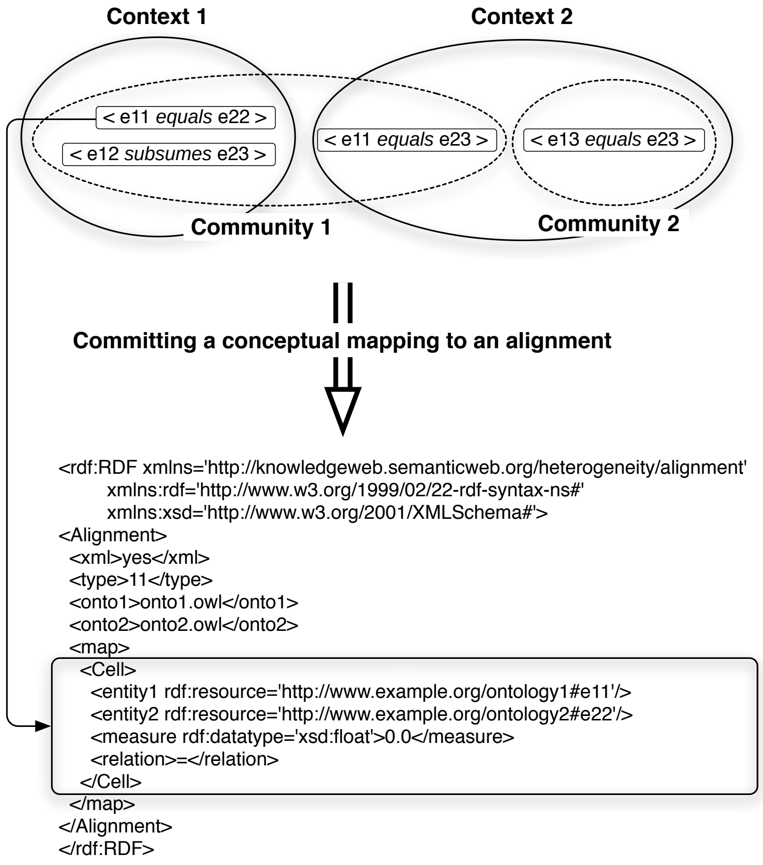
The goal of the commitment phase is to create an application-specific alignment that best approximates the intended use of the alignment. The alignment is modelled in an alignment model described in a certain language to be used by a specific application.

During the commitment process a (group of) actor(s) selects a meaningful set of mappings that best approximates their intended alignment. This selection and reuse process of committing a mapping to an alignment corresponds to augmenting the representation-neutral mapping element to a commitment rule that is valid in this particular alignment described in a particular language. The validity of the commitment-rule is dependent on the intended use of the alignment, which will be domain-, community-, organisation-, or application-specific. Hence, a commitment or alignment can be seen as an interpretation of a set of mappings.

The resulting alignment corresponds to an instance of an alignment model which is described in a specific language. For example, when integrating two OWL-ontologies, the alignment may be an instance of an OWL Alignment Ontology which provides a language-specific interpretation of the mappings from the alignment. Figure 3.3 depicts the process of committing a plausible, conceptual mapping from the mapping phase to an instance of the Aligment API integration model[1] [26]. The resulting alignment is language specific and should be created with a certain application scenario in mind. It is also the interpretation of the conceptual mapping from which it originated. In the example, the conceptual relation of the mapping, *equals*, is interpreted to the language specific relation =. When committing to other formats, the conceptual relation may be interpreted to rules in a certain rule language (for example, RuleML [11], SWRL [35], etc.) or it may interpreted as an *owl:SameAs* relation, etc.

By adopting this clear separation and distinction between the matching and alignment phase, the actors of the system can reuse the existing mappings to create an alignment that best approximates its intended use - which will be domain-, community-, organisation-, or application-specific. We will further elaborate on the advantages of our methodology after we have fully presented our methodology and the semantics of our platform in the following sections.

[1]The Alignment API is a format for expressing alignments in RDF, so that they can be published on the web.



```
<rdf:RDF xmlns='http://knowledgeweb.semanticweb.org/heterogeneity/alignment'
        xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
        xmlns:xsd='http://www.w3.org/2001/XMLSchema#'>
<Alignment>
  <xml>yes</xml>
  <type>11</type>
  <onto1>onto1.owl</onto1>
  <onto2>onto2.owl</onto2>
  <map>
    <Cell>
      <entity1 rdf:resource='http://www.example.org/ontology1#e11'/>
      <entity2 rdf:resource='http://www.example.org/ontology2#e22'/>
      <measure rdf:datatype='xsd:float'>0.0</measure>
      <relation>=</relation>
    </Cell>
  </map>
</Alignment>
</rdf:RDF>
```

Figure 3.3: An example of committing a conceptual mapping to an application specific alignment. In this example, the alignment is an instance of the Alignment API [26] model.

3.3 Mapping Approach

In the previous section, we have reviewed our two-step methodology for semantic integration. Separating the mapping phase from the commitment/alignment phase allows us to introduce other important aspects that have not yet, or only briefly, been addressed in the current literature with respect to semantic integration. We introduce here the tree aspects that make our methodology different from existing approaches: We support uniform, language-neutral, conceptual mappings between models without any constraints on the type of models and their entities. We also introduce the notion of context into our integration approach. We argue that mappings are dependent on the context in which they are created or elicited. Furthermore, we discuss how communities can influence the integration process. We will elaborate on how communities can play an important role in the alignment phase of the integration process. We will also discuss the possibilities that communities bring to the integration process in the context of groupware: how can they help to introduce measures of quality, confidence and thrust in a mapping, and how can a group of actors participate and collaborate to create a larger, richer and better set of high quality mappings.

In the following sections, we will elaborate on each of these three aspects that makes our approach unique.

3.3.1 Uniform Mappings

As we have discussed in section 2.4.2 on Semantic Heterogeneity, there are several forms of heterogeneity that must be overcome to achieve semantic integration: Heterogeneity occurs on the *syntactical level*, on the *terminology level*, and on the *conceptual level*.

One of the goals of this thesis is to propose a conceptual platform on which actors and communities can create, maintain, reason and govern (express their thrust and confidence) about mappings. As we have discussed earlier, to reach this goal, it is required that the mappings are created and stored in a paradigm and application-neutral form. Therefore, we make no assumptions on the possible scenarios, platforms and applications on which these mappings can be used (that's why we have the separate commitment/alignment phase). Hence, it is very important that we also make no assumptions on the type of information sources that need to be integrated, in other words, we must support syntactic heterogeneity on the input side (the models that need to be integrated) as well as on the output side (the alignments between the models resulting from the commitment phase).

The requirements and proposed solution of our integration approach asks for a complete support of syntactical heterogeneity. We call this aspect of our integration platform the support for *Uniform Mappings*. Under uniform mappings we understand the following:

- Allow the creation of mappings between any two different types of models. For example, the platform must support mappings between relational databases and ontologies, but also between UML diagrams and ontologies and of course between ontologies themselves.
- The mappings must be created and stored on a conceptual level, that is, they should only refer to other entities or definitions. We will elaborate on this in more detail when we discuss the semantics of our integration platform.
- The mappings are never used in the form they are created and stored. To use a mapping in a real application or scenario, they must be interpreted and transformed into a representation that is specific for their intended use. Any representation format should be supported. A mapping between concepts from two different ontologies could be committed into different integration models. For example, a mapping with a conceptual *equal* relation could be interpreted as a *owl:SameAs* relation in one commitment, while the same mapping could be interpreted as a complex rule in a certain rule language (such as RuleML [11], RIDL [48], SWRL [35], etc.) in another commitment for another application and scenario.

Model	Entities
Relational Database Table	Columns
OWL Ontology	OWL Classes, properties & relations
UML Diagram	UML Classes & properties

Table 3.1: A number of possible model / entity combinations

To support these uniform mappings, we parse any input model into our own Model - Entity format. A model is basically a set of entities that can be logically grouped together. One or more entities from a first model can be mapped to one or more entities from a second model. No syntactic or semantic constraints are put upon these entities. Table 3.1 lists some possible model/entity combinations:

3.3.2 Community-driven Mappings

By a community we mean here a group of individuals that have a common interest and share a certain goal. They often maintain their own communication and collaboration environments through for example, web portals, thin- or thick-client applications, etc. Recent research has identified a high importance of direct involvement of humans and communities in ontology management: An actor (an agent or human contributor) was shown to be an indispensable part of a semantic network [49], and participation of a community construction has since long been shown as a way to a more complete and up-to-date domain knowledge representation [22, 68].

We argue that the interaction, collaboration and negotiation between actors and communities of actors is also very important in the process of integrating heterogeneous semantic models. As we have stated previously, in this thesis we propose a novel framework that encourages mapping reuse to come to a more efficient and scalable integration process. We also argue that in order to achieve such an efficient and scalable integration platform, actors should be able to reuse mappings that have been created by other actors and individuals. As we have discussed, current work only support private reuse of mappings. This is our first motivation for introducing actors grouped in communities: In order for actors to reuse mappings, they have to know from which actor these mappings originated. Furthermore, it is important that the user of the mappings knows by which community this mapping was created. This is an important fact because each community has a certain goal and process which might have influenced the elicitation or creation of the mapping. The established mapping might be valid or plausible to a certain community while another community might disagree. It is therefore necessary that when an actor reuses an existing mapping, he knows by which community this mapping was established. As a result, one could call these mappings *subjective mappings* in that they are appropriate in one community or domain, but not in another.

Figure 3.4 shows a possible mapping phase of the integration process. There are 2 actively par-

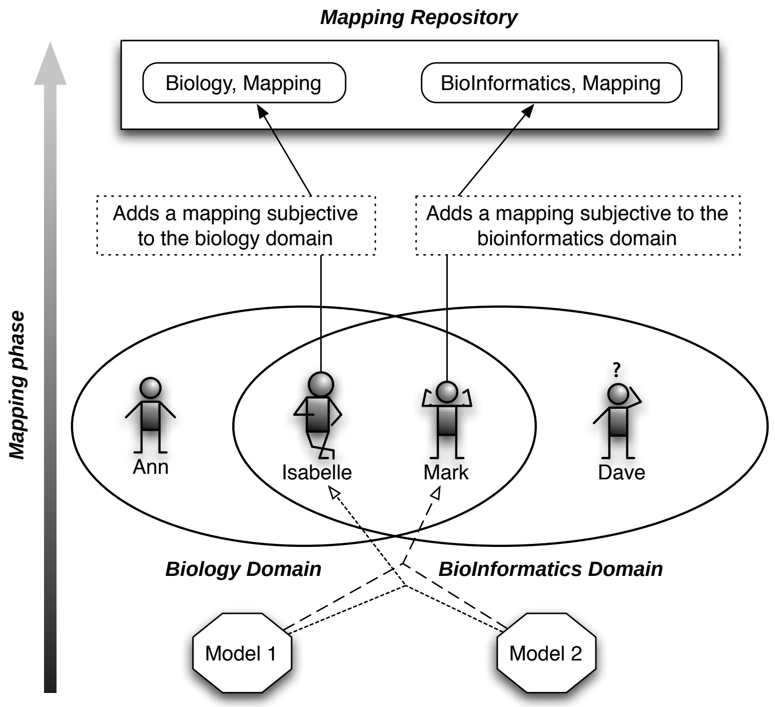


Figure 3.4: An example of a community-driven mapping elicitation and creation process.

ticipating actors in this example: Isabelle and Mark are individuals that are familiar with the biology domain as well as the BioInformatics domain. Both individuals are part of both communities. Both of them create a mapping between two similar entities from a first model from the biology domain and a second model from the bioinformatics domain. However, Isabelle creates the mapping for the community of biology researchers while Mark creates the mapping with respect to the Bioinformatics community. As a result, there are two different mappings stored in the mapping repository between the same entities from the same two models, but with a different community aspect and possibly with a different relation between the entities of the mapping.

However, introducing reuse of mappings created by different actors also implies resolving, among others, such challenges as the level of thrust and confidence that the users have in the existing mappings. Therefore, it is important that members of the community can express their confidence and thrust in mappings. These confidence measures should also be stored with the mapping. When these confidence measures are stored, users can see the mappings in which their peers have most confidence in and reuse these. This measure acts as a useful way to rate the relevant mappings that may be reused, it allows the actor to make a better choice of which mapping he should reuse. This is an important feature because when this would be unavailable, the user could get a very large number of potentially relevant mappings of which he would not know how to make a choice from.

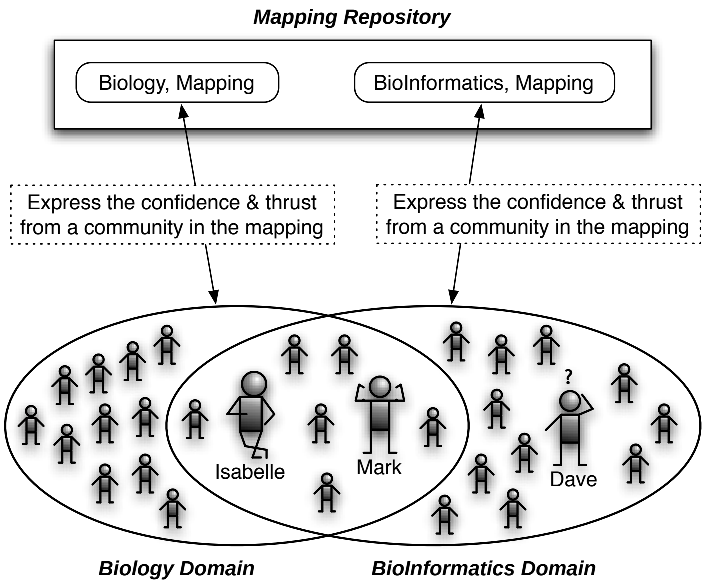

Mapping Repository
Biology, Mapping
BioInformatics, Mapping
Express the confidence & thrust from a community in the mapping
Express the confidence & thrust from a community in the mapping
Isabelle
Mark
Dave
?
Biology Domain
BioInformatics Domain

Figure 3.5: An example of a communication and negotiation process in which the community can express its confidence and thrust in the existing mappings.

To revisit the previous example, figure 3.5 shows the negotiation and communication process in which the community members can express their confidence and thrust in the mappings previously created by their peers. Because these mappings are stored in a conceptual manner and do contain any application-specific technicalities, the knowledge engineers and domain experts can efficiently review the quality of the mappings. When the quality is not sufficient, they can add their own mappings. As a result, when an individual want to reuse a mapping from his community, he will be able to see in which mappings the other actors from his community have most confidence in. When we discuss the context of a mapping, we will show how we can identify similar mappings (mapping between the same entities) from the same community.

In the second phase of our integration methodology, the commitment phase, the actor must make a selection of mappings to create an alignment that best approximates its intended use. In this phase, the actor will browse or search through the mapping repository to find the most relevant mappings. The community-driven integration platform brings two important advantages in this stage of the integration process:

1. Since the mappings are grouped by community, the actor can easily find the relevant mappings for his application. This is important because when the mapping repository grows larger and larger, finding the relevant mappings will prove to become more difficult over time. This is a recursive problem: When the actor is unable to find a relevant mapping, he might decide to create its own and add it to the mapping repository which will become larger. The next actor that needs to find a relevant mapping will have an even harder time to find the relevant mapping and might also decide to create his own mapping, etc.

2. Because the members of the different communities have been able to share their confidence and thrust in these mappings, the actor will be able to better find the best possible mapping to include in his alignment.

We give an example of this phase by revisiting our previous examples. Figure 3.6 depicts two possible commitment scenarios. In the first, Ann needs to build a system that allows the users of the system to query for articles about new crop growing techniques. There is a query system available that can query a large number of heterogeneous information sources by using a given alignment. The only thing that Ann has to do is to build this application-specific (in the format that is understood by this particular query system) alignment that maps between different sources about crop growing techniques (the intended use). As shown in this case, Ann needs a mapping (which we call *Mapping1*) between a two specific entities from two relevant models, one from the biology domain and one from the bioinformatics domain (as we have seen in the previous examples). As shown in the figure, the mapping repository has two mappings that could provide Ann with this particular mapping (called *Mapping1*). One of these mappings is created by someone from the BioInformatics domain (Mark) and the other mapping is created by someone from the Biology domain (Isabelle). While both mappings express the same actual mapping, Ann will most likely choose the one that has been created by one of her colleges from her own community as this mapping will most likely express the relation Ann intends to use it for.

Similarly, Dave needs to create an alignment that will allow the query engine to query for articles about AI-search problems. Mark needs the same mapping as Ann, but will in this case use the one that has been established by a member of his own community.

3.3.3 Context-aware Mappings

As we have discussed in our background chapter, context is a critical concept for applying semantics in practice. For example, in the elicitation and application of ontologies, the meaning of the ontological knowledge is dependent on the context [18]. While it is such an important aspect of semantic technologies, most of the current works on semantic integration do not mention context.

A possible reason for this is that modelling the context is a very difficult problem that has not yet been solved by the research community. Important work in that respect has been done by Lenat in the Cyc Ontology project [43]. Lenat categorises context into 12 dimensions: time, typeoftime, geolocation, typeofplace, culture, sophistication/security, topic, granularity, modality/disposition/epistemology, argument-preference, justification, and ”let’s”. However, in this thesis, we will model the context as a black box so it is up to the user to describe the context. This is the same method that is adopted by most of the current work that handles context. Context is very complex to precisely specify and the pragmatic solution we adopt in this thesis is sufficient for our needs.

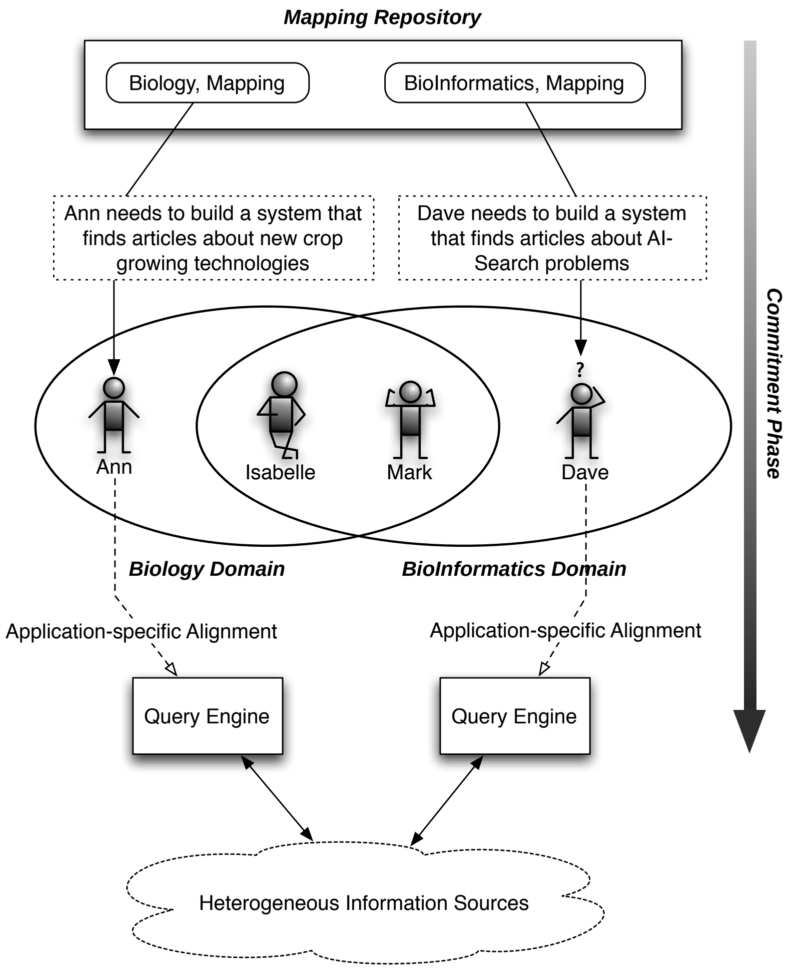

Mapping Repository
Biology, Mapping
BioInformatics, Mapping
Ann needs to build a system that finds articles about new crop growing technologies
Dave needs to build a system that finds articles about AI-Search problems
?
Ann
Isabelle
Mark
Dave
Biology Domain
BioInformatics Domain
Application-specific Alignment
Application-specific Alignment
Query Engine
Query Engine
Heterogeneous Information Sources
Commitment Phase

Figure 3.6: An example of a communication and negotiation process in which the community can express its confidence and thrust in the existing mappings.

The way we use the context of a mapping is very similar to the way as we use the community in our framework: It is an important grouping mechanism that allows the actor to easily retrieve the most relevant mapping to be used in the commitment phase. While the community is a very specific differentiator between mappings (we will give a precise definition of what a community looks like later), the context allows the actors to arbitrarily group mappings together. For example, the context can differentiate between mappings that have been created by a different set of matching algorithms, by a different number of users from the same community, at a different time, etc. Furthermore, there is no link between the context and the community: A community can contain mappings that belong to several contexts but a context can just as well contain mappings that belong to several communities as we will see in the examples. The combination of a context and community describes a unique set of mappings. We will elaborate on the exact semantics of our integration model in the next section.

Because the context is such a undefined, black-box, we think it is best to show its use through a couple of examples:

Context to disambiguate mappings: The first obvious use of context is to disambiguate between mappings. While we introduced the community-awereness of a mapping to specifically disambiguate on the creators of the mappings, we also need to disambiguate on other potential aspects of a mapping. This is particularly critical in our framework. Because the strength of our approach is directly linked with the ability to reuse plausible, conceptual mappings, it is important that enough of these relevant mappings exist. Therefore, we need a way to disambiguate similar mappings on more then only their creators so that enough mappings are available for different scenarios.

Context to disambiguate relations: In ontology management, the context is used to disambiguate similar terms that refer to different concepts. Context can be similarly used in ontology integration whereas a relation term in one context refers to a certain relation definition while the same relation term in another context can refer to another relation definition. For example, a relation in one context might meant to be transitive while the same relation in another context might not. It is important that our platforms provides a way to disambiguate these two relations by letting them refer to a different relation definition. We will discuss this in more detail when we present the semantics of our integration platform.

Context to support collaboration: To allow public reuse of mappings, the mappings must be stored in a repository which runs on a server accessible by different actors through different systems. However, such a collaboration environment introduces typical groupware problems: What to do if two actors edit the same mapping at the same time and store it back into the repository? The offline editing of mappings and storing them back in to the repository introduces a number of typical problems. One possible solution is to view the mappings from a database perspective and use several locking techniques. However, the scalability of this approach is questionable. Another possible solution is to build a merge-operator that merges different versions of a single mapping. A potential implementation of this solution could be to automatically create a new context when two mappings can not be merged. This could be extended to introducing dependencies between contexts and brings us to the concept of evolution. We will discuss this in more detail when we elaborate on the additional benefits of our approach.

3.4 Mapping Semantics

In this section we elaborate on the specific semantics underlying our methodology and platform. We will provide definitions of the different concepts we have introduced in the previous sections: the context and community of a mapping, the model and its entities, the conceptual relation of a mapping and of course the mappings themselves. It is important that we provide well defined descriptions of these concepts in respect to our specific integration approach as these concepts are used in many other scenarios and have been overloaded with different meanings over time.

Several different mapping models have been proposed in the current literature. A very formal and logic-based model is described in [12]. Another, similar model - one that is often-used in the field of ontology alignment - is described by Euzenat [26]. Our model is based on to the latter but extended to support our methodology and framework: we introduce the notions of *entities and models* to support the uniform mappings we discussed before, *context* and *community*.

3.4.1 Model & Entities

We start by defining our data model. With data model, we understand here the models and the content of the models that are being integrated by the framework. As we have discussed, one of the goals of our platform is to enable the integration of heterogeneous data models (e.g. OWL ontologies, Dogma Lexons, UML diagrams, relation database schemas, etc.). To enable this integration, we parse any input data model into a model / entity combination. The goal is not to describe or specify the full model, we only store a description of it, the type of the model, a reference to the actual model, and the entities that are part of the model. The following definition captures the semantics of an Entity Model:

Definition 8 (Entity Model) :
An Entity Model Υ is represented as a 4-tuple: $< ref, desc, \mathcal{T}, \mathcal{E} >$, where:

- ref is the reference identifier to the actual model, this should be a Unique Reference Identifier (URI);
- $desc$ is the description of the model;
- $\mathcal{T}$ is the type of the model, eg. *RDFS*, *OWL*, *Dogma-Lexon*, *Relational Database Schema*, ...;
- $\mathcal{E} = \{e_1 ... e_n\}$ is the set of Entities e_1 to e_n that the model contains.

An entity can be anything that is part of the model. It can be a class, a relation, a property, an instance, etc. We have defined an Entity in a similar way as the Entity Model, that is, it has a reference name, a representational name, and an Entity Model of which it is part of, the entity itself is not stored. We have defined it as such:

Definition 9 (Entity) :
An Entity e is defined as a 3-tuple: $< ref, term, \Upsilon >$, where:

- ref is the reference identifier of the entity;
- $term$ is the representational term of the entity;
- Υ is the model of which the entity is part of.

3.4.2 Mapping Element

We first introduce the 2 concepts of *Community* and *Context* before we define the mapping element. At this point, our goal is not to provide a complete and sound definition for the community and the context. They are very complex dimensions. Their use is mostly pragmatic and lies in the commitment phase, where they will both be leveraged by the actors involved in the commitment phase to commit to a final alignment.

However, to agree on a certain terminology, we give an informal, pragmatic definition of both concepts:

Definition 10 (Community) :
A community $\mathcal{P}$ has two intrinsic elements: an *ID* which identifies the community and a number of *actors* which identify the persons involved in the community. Two extra extrinsic characteristics are the *goal* of the community (for example, agreeing on an alignment of two ontologies) and the *process* they want to perform on the knowledge they share.

The community $\mathcal{P}$ is modelled as a 3-tuple: $< \mathcal{A}, \mathcal{G}, \rho >$ where:

- $\mathcal{A}$ is the set of Actors;
- $\mathcal{G}$ is the goal of the community; and
- ρ is the process the community wants to follow.

At this time, the context is still a black box. We do not define a precise model for the context. The actors are free to model it as they wish. The goal of the context however, is to model the timeframe, algorithms used, domain, ... in which the creation of the mapping element took place. While the context is not formally defined, it must not be overlooked. The context is an important information-source for the actors in the commitment-phase and is needed to keep the mapping element unambiguous from other mappings which share the same concepts, relation and community elements.

Definition 11 (Context) :
A Context γ is represented as the tuple: $< ref, term, description >$ where:

- ref is the unique reference to this particular context;
- $term$ is the terminological name of the context; and

- $description$ is the informal description of the context.

Now that we have provided definitions for the community and the context, we can move on to give the formal definition of a Mapping Element. We use the term Ontology in our definition very broadly: It can be a very rich ontology described in OWL or modelled in Dogma, but it can also be a lightweight "Ontology" such as a shared vocabulary, taxonomy, XML model, ... that contains far less (up to none) explicit semantics. The two ontologies used in a mapping also don't need to have a similar level of explicit semantics of formalism. The purpose is to allow mappings between very different data sources, which is an important requirement to be able to use this mapping framework in all of the foreseen scenarios.

Definition 12 (Mapping Element) :
We define a mapping element $\mathcal{M}$ between 2 ordered lists of entities $\mathcal{E}_1 = \{e_1, ..., e_x\} \epsilon\ Ontology\ \mathcal{O}_1$ and $\mathcal{E}_2 = \{e_{x+1}, ..., e_y\} \epsilon\ Ontology\ \mathcal{O}_2$ as a 6-tuple $< \mathcal{P}, \gamma, \mathcal{E}_1, \mathcal{E}_2, \mathcal{R}term_x, n >$ where:

- $\mathcal{P}$ stands for the community that was responsible for creating the mapping;
- γ is the context in which the mapping was created;
- $\mathcal{E}_1$ is the ordered list of entities belonging to Ontology $\mathcal{O}_1$;
- $\mathcal{E}_2$ is the ordered list of entities belonging to Ontology $\mathcal{O}_2$;
- $\mathcal{R}term_x$ is the relation term. This is not the actual relation definition. $\mathcal{R}term$ is the terminological name of the relation and should, in combination with the context, refer to a unique relation definition. We will discuss this in more detail when we introduce the semantics of a relation.

  When the relation is fuzzy, x is the strength of the relation.
- n is the degree of confidence, a measure of thrust in the fact that the mapping is appropriate. This degree of confidence will given by the community (directly or indirectly) to the mapping.

Both the strength of the relation (x) and the degree of confidence (n) are normalised between an upper bound ($\top$) and a lower bound ($\perp$). We set the upper and lower bound to respectively 1 and 0. A mapping element will always be at least unidirectional, however, the direction of the mapping depends on the relation definition. The relation definition will always be at least unidirectional but may also be bidirectional. A mapping is also uniquely identified by it's community, context, and entities. That means that only one possible relation between two sets of entities in a certain context and community is possible.

The mapping element can also be written in a less verbose manner. This will be the format we will use in the remainder of this thesis:

$$\mathcal{P}, \gamma : \mathcal{E}_1\ \mathcal{R}term_{x,n}\ \mathcal{E}_2$$

Let us clarify this definition with some examples:

- A mapping from the entity e_1 from Ontology $\mathcal{O}_1$ to an entity e_2 from Ontology $\mathcal{O}_2$. The mapping has a non-fuzzy *equivalence* relation. The confidence in the mapping is 0.85. This mapping would be written as:

$$\mathcal{P}, \gamma : \{e_1\}\ equivalence_{1,.85}\ \{e_2\}$$

- A mapping between the same entities, but now by a different community in a different context and with a fuzzy *subsumes* relation with strength 0.90 and confidence 0.70.

$$\mathcal{P}', \gamma' : \{e_1\}\ subsumes_{.90,.70}\ \{e_2\}$$

- A mapping between property $\{foaf : name\}$ from the FOAF Ontology[2] and properties $\{person : firstName, person : middleName, person : lastName\}$ from ebiquity's Person Ontology[3], with a fuzzy *union* relation with strength 0.75 and confidence 0.95.

$$\begin{gathered}\mathcal{P}'', \gamma'' : \{person : firstName, person : middleName, person : lastName\} \\ union_{.75,.95} \\ \{foaf : name\}\end{gathered}$$

3.4.3 Mapping Relations

A mapping element is a lexical representation of a conceptual mapping between two lists of entities. As the definition of a mapping element shows, the entities are only references to the actual entities from the model. In the same way, the relation name is only a reference to a conceptual relation definition. A relational name $\mathcal{R}term$ from a mapping combined with a context γ refers to exactly one unique relation definition $\mathcal{R}$. Going from the lexical representation to the conceptual relation corresponds to articulating the relational term into a relation definition. This articulation is done using an articulation mapping ct:

Definition 13 (Relation Articulation) :
Given the partial function $ct : \gamma \times \mathcal{R}term \rightarrow \mathcal{R}$, then

$$ct(\gamma, \mathcal{R}term) = \mathcal{R}.$$

An association $ct(\gamma, \mathcal{R}term) = \mathcal{R}$ is called the "relation articulation" or articulation of a relation term $\mathcal{R}term$ (in particular context γ) into a relation definition $\mathcal{R}$.

A relation definition should provide the system with a well-defined relation that can be interpreted by the different committers:

[2]http://www.foaf-project.org/
[3]http://ebiquity.umbc.edu/ontology/person.owl

Definition 14 (Relation Definition) :
A Relation Definition $\mathcal{R}$ is represented as the tuple: $< name, description, definition, properties >$ where:

- $name$ is the name of the relation definition;
- $description$ is the textual description of the relation;
- $definition$ is the logical description of the relation, in first order or description logic;
- $properties$ are the properties of the relation; these can be one or more of the following: unidirectional, bidirectional, transitive, reflexive, irreflexive, symmetrical, assymetrical and antisymmetrical.

3.5 Motivation

Now that we have thoroughly reviewed the methodology of our integration approach and its differentiating characteristics, we will elaborate on the motivation for our choices. We will give a number of additional advantages and benefits that are a direct consequence of our uniform, context-aware and community-supported integration platform. We argue that the advantages that we will discuss here show how this thesis tackles some of the problems of existing works in the semantic integration domain.

3.5.1 Scalability

Semantic integration is an expensive process and one that is currently not scalable: For every integration scenario the entire integration process must be redone. While this is not a big problem when the user is confronted with only a few integration problems, this becomes a critical issue when the user must integrate a large number of heterogeneous date sources - which is a trend that we see more and more in the current and future IT applications.

To tackle this scalability problem, we have split the integration process into two separate phases. The first phase, the mapping phase, must only be done once. After the plausible mappings are established in the mapping phase and stored in an application-neutral manner, they can be reused by selecting the relevant mappings for the specific integration scenario and committing them to the application-specific integration language. In that way, the end user can fully leverage the existing mappings so that the full integration process becomes much shorter, hence faster and more scalable.

However, by introducing mapping reuse, another issue of scalability represents itself: The storage of mappings must also be sufficiently scalable. It must scale in two different dimensions:

1. Performance wise: Fast querying of the mapping repository;
2. Usability wise: The actors must be able to efficiently find the relevant mappings.

In order to satisfy these concerns, the mappings stored in the repository have a simple structure which allows efficient storage and retrieval. While there will be a very large number of mappings stored in the mapping repository, every mapping has a *community* and *context* dimension which can be used, among others, as a grouping mechanism to guide the actor through this large mapping base and allow efficient look-up, browsing and searching in this very large mapping store.

3.5.2 Efficiency

One of the key goals of this thesis is to propose a methodology and platform to enable more efficient integration of heterogeneous data sources. Therefore, during the research on this thesis, a lot of focus was given to the efficiency and performance of our proposed mapping methodology and platform. As we have argued before, the separation of the standard integration process into our two phases is the core of our methodology. This can be leveraged to increase the performance and efficiency of the integration process in many ways:

Efficiency of the resulting alignment Our proposed methodology allows the actor to create an alignment that is specifically focused on its intended use (that is, application specific). In previous integration frameworks, the involved actors might have opted to build a more general alignment which could be used in more scenarios because the high cost of the integration process. This however makes the alignment less efficient towards the specific applications for which it is intended to be used.

Our approach to increase the efficiency of the integration process is similar to the OMGs vision of model driven development where platform independent models are created and can afterwards be transformed into platform specific models.

Furthermore, because the created mappings are context-aware and community-driven, they represent the domain and connection with other domains more comprehensibly then the alignments created by the traditional integration approaches. External knowledge engineers are typically the bottleneck to the alignment comprehensiveness, as they are not capable to capture all the varieties of the mapping elements that might take place in a community and associated communities.

Efficiency of the integration process Not only are the resulting alignments more efficient, the integration process itself is also more efficient. The increased efficiency of the process is achieved by several properties of our proposed methodology and platform:

- The mappings stored by the mapping layer are representation-neutral but still semantically well-defined. This makes it possible to split up the integration process to different responsible stakeholders. For example, a domain expert who has no knowledge of the representation model in which the alignment should be created, can create the valid mappings in our application-neutral format. In the second phase, the application engineer, who has insufficient knowledge about the domain to create the mappings, can select the mappings created by the knowledge engineer and commit them to the application-specific alignment.

- The reuse of existing mappings can be leveraged to automatically create new mappings using inference. Automated matching algorithms can look-up existing mappings and use reasoning engines to find new mappings. It has been shown that reusing existing mappings can help in matching concepts from new ontologies. We have outlined several example scenarios in Figure 3.7.

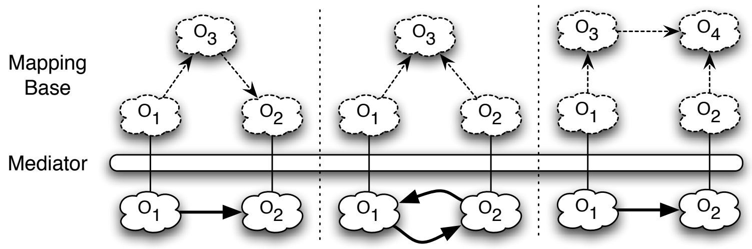


Figure 3.7: Different scenarios in which existing mappings can be reused in Ontology matching to create new ones.

As illustrated in the figure above, the mediator allows the actor to find existing mappings in the mapping repository between the given ontologies $\mathcal{O}_1$, $\mathcal{O}_2$ using the already mapped ontologies $\mathcal{O}_3$, $\mathcal{O}_4$. The mappings can be reused by the actor (which can be a real user or an automated matching algorithm) to create new and better mappings between the two given ontologies.

In the first scenario, mappings already exist from ontology $\mathcal{O}_1$ to ontology $\mathcal{O}_3$ and from ontology $\mathcal{O}_3$ to $\mathcal{O}_1$. These mappings can be used to create (automatically or guided by a human actor) new mappings from ontology $\mathcal{O}_1$ to ontology $\mathcal{O}_2$. In the second scenario, both ontology $\mathcal{O}_1$ and $\mathcal{O}_2$ have been mapped to ontology $\mathcal{O}_3$. Therefore, this can be leveraged to create new mappings from $\mathcal{O}_1$ to $\mathcal{O}_2$ and the other way around. The last scenario goes one step further in that both $\mathcal{O}_1$ and $\mathcal{O}_2$ are already mapped to two other ontologies, respectively $\mathcal{O}_3$ and $\mathcal{O}_4$. This can be used in turn to create new mappings from $\mathcal{O}_1$ to $\mathcal{O}_2$.

- In community-driven integration, the expenses of the integration process are shifted from the ontology/alignment maintainers to the communities employing them. This shift results in an adequate investment distribution among the mappings. Specifically, the mappings of higher importance to the communities gain more support in terms of more associated resources.

- The storage of existing mappings provides an excellent evaluation environment for new mappings created by automated matchers. These mappings can be checked against similar existing mappings, created by human actors such as domain experts and knowledge engineers. In this way, one can create an extremely large test-bed for the development of new matching algorithms: Use the algorithm to map a very large number of ontologies that already have high quality existing mappings in the mapping repository (probably created by human actors). This allows for much more fine-grained and thrust-worthy evaluation results compared to the toy-examples that are usually used to evaluate matching algorithms. The existing mappings could also be leveraged to make the algorithms use machine-learning principles to "learn" their optimal set of configuration settings which, based on our own experiences, has a large influence on the efficiency and quality of such matching algorithms.

3.5.3 Evolution

When mappings can not be stored in a application-neutral manner and reused at a late time-frame, this raises serious issues in real-world scenarios with respect to the evolution of the mappings. In real applications, the models that need to be integrated evolve constantly. Furthermore, also the applications that use the resulting alignments will change. For example, a new updated version of the application might introduce changes in the alignment format it uses. When mappings are not stored in a application-neutral manner, they do not support any evolution on the input (the integrated models) or the output (the resulting alignments) side. We will point out the problems of the current integration approaches and how our approach deals with these problems for the two possible scenarios:

Evolving input models The first issue presents itself when one or both or the integrated models change. When this happens, the existing alignment will no longer represent the correct integration of both models. In the existing integration frameworks, this scenario obligates the actors to make an entire new alignment from scratch. This is almost as much work as creating the initial alignment. The amount of work to integrate the new models is also not dependant on the number of changes in the models: Even if the models only changed on a few entities, the entire alignment needs to be redone.

In our approach, the actor must only create new mappings for the new or changed entities. Many of the existing mappings between both models will still be valid and can be reused. We show an example of this scenario in figure 3.8. As shown in the figure, initially model M1 is mapped to model M2 and the mappings are stored in a mapping repository. When model M1 changes to M1′, the existing mappings stay the same in the mapping repository. Only new mappings must be created for the changed entities in model M1′. As shown in the figure, the new mappings *Mappings′* stay the same except for the extra mappings denoted by the shaded rectangle. Another consequence of this approach is that now both models M1 and M1′ (the old one and the updated one) have been mapped to the

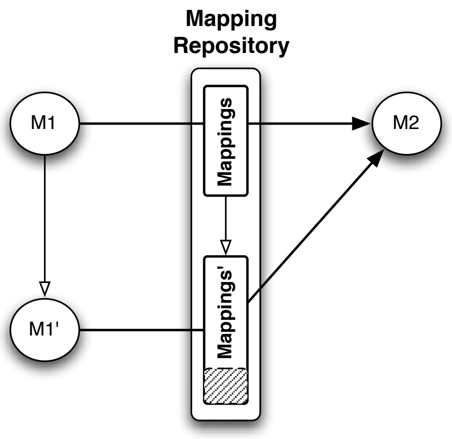


Figure 3.8: A possible evolution scenario where one model changes and the mappings need to be updated accordingly.

other model M2 so the previous version of the model is still supported by the mapping repository.

Evolving application scenarios The second scenario represents itself when the application in which the alignment resulting from the integration process must be used, changes. In existing works, the entire alignment must again be entirely recreated. In our case, only the second phase of our integration methodology must be redone: The actor can reuse the mappings stored in the repository to commit to the new alignment format.

3.6 Conclusions

We started this chapter by introducing the two cornerstones of our proposed integration approach: Mapping reuse and application-specific mappings. We identified several scenarios that are problematic for the existing integration frameworks, but are much better handled using our methodology. In the second section of this chapter, we elaborated in more detail on our methodology. We discussed in detail the two phases of our integration approach: The Mapping phase and the Commitment phase. We discussed the tree most significant characteristics of our approach and methodology: Uniform and context-aware mappings, and a community-driven mapping process. In the third section we defined the precise semantics of the conceptual data model used by our methodology: We provided exact definitions of the Context, the Community, the Mapping, the Model, the Entity, and the Relation Definition used in our approach. To conclude the section, we elaborated on the motivation for our choices. We showed how our approach has significant benefits over existing approaches and how it accomplishes the goals we have set for this thesis.

4
The Platform

In this chapter, we present our integration platform implementing the conceptual framework and specifications we presented in the previous chapter. We first discuss the architecture of the platform: We describe how we have mapped our two-step methodology (the mapping phase and commitment phase) and the specific characteristics of our approach (uniform, context-aware, and community-driven mappings) to our integration platform. In the second section we elaborate on the implementation details of the Platform Server. We show its technology stack and motivate our choices. In the third section of this chapter, we provide the reader with an overview on our Mapping Client tool. We also give a brief overview of its implementation while not going into too much irrelevant details. We end this chapter with a preliminary evaluation of the platform. In a real-life scenario we integrate two ontologies using the platform by following our methodology. In this way, we show how the platform we developed supports the integration of two heterogeneous models using the methodology we presented in the previous chapter.

4.1 Overview

In the previous chapter we have outlined the methodology and approach of our integration process. In this section, we will present the architecture of our platform that implements this methodology. As we have discussed, the two cornerstones of our proposed integration approach are mapping reuse and application-specific mappings. To enable these properties on our platform, we have split-up the platform into two layers in two orthogonal dimensions, as depicted in figure 4.1. In the first dimension we split-up the platform into a *Remote Layer* and a *Client Layer*. In the second dimensions we map our two step methodology, the *Mapping phase* and the *Commitment phase* upon the Client Layer. Furthermore, a mediator is needed that mediates between the server and the clients and provides the actors with the necessary services to manage and leverage the existing mappings.

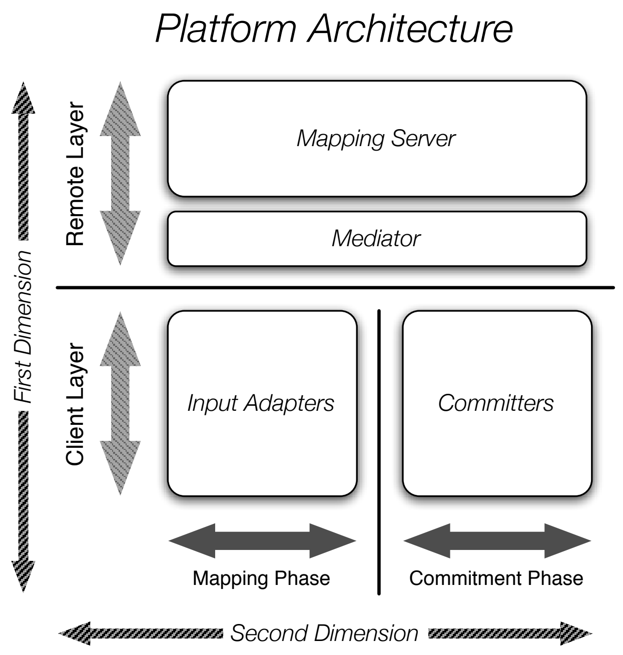


Figure 4.1: The two orthogonal dimensions of separation in our platform architecture

The first dimension is the local-remote dimension. To allow public mapping reuse and collaboration, the mappings must be stored on a server that can be accessed by the stakeholders. In order for these actors to be able to create, manage, maintain and commit these mappings, they must have one or more client applications that support these features. Therefore, we have split up our platform into the two horizontal layers depicted in figure 4.1:

Remote Layer: In the remote layer, the mappings created in the mapping phase are stored on a *Mapping Server*. A *Mediator* is needed to enable the communication from different types of clients with this mapping server. Furthermore, the mediator provides an extra layer of services on top of the simple storage of mappings offered by the mapping server.

Client Layer: The client layer contains all the applications that the involved actors will use to interact with the mappings stored on the remote layer. We can identify two kinds of interactions with the remote layer corresponding to the second dimension and similarly to our two-step integration methodology:

1. In what we call the *Mapping phase*, the actors will elicit and create the mappings between the heterogeneous models. This will often include the *matching* of the different entities of the models. Furthermore, this phase also entails the different aspects of our integration platform we have discussed in the previous chapter: In this phase, different communities can govern and express their confidence and thrust in the different mappings, new mappings can be created with a specific context by leveraging existing mappings or because the input mappings have changed and the mappings must evolve accordingly.
2. The second kind of interactions by the client applications with the remote layer correspond to the second step of our methodology and is called the *Commitment phase*. In this phase, the actors will select a meaningful set of mappings which are stored on the mapping server and create an application-specific alignment from this set. The choice of mappings which they select should be specific for the intended use of the resulting alignment. The intended use will be application-, domain-, and community-specific. Furthermore, the client applications from this phase must also enable the actor to transform his selection of representation-neutral, uniform mappings from the mappings server to an application-specific alignment format such as OWL, ω-RIDL, the Alignment API, Semantic Web Rule Language (SWRL), etc. To do that, the application must be able to interpret the conceptual relations from the mappings in the mapping server to mappings or rules in the chosen alignment format.

In the following sections, we will elaborate on each layer of our architecture. In each layer, we will further discuss the different parts it contains and the role and responsibilities of each part. After that, we will present the full, detailed view of our architecture and move on to discuss its implementation details.

4.1.1 Remote Layer

The goal of the remote layer is to enable the complete separation of the mapping- and commitment phase. To arrive at such a separation, the mappings must be stored in a repository to be reused at a later time. By storing the mappings in a repository, the mappings have a persistent state between the mapping and commitment phase which allows the complete separation between these phases.

Requirements

Based on our methodology and integration approach, we identified the following requirements and responsibilities for the remote layer:

- The mappings created in the mapping phase must be stored in a repository which must be remotely accessible.

- This repository must be able to store mappings from any given input format. That is, it must be possible to create mappings between any kind of model and store these on the remote layer of the architecture.
- The mappings must be stored in a application-neutral, uniform manner. The mappings must also be context- and community-aware. The relations in the mapping must furthermore be well-defined and context-dependant. Therefore, the remote layer must support the relation articulation function we have defined in the section on our mapping semantics.
- The involved actors must be able to select a meaningful set of mappings from this mapping repository in order to create an application-specific alignment. They should be able to efficiently search and browse the mapping repository to discover the relevant mappings to allow for an efficient and scalable integration process.

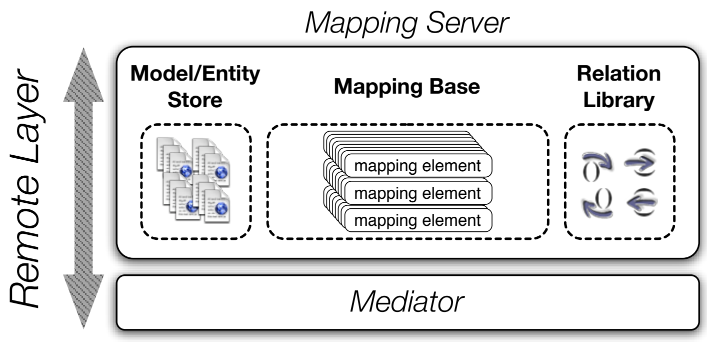


Figure 4.2: The different parts of the remote layer.

These requirements have led to four big parts that constitute the Mapping Layer: the *Mapping Base* which stores the mappings, the *Relation Library* which provides definitions of the relations used in the mappings, the *Model and Entity Store* that enables the uniform use of different data sources and a *Mediator* which mediates between this layer and the actors of the system. The overview of the remote layer is depicted in figure 4.2.

Mapping Base

The *Mapping Base* is the repository in which the mapping elements we have defined in section 3.4 are stored. In time, the Mapping Base will become very large (to the order of millions) as a match between two models with respectively n and m number of elements will yield n^m mappings. Not all of these mapping elements should be added to the mapping base however as most entities will be completely unrelated. Nevertheless, as we described in our integration methodology, the selection of relevant and meaningful mappings happens at the commitment phase. Hence, the actor should not decide (up to some limit - probably using some threshold value) which mappings are relevant enough to add to the mapping base. The large size of the mapping base is one of the features and brings many of the advantages to the framework: The larger the Mapping Base, the more mappings can be reused.

Because of the large size of the Mapping Base, it is important that it allows efficient look-up, search and browsing through this large repository. It must be able to quickly return the mappings given a certain concept, context, community, etc.

Relation Library

The relation between the different entities of the mapping is the most important element of a mapping element. The relation denotes how the entities correspond and relate to each other and how the first list of entities can be translated to the second. The possible scope of the complexity of the relations is very wide, ranging from very simple relations like "Equal" or "Subtype of" to complex, composed relations expressed in a certain language *L*. For example, a complex relation expressed in first-order logic could be:

$$\forall x, z\ grandparent(x, z) \implies \exists y; parent(x, y)\ \wedge\ parent(y, z)$$

Furthermore, even simple relations might have small, but still significant semantic differences between different mappings. For example, the exact semantics of "subtype" might be different depending on the context of the mappings. Therefore, as we have discussed in section 3.4, we use a relation-articulation function to articulate the lexical term of the relation in the mapping element to a relation definition stored in the relation library.

Model and Entity Store

As we have stated before, the goal of our integration platform is to provide a unified platform that allows actors to integrate entities from heterogeneous models. To enable such a platform, we store the entity and model information in a separate *Model and Entity Store*. In this repository, we save the necessary information about a model and it's entities such as the type of the model, a representation term, a reference identifier that allows the actor to uniquely identify entities, and a description of the model and its entities.

Mediator

The mediator should be seen as the interface to the mapping layer. It enables the actors to add and remove mapping elements. It also acts as a service that connects committers to the mapping base: through the mediator, actors can query, browse and search the mapping layer. It is modelled as a distinct part from the mapping layer because it also allows some basic reasoning on alignments and mappings, this application logic should be separated from the mapping base and dictionaries whose sole responsibility is to store the mapping elements.

4.1.2 Client Layer

The *Client Layer* contains the client-side applications the actors will use to do the actual integration. As we have discussed in section 3, our methodology separates this integration process into a *Mapping Phase* and a *Commitment Phase*. The Client Layer must provide the actors with the necessary tools that support these phases. Several different tools can connect with the remote layer and each of them can support one or more of the necessary features.

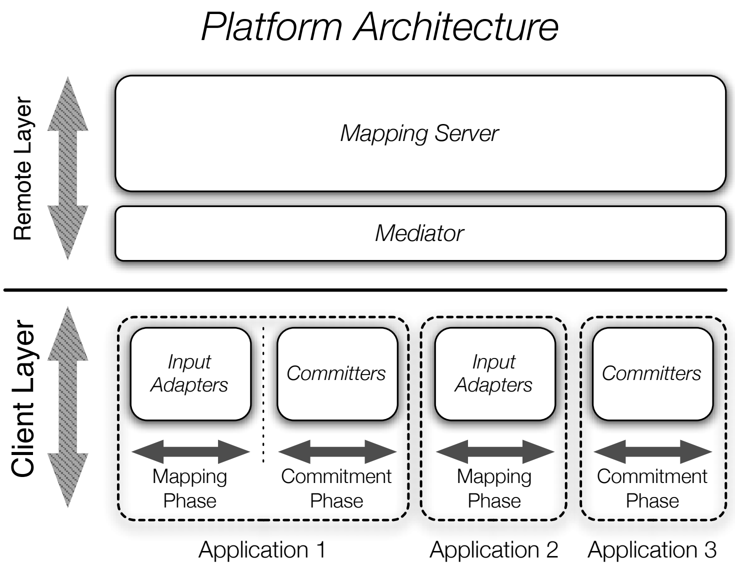


Figure 4.3: Different applications from the Client Layer supporting our two-step methodology connecting to the Remote Layer.

Figure 4.3 shows different possible clients from the Client Layer. Each client may support a part of our two-step methodology. In this section, we will discuss the different features that must be supported by the tools to enable the actors to perform each phase of our methodology.

Mapping Phase

The *Mapping Phase* is the first step of our integration approach. In this step, the actor will match two heterogeneous models and elicit a number of mappings between the entities of those models. These mappings will be stored on the mapping server in order to be used in the commitment phase.

In practice, the process creating of mapping between entities of two heterogeneous models and storing them on the mapping server can again be divided into several methodological steps, some only computational and some needing human interaction:

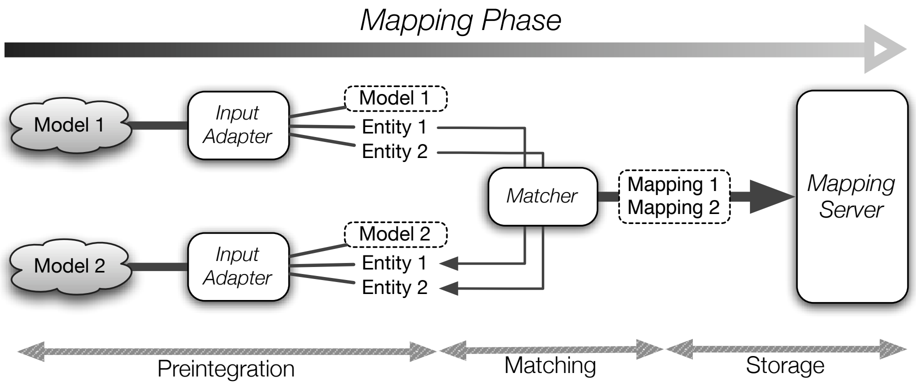


Figure 4.4: The different methodological steps of the Mapping Phase.

1. **Preintegration:** In this step, the syntactical heterogeneities of the models are overcome by parsing them into our uniform mapping model: The model and its content are transformed into our Model and Entities data model. The transformation from the original input models to our own Model/Entity data model is handled by *Input Adapters*. The Input Adapters must make sure that each entity has a unique reference identifier, a description and is part of a specific model. Each model must also have a unique reference identifier, representation name and a model type. For each supported format, an input adapter must be created to transform the format into our mapping model. For example, an input model may be an OWL Ontology, RDFS Ontology, SQL Schema, etc.
2. **Matching:** In the second step, the different entities resulting from the preintegration phase are matched and mappings are created between similar entities. Furthermore, the relation term from each mapping must point to a specific relation definition. It can point to an existing relation definition from our relation library or in the other case, the actor must create a new relation definition.
3. **Storage:** The Model and Entities, Mappings, and Relation Definitions must be stored in the Model/Entity store, Mapping Base, and Relation Library on the Mapping Server respectively in order to be shared and reused in the commitment phase.

Commitment Phase

In the commitment phase, the actor will select a meaningful set of mappings and create an application-specific alignment from this set. We call the process of selecting a meaning full set of mappings for the intended alignment and interpreting and transforming these mappings to an application-specific alignment format ***Committing*** *to an application-specific alignment*. The tools from the Client Layer must help the actor to discover and search the mapping base to find the relevant and meaningful mappings for the intended use of the alignment. Furthermore, the tools must guide the actor to interpret this set of mappings with their relation definition to transform it to an application-specific alignment in a certain integration format.

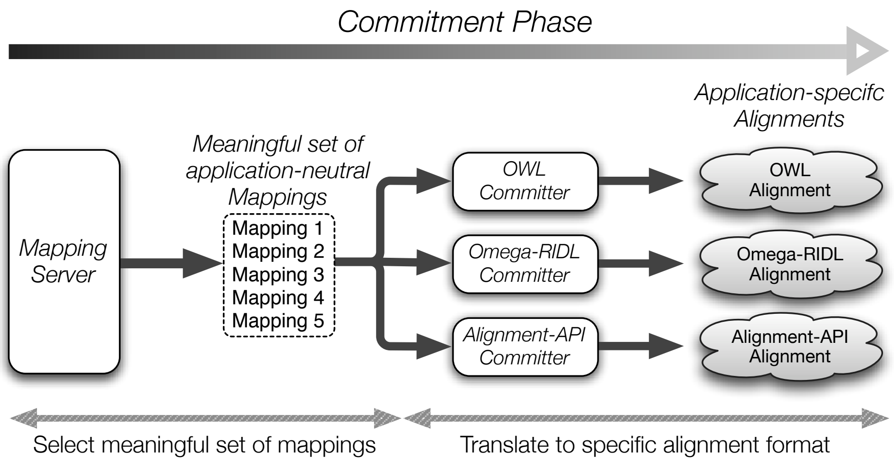


Figure 4.5: The methodological steps of the Commitment phase and the necessary tool-support.

Figure 4.5 depicts the Commitment Phase process. We identify the two methodological steps that are needed to create the final, application-specific alignment. First, a meaningful set of mappings must be selected from the Mapping Base. Secondly, this set of mappings must be interpreted and translated into a specific alignment format. We will now elaborate on both steps of the commitment process:

1. **Select meaningful set of Mappings:**

   When an actor needs an alignment to integrate two heterogeneous data sources, this alignment will be application-specific: To integrate the two sources, the alignment will need to be executed to, for example, transform data in one format to another data model format. The kind of mappings that will be needed to do this integration are dependant on the use of the alignment. For example, two scenarios in which the same two models need to be integrated, may still need a different alignment. In the first scenario, it might be sufficient that a certain concept from the first model is similar to a concept from the second model; while in the second scenario, the concept from the first model might need a specific transformation function to be translated to the concept from the second model.

   In this step, the actor must search and browse the mapping base to discover the meaningful set of mappings that best models or approximates the intended use of the alignment. He can do this efficiently because of our context-aware and community-driven mappings: The mappings are grouped by context and community which enables the actor to quickly find the relevant mappings. Furthermore, the community might have expressed their thrust and confidence in certain mappings which may be used by the actor as a measure of quality and acceptance of the mapping.

2. **Translate to a specific alignment format:**

   Given the meaningful set of mappings from the previous step of the committing process, the actor must now interpret and translate these conceptual mappings to an application-specific integration model or format. This is necessary as the final alignments will need to be executed which means they must be understood and interpreted or compiled for a certain application.

   Translating the application-neutral mappings into a specific alignment format conforms to: i) Interpreting the conceptual relation definitions into executable relations, functions, or rules in the alignment format and ii) translating the mappings into the format specified by the alignment model. These two functions must be supported by the different *Committers*. In some cases, both functions can be executed fully automatically, but in most cases, some human interaction will be necessary.

   Given the meaningful set of mappings from the previous step, the actor can translate it into several final alignments, each in a specific alignment format. Each alignment format will have his own committer that can interpret the relations and translate the mappings into the specific format. The final alignments, independently from the format used, will typically contain the set of mappings, the arity of the alignment (one-to-one, one-to-many, etc.), the models that are aligned, etc.

4.2 Mapping Server

In this section we present in detail our Mapping Server which implements the Remote Layer of our conceptual architecture. We present the technology stack our server is build upon and discuss each relevant layer of the stack in more detail. We will however not go into to much implementation details as this goes beyond the scope and purpose of this thesis.

One of the goals of this thesis is to present a scalable integration approach and platform. Therefore, we have opted to use a classical three-tiered J2EE architecture. This allows the platform to scale horizontally as well as vertically as we will show in a later section. Figure 4.6 depicts our three-tiered architecture and the technologies or solutions we have chosen for each layer.

In the following sections we will elaborate on the Data Tier and Business Tier. The Client Tier corresponds with our Client Layer, which we will discuss in more detail later.

4.2.1 Data Tier

The cornerstone of our approach is our two-steps methodology which introduces an intermediate, persistent state of application-neutral mappings. In order for our platform to support this methodology, we have a Data Tier that handles the persistence of the mappings. The Data Tier consists of two successive layers:

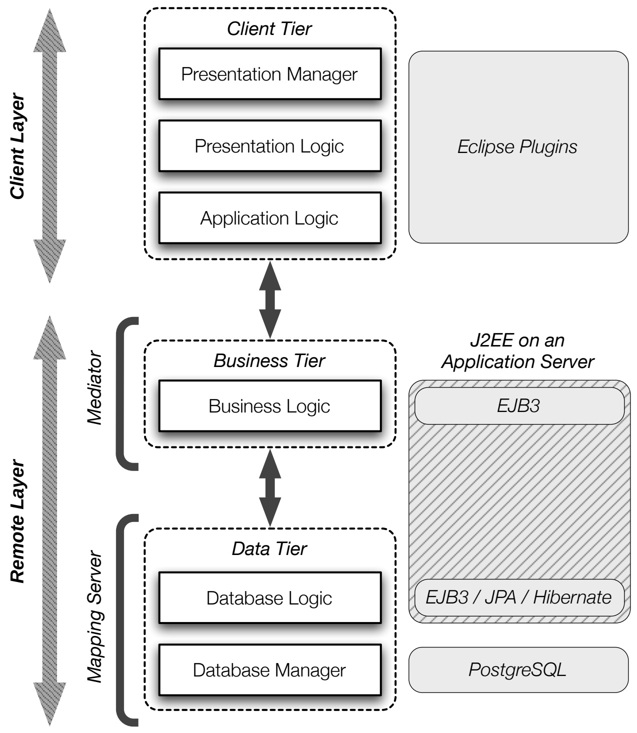


Figure 4.6: The architecture and technology stack of the integration platform implementing our approach and methodology.

1. **Database Manager:** The first layer of the Data Tier is the *Database Manager*. The Database Manager is responsible for the low-level storage of the mappings. We use the PostgreSQL DBMS[1] to store the mappings. The data model used by the database server is depicted in figure 4.7.

2. **Database Logic:** The Database Logic layer on top of the Database Manager layer handles the interaction with the database. Its purpose is to abstract the persistent level from the other parts of the platform architecture. In that way, the persistence layer can be easily swapper for another one, like another DBMS (MySQL, Oracle, etc.) or another persistent model, using ontologies instead of relational databases for example. The other parts of the system only interact with the domain object using Java classes and the database logic layer translates these interactions to SQL queries on the database. There are several best-practices and patterns in the J2EE community that enable such an abstraction. An overview of the ones we have adopted is depicted in figure 4.8:

   Java Persistence API: The Java Persistence API (JPA) specification allows developers to manage relational data as Java classes and objects. It is an object-relational mapper

[1]PostgreSQL is a powerful, open-source DBMS: http://www.postgresql.com

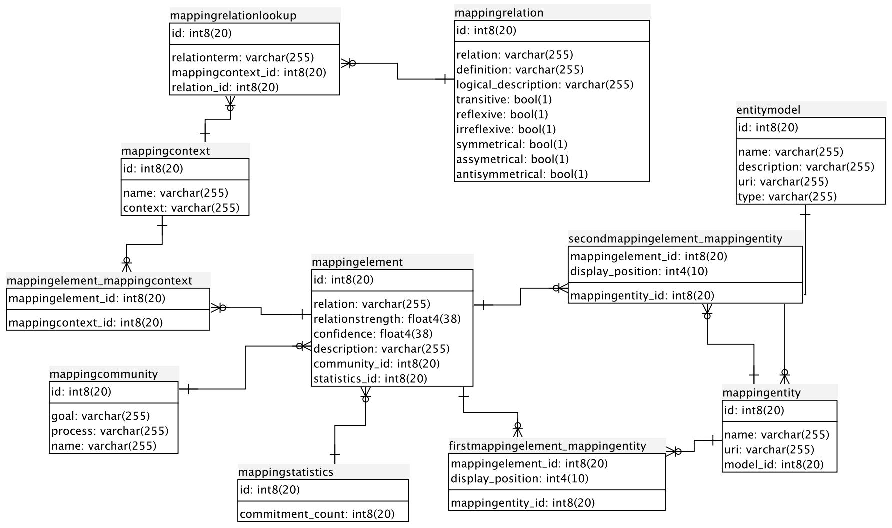


Figure 4.7: The Entity-Relational Model of the persistent data stored in the PostgreSQL DBMS.

that maps relation tables and rows into Java classes and objects respectively. JPA is only a specification and an actual product implementing this specification is needed. We use Hibernate[2] as the implementation for the JPA specification. Hibernate is a powerful, open-source solution that has become the de-facto standard within the Java community.

Data Transfer Object: As we have discussed, the JPA enables the system to interact with Java objects instead of performing SQL queries on the database. The relational model used in the database is therefore translated to a domain model of Java objets. We have used the Data Transfer Object (DTO) pattern to implement these domain objects. The DTOs do not contain any behaviour except for storage and retrieval of its own data (accessors and mutators). The domain object implemented as DTOs are Entity Beans: They can be deployed on the Application Server and used throughout the platform, locally and remotely.

Figure 4.9 shows the domain model as a UML class diagram. As we have used the DTO-pattern, only getter and setter operations exist on the classes so we have not shown them in the figure.

Data Access Object: Data Access Objects (DAO) are a Core J2EE Design Pattern and considered best practice. The advantage of using data access objects is that any business object (which contains application or operation specific details) does not require direct knowledge of the final destination for the information it manipulates. As a result, if it is necessary to change where or how that data is stored that modification can be

[2]http://www.hibernate

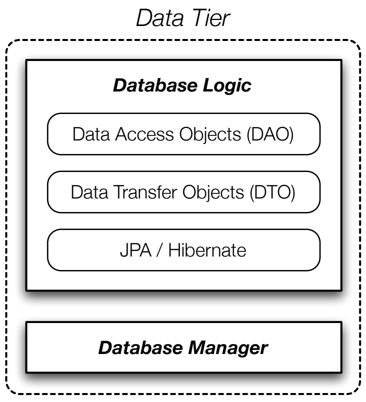


Figure 4.8: The patterns and best-practices used in the Database Logic layer.

made without needing to change the main application. Data Access Objects can be used in Java to insulate an application from the underlying Java persistence technology, which could be JDBC, JDO, EJB CMP, TopLink, Hibernate (which we use), iBATIS, or any one of a range of technologies. Using Data Access Objects means the underlying technology can be upgraded or swapped without changing other parts of the application.

Figure 4.10 depicts the DAO interfaces for each class from the domain model DTOs. By implementing the DAO pattern using interfaces, the architecture is even further abstracted from the underlying technologies. In the second figure 4.11, the JPA-based implementation of the DAO interfaces are shown using a UML class diagram. We have omitted the operations and attributes in the second figure as these are exactly the same as the ones from the interfaces they implement.

Figure 4.11 also shows that the JPA-specific DAO classes are *Stateless Session Beans* from the EJB3 specification. A class can be specified as a Stateless Session Bean using *annotations* first introduced in Java 5. When a class is specified as a session bean, the class automatically becomes available in the application server and can be used remotely or locally to support the *Inversion of Control (IOC)* pattern.

4.2.2 Business Tier

The *Business Tier* of our implementation architecture corresponds with the mediator of the platform. It mediates between the data tier and the requests from the remote client applications. It provides services which allow the clients to interact with the data stored in the data tier. We have implemented this tier using two specific patterns and best-practices: the *Session Facade* pattern and the *Business Delegate* pattern.

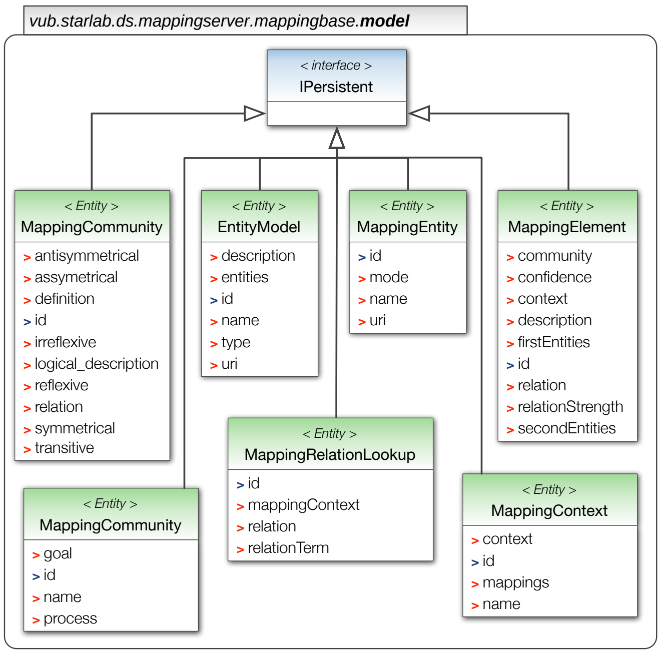


Figure 4.9: The domain Model in UML, implemented using the DTO pattern.

Session Facade

The Session Facade pattern defines a higher-level business component that contains and centralizes complex interactions between lower-level business components such as the DAOs and DTOs we discussed earlier. A Session Facade is implemented as a session enterprise bean. It provides clients with a single interface for the functionality of conceptual server subset. It also decouples lower-level business components from one another, making designs more flexible and comprehensible.

Fine-grained access through remote interfaces is inadvisable because it increases network traffic and latency. Figure 4.12 shows the sequence diagram of one of our session facades. Without a Session Facade, the client would have to access the fine-grained business objects through a remote interface. The multiple fine-grained calls would create a great deal of network traffic, and performance would suffer because of the high latency of the remote calls.

Business Delegate

The purpose of the Business Delegate pattern is to reduce coupling between presentation-tier clients and business services. The Business Delegate hides the underlying implementation details of the business service, such as lookup and access details of the EJB architecture.

The Business Delegate acts as a client-side business abstraction; it provides an abstraction for,

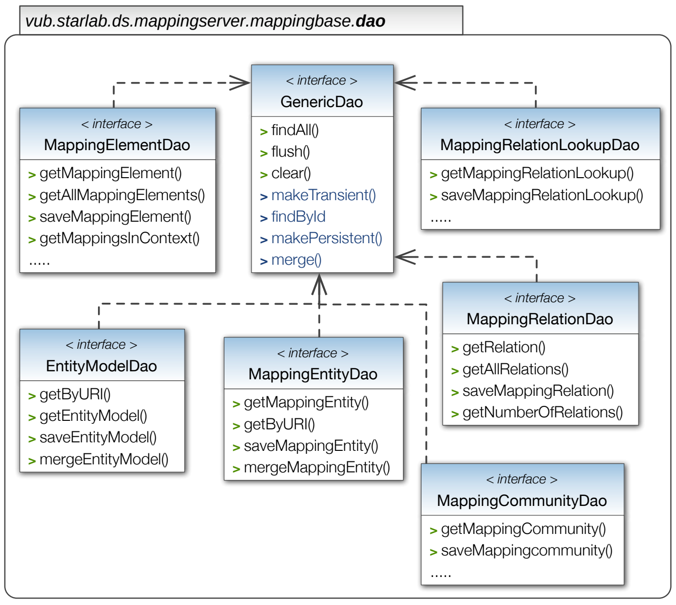


Figure 4.10: The Data Access Objects Model interfaces in UML.

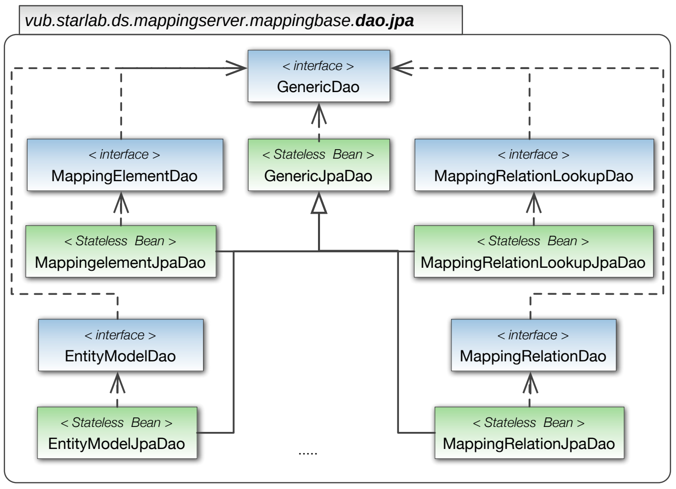


Figure 4.11: The JPA-specific Data Access Objects Model implementations in UML.

and thus hides, the implementation of the business services. Using a Business Delegate reduces the coupling between presentation-tier clients and the system's business services.

We also use the factory pattern in combination with the Business Delegate pattern to increase the level op abstraction and reduce the level of coupling even further.

Figure 4.13 depicts a subset of the classes involved with the MappingService in the Business Tier as a class diagram.

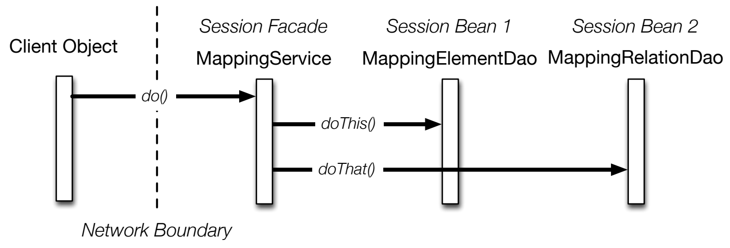


Figure 4.12: A sequence diagram showing the Session Facade pattern applied in the Business Tier.

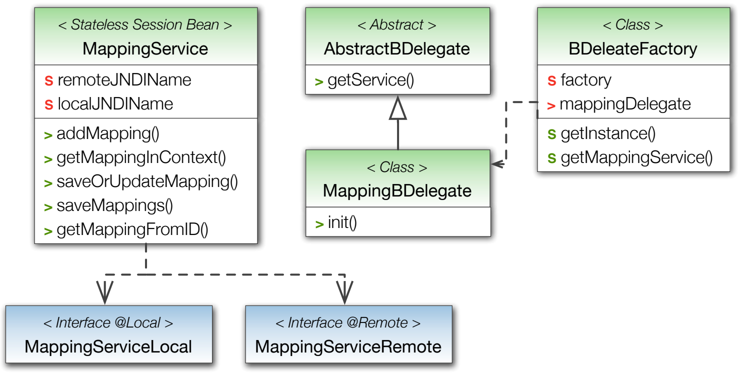


Figure 4.13: A class diagram showing the classes and interfaces from the Business Tier relevant to the Mapping Service.

In Figure 4.14 a UML sequence diagram of a client object calling the *BDelegateFactory* to receive a *MappingServiceRemote* interface is depicted. The *MappingServiceRemote* interface is used remotely to interact with the Data Tier. On the server, the remote interface is implemented by the *MappingService* stateless session bean.

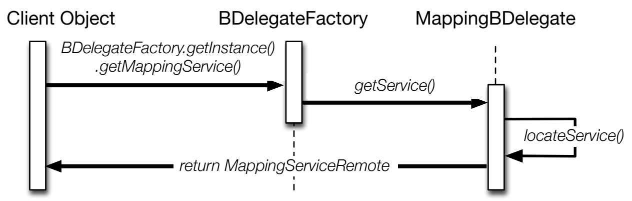


Figure 4.14: A sequence diagram showing the Business Delegate pattern applied in the Business Tier.

4.2.3 Scalability

One of the goals of this thesis is to present a scalable integration platform. This scalability requirement is reflected in our integration approach and two-step methodology promoting reuse of mappings. However, the scalability factor is equally important in our implementation architecture: If the methodology allows for a scalable integration approach, the platform and architecture enabling the methodology must too. Therefore, we will discuss briefly the scalability potential of our platform. While this discussion falls mostly outside the scope of this thesis, we find it important to discuss it briefly in order to back-up our choice of technologies and frameworks on which we have build our platform.

There are two types of scalability, vertical scalability and horizontal scalability:

Vertical Scaling is achieved by adding capacity (CPU, memory, storage) to the existing infrastructure. It requires no changes to the architecture of a system and is usually cheaper than horizontal scalability. However, one can not endlessly vertically scale-up a system and the limits are quickly reached. Furthermore, vertical scaling decreases the reliability and availability of the system as a single failure is more likely to lead to system failure. J2EE supports vertical scaling because of its automatic lifecycle management: Adding more capacity to a server allows it to manage more components (Entity beans, Service beans, etc.).

Horizontal Scaling is achieved by adding servers to the system. While it increases the complexity of the system architecture, it should be possible to scale a system almost endlessly in this way. It further increases the reliability, availability, capacity and performance of the system. Our architecture supports horizontal scaling because the J2EE application servers on which the business logic runs and database management system for storage can be replicated and clustered and the whole can be load-balanced without many changed to the architecture of the platform.

4.3 Mapping Client

To validate our methodology and approach, we have implemented a client-application that enables actors to create mappings and store them on the Mapping Server. The tool also allows actors to browse the Mapping Server and select a meaningful set of mappings to commit to a final alignment.

In this section, we will present our tool: We give an overview of the looks and features implemented by the tool and provide a brief overview of its implementation details.

We start the discussion of our Mapping Client with an overview of the main parts of the Mapping tool and how it is situated within our platform's architecture. Figure 4.15 gives an overview or the most important components of the tool. It further shows how and where the tool is situated within our platform's architecture we presented in section 4.1. The Mapping Client is one of the potentially many client applications the actors can use to integrate heterogeneous models using our integration methodology. In this particular case, the actor can use the tool to do both phases of our integration methodology: The tool supports the Mapping Phase as well as the Commitment Phase.

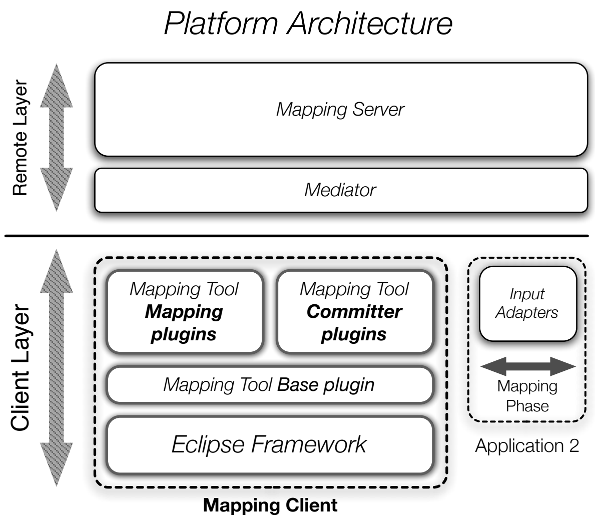


Figure 4.15: The high-level architecture of our mapping Client and its location and role within our integration platform.

Our Mapping Client is build upon the Eclipse Framework using a number of plugins. The collection of plugins can be divided into three main components depending on their role within our integration approach and methodology:

1. The Base plugin is the foundation for the client.
2. The Mapping plugins corresponds to the plugins needed to do the Mapping-phase of our integration methodology. They are responsible for providing the necessary features and support the actor in transforming heterogeneous models to our data model and creating mappings between these models to be stored on the Mapping Server.
3. The Committer plugins corresponds to the plugins needed to do the Commitment phase of our integration methodology. That is, finding and selecting a meaningful set of mappings from our Mapping Server and transforming them into an application-specific alignment.

In the next section we discuss the two components corresponding to the two phases of our methodology in further detail after we have briefly introduced the Eclipse Framework and our Base plugin.

The Eclipse Tool Project[3] is an open source software development project that provides a ro-

[3]http://www.eclipse.org

bust and full-featured industry platform for the development of highly integrated tools. It is composed of three subprojects, Platform, JDT - Java development tools, and PDE - Plug-in development environment. The success of the Eclipse Platform depends on how well it enables a wide range of tool builders to build best of breed integrated tools.

Building a visual component requires a graphical framework of some sort. In Eclipse this is provided by GEF. The Graphical Editing Framework (GEF) allows developers to create a rich graphical editor from an existing application model.

GEF consists of two plug-ins:

- The `org.eclipse.draw2d` plug-in provides a layout and rendering toolkit for displaying graphics.
- The `org.eclipse.gef` framework, which defines a lot of utility classes, ready for the user to be extended.

The developer can take advantage of the many common operations provided in GEF and/or extend them for the specific domain. GEF employs a strict MVC (model-view-controller) architecture which enables simple changes to be applied to the model from the view and back, but always via a controller. Using this powerful plug-in architecture as a backbone, it enables developers to add functionality as time progresses. It forces them to a good design, avoiding a monolithic, static environment unable to accept future changes.

4.3.1 Mapping Client Base

The Mapping Client Base is the foundation for the tool that was implemented (see Figure 4.15). It contains all the basic building blocks needed by the Mapping and Committer plugins and can be viewed as the framework of our Mapping Client.

In the next section we explore the base plug-in in more detail to give the reader an idea as to how it was implemented using the eclipse framework.

Architecture

Because GEF enforces a strict Model-View-Controller architecture [41], the developer has to model his software component using:

- *models*: to represent the business logic;
- *controllers*: communicate between the model and the view;
- *views*: graphical representation of elements in the model.

We discuss each component in more detail:

Model The first component that we discuss is the model. The idea is that this model contains all the business logic needed to graphically represent the Mapping Elements, Models and Entities and support editing operations upon them. The model in the Base plugin is similar to the model from the Data Tier on the Mapping Server. However, we have to extend the model classes from the Data Tier in order to support the graphical representation of the models. Furthermore, GEF requires that each "thing" that can be selected in the views and editors must have its own class.

Figure 4.16 depicts the two types of domain objects defined by the Mapping Client Base: The *persistent objects* are a one-to-one mapping to the domain objects from the server-side Data Tier (the DTOs) while the UI domain objects are needed to support the different graphical views and interactions of the Mapping Client.

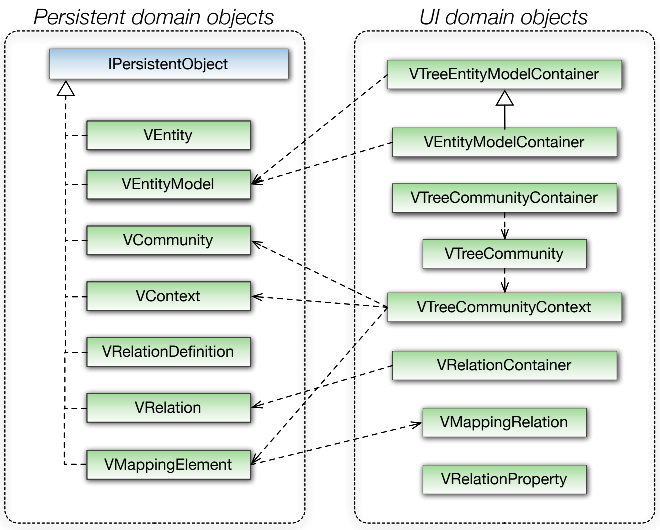


Figure 4.16: The domain objects in the Mapping Client Base.

View The view is the graphical representation of the model. This is what the users of the software see on their screen when they work with the tool. The figures are an almost one to one mapping to the models, aside from the more abstract container figures, which are in fact the canvases that contain all the other figures. Figure 4.17 gives an overview of the different figures.

The figures here will be further specialised in the plugins for the mapping and commitment phases. We will show screenshots of the figures when we discuss the plugins for these two phases in more detail.

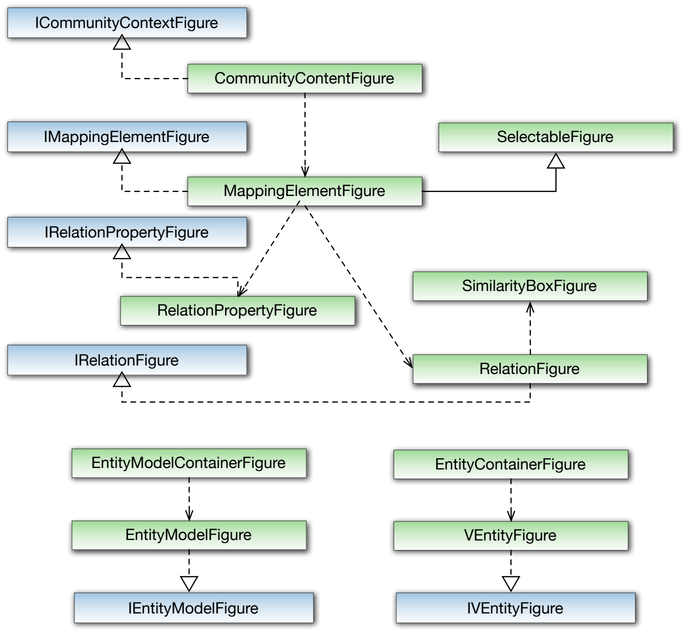


Figure 4.17: The different figures from the Base plugin.

4.3.2 Mapping phase

In this section, we present the part of the Mapping Client that enables the actor to transform heterogeneous models to our data model and create mappings between the entities of these models. The actor should furthermore be able to store these mappings on the server.

In section 4.1.2, we have reviewed the methodological steps that make up the mapping phase of our integration approach. Let us recap this process with Figure 4.18.

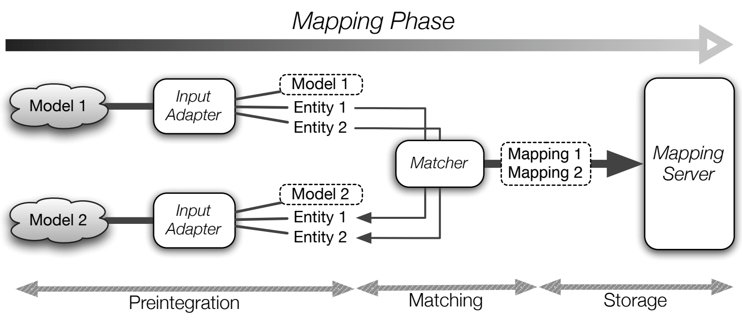


Figure 4.18: The different methodological steps of the Mapping Phase.

Figure 4.18 depicts the methodological steps of the Mapping Phase. Our Mapping Client will need to support each of these steps. This helps us to identify the following requirements for the mapping-phase part of the client:

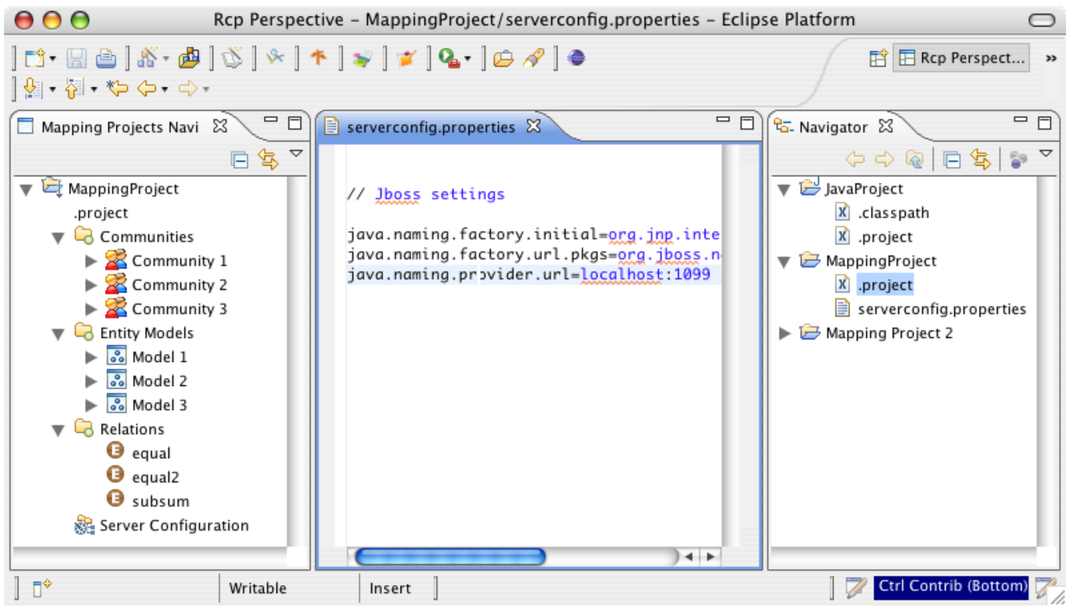
Preintegration The Mapping Client should support the conversion of specific model formats into our own data format. It should be able to convert these modal automatically with none to minimal interaction from the user of the tool.

Matching The client should make it easy for the user to create mappings between entities from different models. It should further allow the user to create these mappings in a certain context and

Storage The entities, models and mappings must be stored on the Mapping Server. This process should be as simple as possible for the user. The user of the tool should not have to know he is working remotely on the server. The line between the local and remote storage should be completely transparent.

Look & Feel

Now that we have identified the major requirements for our Client Tool to support the Mapping phase of our integration approach, we will present here the features and look & feel of our tool. We feel that this is best done using a number of relevant screenshots accompanied with the necessary explanations.



```
// Jboss settings

java.naming.factory.initial=org.jnp.inte
java.naming.factory.url.pkgs=org.jboss.n
java.naming.provider.url=localhost:1099
```

Figure 4.19: The traditional Eclipse-workbench with our Mapping Project Navigator.

Our first screenshot, depicted in figure 4.19, shows the traditional Eclipse-workbench. At the right, the standard Eclipse Navigator is shown. This navigator shows all the files stored on disk within the workbench. This navigator contains a standard Java Project and a general project named "MappingProject". Note that this project has only two files stored on the local disk:

.project and *serverconfig.properties*. The serverconfig.properties file, shown in the center of the screenshot, contains the server information which is used by the plugins to connect to our Mapping Server, more specifically, the Java Application Server.

In Eclipse, the user can add a *Nature* to a project. We have created or own nature for our Mapping Client and added it to the ”MappingProject” project. At the left side of the screenshot, our own Mapping Project Navigator is shown. This navigator shows only projects that have our ”Mapping Nature” added to them. Furthermore, it uses the information stored in the serverconfig.properties file to connect to the Mapping Server. The client receives from the Mapping Serve the Communities, Contexts, Mapping Elements, Models, Entities, and Relation Definitions; which are shown in the navigator.

One of the key-points of our approach and methodology is the support for context-aware and community-supported mappings. Therefore, we have designed the mapping-phase plugins in such a way the actor always works with the combination of a certain context and a certain community. The combinations of a context and community contains a unique set of mappings. If we would enable the actor to work within a context, the mappings could still belong to several different communities. This goes both ways of course: When an actor would work with a number of mappings belonging to a specific community, these mappings could still belong to several different contexts.

With this requirement in mind, we have developed a navigator on which the actor can browse different communities and contexts. A detailed screenshot of the navigator is depicted in figure 4.20.

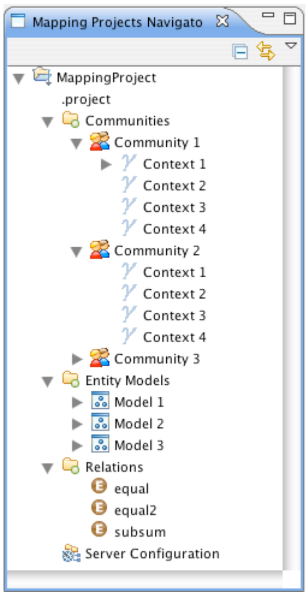


Figure 4.20: The traditional Eclpse-workbench with our Mapping Project Navigator.

Note that while in this screenshot, a context is placed below a community, this could just as well be the other way around. There is no connection between a context or community. The four contexts below *Community 1* and *Community 2* in the screenshot are the same contexts, however, the combination *Community 1* + *Context1* will result in a different set of mappings than the combination *Community 2* + *Context 1*. If the user double-clicks on a context in the navigator. An editor window opens in which he can create or edit the mappings within that community + context combination. The editor window has 4 tabs: a *Mappings-*, *Editor-*, *Context-*, and *Community-* tab.

The Mappings tab, as depicted in figure 4.21, shows the user all the mappings contained in the selected Community and Context combination. Each mapping must be read from left to right. In the center, the relation string is shown. The combination of this relation string and Context of the mapping points to a specific Relation Definition. The number to the left of the relation string is the strength of the relation. The number to the right of the relation string is the confidence measure, representing the confidence and thrust from the community in the mapping.

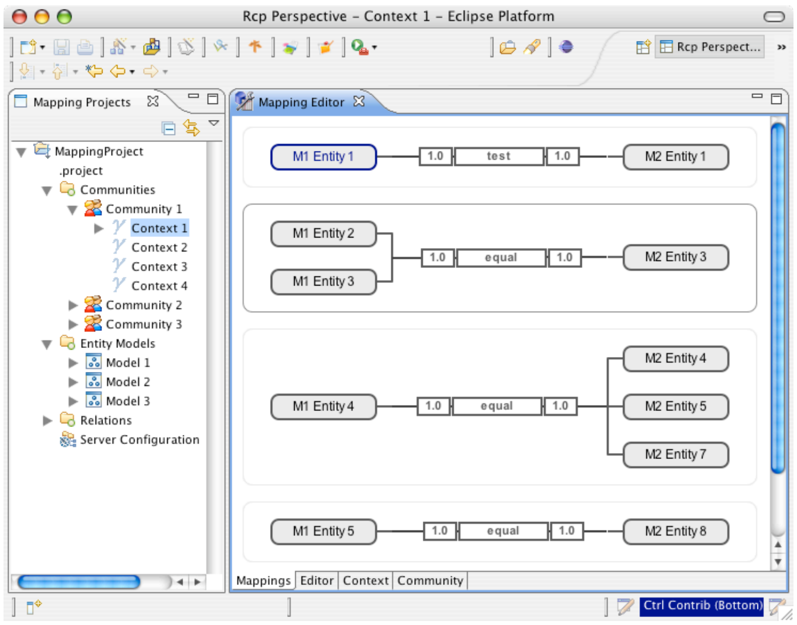
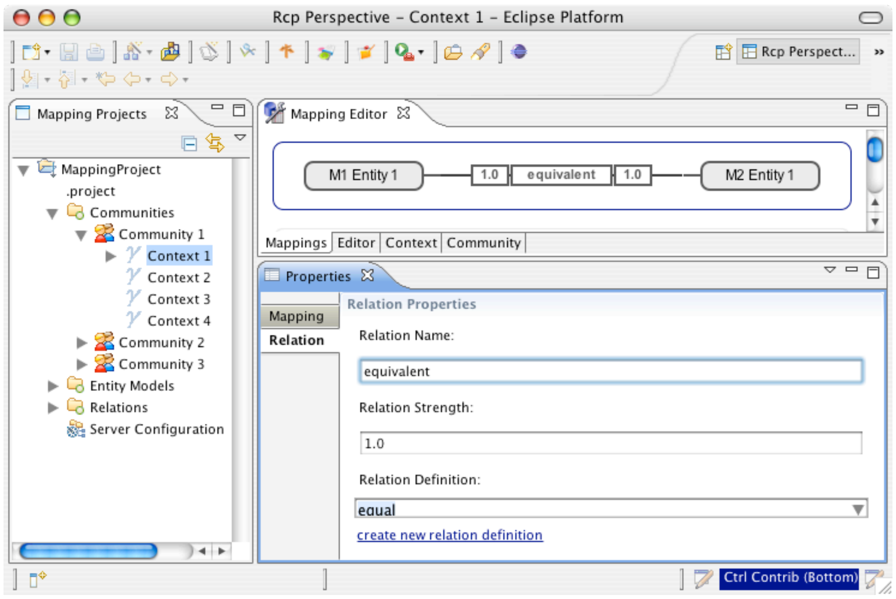


Figure 4.21: The Mapping Editor window with the *Mappings tab* selected.

The articulation of a relation string to a relation definition is shown in figure 4.22. The screenshot shows a mapping from "M1 Entity 1" to "M2 Entity 1" with the relation name "equivalent". The strength of the relation is 1.0 and the confidence in the mapping is also 1.0. The mapping belongs to the community "Community 1" and to the context "Context 1" as can be seen in the navigation view on the left. The properties view on the screenshot shows the relation properties of the mapping. It shows that the relation string "equivalent" points to the Relation Definition "Equal" within context "Context 1".



Relation Properties

Relation Name: equivalent

Relation Strength: 1.0

Relation Definition: equal

Figure 4.22: A selected mapping in the Mapping Editor with the properties view showing the Relation Definition assigned to the mapping.

While the Mapping tab is useful to view and edit existing mappings, it it not convenient to create new mappings using this view. Therefore, we have created a second view which can be accessed using the editor tab. In this view, the user can drag Models from the navigator into this view. The Models will be placed next to each other. The user can than easily create new mappings by dragging lines from an entity from one model to an entity of a second model. Figure 4.23 depicts a screenshot of the editor view. This editor view corresponds to the **Matching step** in the mapping phase.

The Client Tool must also support the **Preintegration step** from the Mapping Phase. The tool must enable to user to easily import different models and convert them into our own data integration model. Therefore, we have created input adapters to convert a specific data format into our data model. In the evaluation of our approach and tool in section 4.4, we will show how we have created an RDF/XML input adapter to convert an OWL-ontology into our data model.

The **Storage step** of the mapping phase is completely transparent to the user of the tool. As we have briefly discussed in the beginning of this section, the user is constantly working with remote data. The mappings, models and relations are never stored locally on disk. When the user performs the standard *Save* operation, the data is send and stored on the server.

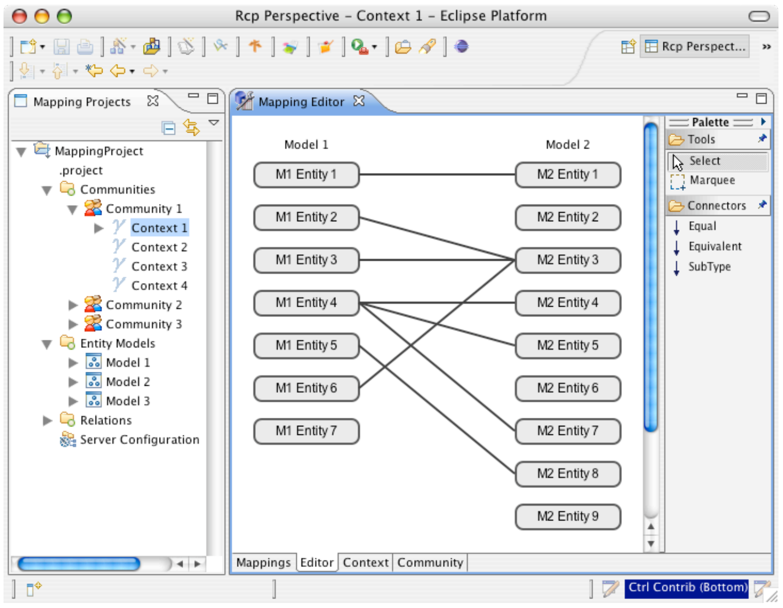


Figure 4.23: The Mapping Editor window with the *Editor tab* selected.

4.3.3 Commitment phase

In this section, we will present the part of our Mapping Client that enables the actor to commit a meaningful set of mappings to a final alignment. In section 4.1.2 we have reviewed the methodological steps that are needed in the commitment phase of our integration approach. The Client tool must provide support for the two methodological steps from the commitment phase as depicted in figure 4.24:

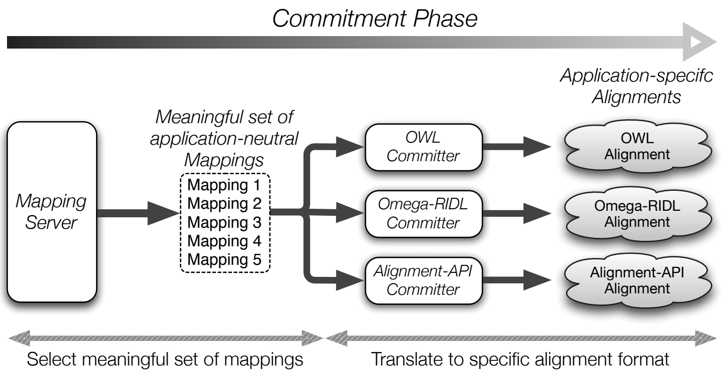


Figure 4.24: The methodological steps of the Commitment phase and the necessary tool-support.

Select meaningful set of mappings The actor must be able to browse the Mapping Server and find the relevant and meaningful mappings for the intended use of his alignment. He then must be able to select these relevant mappings to create the final alignment.

The Mapping Client supports this requirement with the same navigator and editor window we have presented in the Mapping phase. The user can select the mappings he wants to include in the final alignment by dragging them to a new editor window in which the final Alignment is shown. This window perspective and process is depicted in figure 4.25

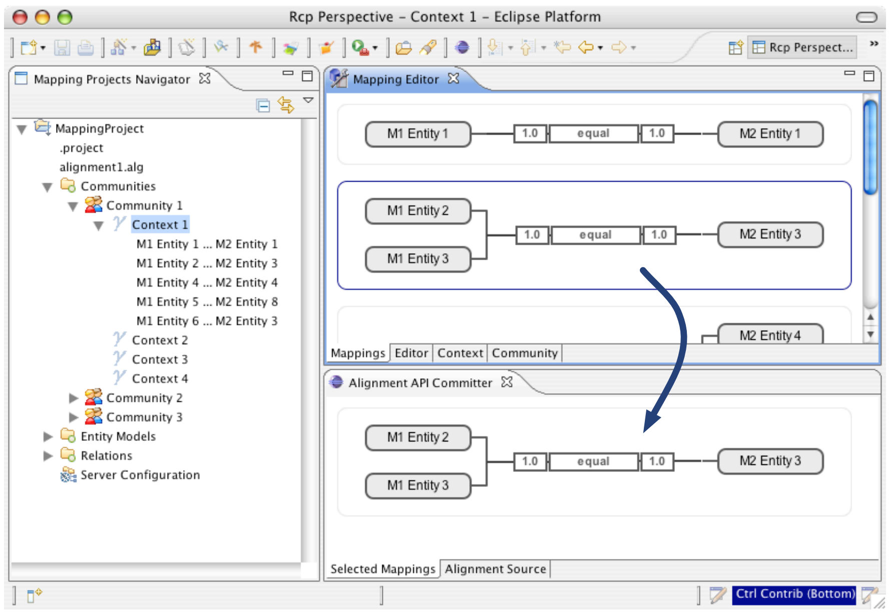


Figure 4.25: The perspective enabling the user to commit a meaningful set of mappings to a final alignment.

Translate to specific alignment format When the actor has selected the relevant, application-neutral mappings from the Mapping Server, these mappings must be translated into a specific alignment format. Therefore, a special committer must be available that interprets the mappings with their relation definitions and converts them into the specific alignment format.

The user of the Mapping Tool can select which kind of alignment he wants to create. Then, according to the chosen alignment format a corresponding committer plug-in will be loaded. The relations of the mappings he select to constitute the alignment will be interpreted by this specific committer and translated into the alignment format.

In figure 4.25, such a committer plug-in is shown. It enables the user to create an alignment in the *Alignment API* format. As shown on the figure, the alignment view has two tabs:

1. Selected Mappings: This tab shows the mappings the actor has chosen for this particular alignment. The mappings are shown in the same application-neutral manner from the Mapping phase.

2. Alignment Source: This tab shows the selection of meaningful mappings from the alignment translated into the specific alignment format chosen by the user.

4.4 Evaluation

To validate our methodology and approach, we have implemented an integration platform consisting of a server and client tool following the specification of the platform from chapter 3. In the previous sections, we have presented the Mapping Server and Mapping Client in further detail and have elaborated on how they support our proposed methodology.

In this section, we will further evaluate these tools using a more real-life scenario in order to do an preliminary evaluation of our thesis. It should be noted however that is is very difficult and out of the scope of this thesis to do a full evaluation of our community-driven, context-aware, and scalable integration approach. In order to be able to evaluate these elements of our approach, we would need a much larger test- and user-base. Both things take a significant amount of work to set up and accurately measure, therefore, we feel that this falls outside of the scope of this thesis.

As a preliminary evaluation of this thesis, we present a real-life scenario in which an alignment must be created between two heterogeneous ontologies. We have opted to integrate two RDF-based ontologies as these should sufficiently show the strengths and weaknesses of our platform.

In section 4.4.2 of this evaluation, we will precisely show how the user can create mappings between two heterogeneous models and store these mappings on the server, within a certain context and community. In section 4.4.3, we show how the user can browse the contexts and communities from the server and get a detailed view of every mapping contained in such a context/community pair. We furthermore show how the user can then select the relevant mappings and create an alignment between these two heterogeneous models in the *Alignment API* format.

4.4.1 Scenario

In this evaluation we will go through the entire process of integrating two similar, but heterogeneous RDF-based ontologies. We have chosen two citation ontologies:

1. UMBC Publication Ontology[4]: This is a publication ontology developed by the UMBC ebiquity research group from the university of Maryland. The authors use the ontology to publish their publications on their home-page conforming the Semantic Web principles.

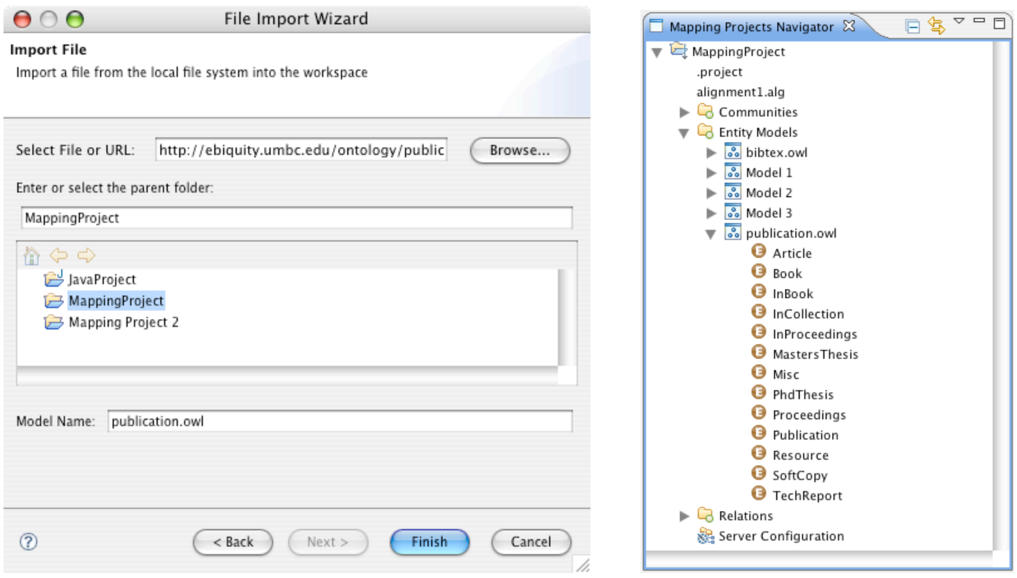
[4]http://ebiquity.umbc.edu/ontology/publication.owl

2. Bibtex citation Ontology[5]: The *bibTeX Definition in Web Ontology Language (OWL) Version 0.1* is developed by Nick Knouf, a researcher at the MIT Media Lab.

In this evaluation, we will integrate these two ontologies by creating an alignment from the Bibtex Ontology to the Publication Ontology. As we have repeatedly argued in this thesis, this alignment must be stored in a certain alignment format. In this case, we have opted for the *Alignment API*[26] as the format to store the final alignment in. The Alignment API is an API and implementation for expressing and sharing ontology alignments. It is mainly developed and maintained by Jerome Euzenat from INRIA Rhone-Alpes in France.

4.4.2 Mapping phase

To integrate these two heterogeneous ontologies using our platform and integration methodology, the models must first be translated into our own data model. We have written an input adapter that translates RDF documents into our Model/Entity model. How this process is supported by our Mapping Client is shown in figure 4.26.


Select File or URL: http://ebiquity.umbc.edu/ontology/public

Enter or select the parent folder: MappingProject

Model Name: publication.owl

Figure 4.26: Importing an RDF Ontology using an import wizard (left). The imported ontology is now available as a Model with its entities in the Mapping Navigator (right).

As shown on the above figure, we have created an RDF Import wizard which can be used to import RDF Ontologies. The user provides the wizard with the file or URL of the ontology and the wizard automatically translates it to our own Model/Entity model. When the wizard is finished, the new model is stored on the Mapping Server and can be seen in the Mapping Navigator.

[5]http://zeitkunst.org/bibtex/0.1

The next step from the Mapping phase is the actual matching of the elements from both models. In this case, the actor has to create mappings from the Bibtex ontology to the Publication ontology. The user can double click on an Community-Context combination in the Mapping Navigator. This will open a Mapping Editor window in which the user can create the mappings. Next, the user can drag an drop the two models to the Editor tab of the Mapping Editor. Having done that, the entities from both models will be displayed in this editor tab and the user can draw mappings from one entity to another. The result of these steps are depicted in figure 4.27 and 4.28.

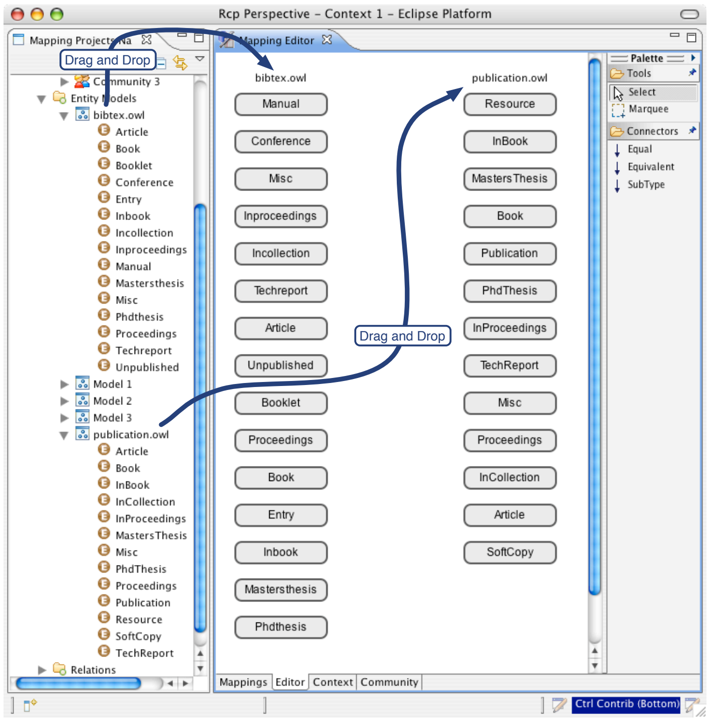


Figure 4.27: The result of opening an Editor View and drag and dropping the two models into the Editor tab.

For every mapping drawn in the Editor tab, a detailed mapping is shown in the Mapping tab. The Mapping tab shows all the mappings with the given Community and Context. When a user clicks on a mappings, he can specify its details. Most importantly, he can specify to what Relation Definition the Relation Name (string) must point to.

The process of selecting a mapping and pointing the ”equal” Relation Name to the *Equal* Relation Definition is depicted in figure 4.29.

When the user has created the mappings and saves the Editor window like in traditional applications, the mappings are stored on the Mapping Server.

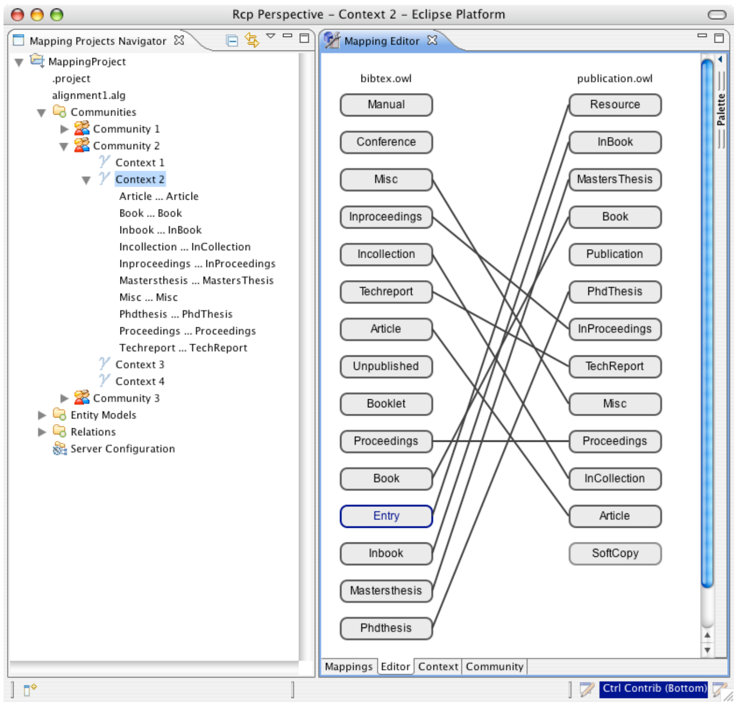


Figure 4.28: Drawing mappings between the entities of both models.

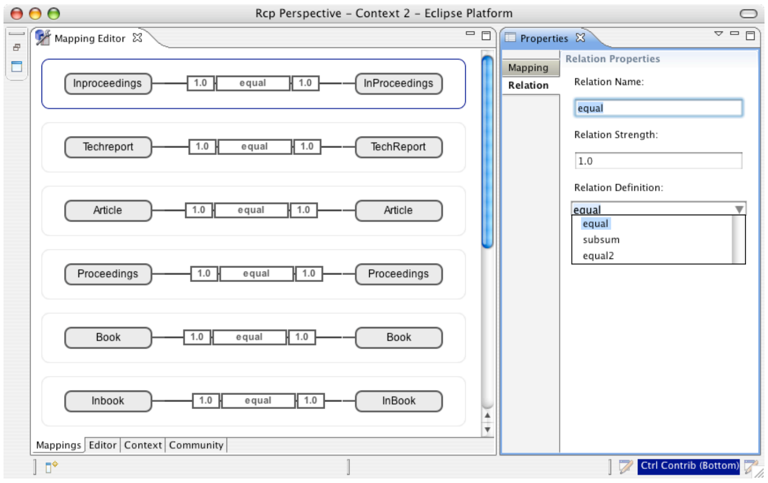


Figure 4.29: The Mapping tab where the detailed mappings are shown and their properties can be edited.

4.4.3 Commitment phase

Now that the mappings have been created and stored on the Mapping Server. An actor can use our Mapping Client to make the final alignment. This actor must not be the same as the one that has made the mappings as all mappings are available remotely through the Mapping Server.

We have named the process of creating a final alignment by selecting a meaningful set of mappings and transforming them to the required alignment format *committing an alignment*. As we have shown, the client can support many different committers depending on the alignment format needed. In this scenario, we will use the Alignment API Committer. This committer will enable the user to select the mappings from the Mapping Server and drag them to the alignment. In the same time, the comitter will interpret the Relation Definition from these mappings and translate the mappings into the Alignment API format. This process is depicted in figure 4.30 and 4.31.

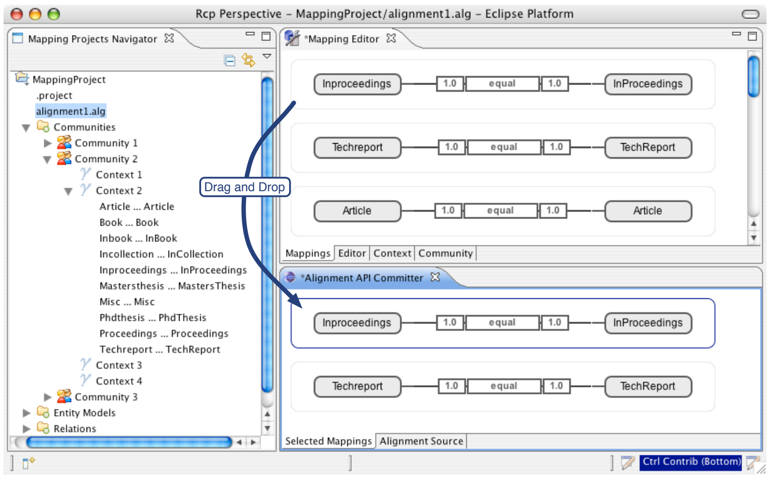

Rcp Perspective – MappingProject/alignment1.alg – Eclipse Platform
Mapping Projects Navigator
MappingProject
.project
alignment1.alg
Communities
Community 1
Community 2
Context 1
Context 2
Article ... Article
Book ... Book
Inbook ... InBook
Incollection ... InCollection
Inproceedings ... InProceedings
Mastersthesis ... MastersThesis
Misc ... Misc
Phdthesis ... PhdThesis
Proceedings ... Proceedings
Techreport ... TechReport
Context 3
Context 4
Community 3
Entity Models
Relations
*Mapping Editor
Inproceedings 1.0 equal 1.0 InProceedings
Techreport 1.0 equal 1.0 TechReport
Article 1.0 equal 1.0 Article
Mappings | Editor | Context | Community
Drag and Drop
*Alignment API Committer
Inproceedings 1.0 equal 1.0 InProceedings
Techreport 1.0 equal 1.0 TechReport
Selected Mappings | Alignment Source
Ctrl Contrib (Bottom)

Figure 4.30: The *Selected Mappings* tab from the Alignment API Committer plug-in.

The Alignment API Committer has two tabs:

1. Meaningful Mappings tab: This tab will show all the mappings contained in the alignment, in the same application-neutral way as they are shown in the Mapping Editor. Selecting the relevant and meaningful mappings from the Mapping Server is done by dragging them from the Mapping Editor to this tab window.

2. Alignment Source tab: In this tab, the final alignment is shown. This result from interpreting the selected mappings and converting them to the specific alignment format. As shown in figure 4.31, the transformation of the two mappings:

$$''Community2'', ''Context2'' : \{Inproceedings\}\ equal_{1,1}\ \{Inproceedings\}$$

and

$$''Community2'', ''Context2'' : \{Techreport\}\ equal_{1,1}\ \{Techreport\}$$

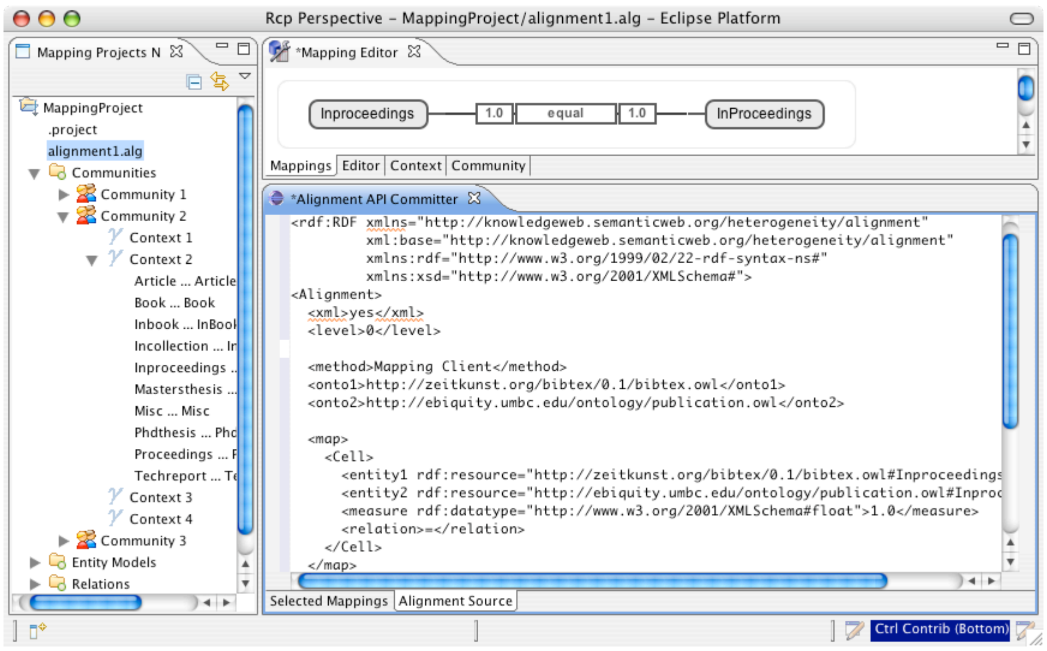
is transformed into the following alignment conform the Alignment API format:

```
<?xml version="1.0" encoding="utf−8" standalone="no"?>
<rdf:RDF xmlns="http://knowledgeweb.semanticweb.org/heterogeneity/alignment"
         xml:base="http://knowledgeweb.semanticweb.org/heterogeneity/alignment"
         xmlns:rdf="http://www.w3.org/1999/02/22−rdf−syntax−ns#"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
<Alignment>
  <xml>yes</xml>
  <level>0</level>

  <method>Mapping Client</method>
  <onto1>http://zeitkunst.org/bibtex/0.1/bibtex.owl</onto1>
  <onto2>http://ebiquity.umbc.edu/ontology/publication.owl</onto2>

  <map>
    <Cell>
      <entity1 rdf:resource="http://zeitkunst.org/bibtex/0.1/bibtex.owl#Inproceedings"/>
      <entity2 rdf:resource="http://ebiquity.umbc.edu/ontology/publication.owl#Inproceedings"/>
      <measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
      <relation>=</relation>
    </Cell>
  </map>

  <map>
    <Cell>
      <entity1 rdf:resource="http://zeitkunst.org/bibtex/0.1/bibtex.owl#TechReport"/>
      <entity2 rdf:resource="http://ebiquity.umbc.edu/ontology/publication.owl#TechReport"/>
      <measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
      <relation>=</relation>
    </Cell>
  </map>
 </Alignment>
```



```
<rdf:RDF xmlns="http://knowledgeweb.semanticweb.org/heterogeneity/alignment"
         xml:base="http://knowledgeweb.semanticweb.org/heterogeneity/alignment"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
<Alignment>
  <xml>yes</xml>
  <level>0</level>

  <method>Mapping Client</method>
  <onto1>http://zeitkunst.org/bibtex/0.1/bibtex.owl</onto1>
  <onto2>http://ebiquity.umbc.edu/ontology/publication.owl</onto2>

  <map>
    <Cell>
      <entity1 rdf:resource="http://zeitkunst.org/bibtex/0.1/bibtex.owl#Inproceedings
      <entity2 rdf:resource="http://ebiquity.umbc.edu/ontology/publication.owl#Inproc
      <measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
      <relation>=</relation>
    </Cell>
  </map>
```

Figure 4.31: The *Alignment Source* tab from the Alignment API Committer plug-in.

4.5 Conclusions

We started this chapter by presenting the high-level architecture of our integration platform. The platform consists of a Remote layer and a Client layer. In the client layer, a separation can be made between the two steps of our integration methodology: the *Mapping phase* and the *Commitment phase*. We identified the different steps that need to be followed in each phase and how these steps are translated into requirements for the Mapping Client. In the second section of this chapter, we discussed the Remote Layer of our platform in further detail. We presented the technology stack on which the remote layer is based: a three-tiered J2EE architecture. We further elaborated on each of the two remote tiers, discussing the implementation details using UML class and sequence diagrams and elaborating on the patterns we adopted. In the third section, we discussed the Mapping Client in further detail. We presented the architectural foundation of the tool. Next, we discussed how the Mapping Client supports the two methodological steps of our integration methodology. In the last section, we performed a preliminary evaluation of our platform and methodology using a real-life scenario. We showed how two heterogeneous ontologies can be integrated using our two-step methodology.

5 Conclusions

In this chapter we reflect on the questions we have asked ourselves in this thesis and on how we answered those questions. Finally we list what we feel are the most important points of future work

5.1 Research Questions and Objectives

In this thesis we have asked ourselves the following questions and answered them in distinct chapters of this thesis:

1. What kinds of information heterogeneity exist and where does this heterogeneity come from? *(Chapter 2)*
2. What is the existing work on integration methodologies, algorithms and frameworks? *(Chapter 2)*
3. Can we propose a methodology based on the notions of collaboration, context-awareness and reuse to attain a scalable integration approach? *(Chapter 2)*
4. What would the advantages and disadvantages of such an approach be? *(Chapter 2)*
5. Can we develop an platform that implements and support this methodology? *(Chapter 3)*

In the next Section we describe how we have answered these questions and met our objectives.

5.2 Results and Contributions

We now present the results of the thesis. We split up the presentation of our results in three parts following the research methodology we adopted:

5.2.1 Background

We answered our first two questions in this chapter. We discussed the different forms of semantics and how these can lead to heterogeneity. We further discussed STARLab's DOGMA approach to ontology engineering as this framework forms the basis of many of the properties of our own approach and methodology for information integration. Next, we discussed the use of semantics in practice, more specifically, we presented the Semantic Web initiative and the OMG's model driven architecture (MDA) and ontology definition model (ODM) initiatives. We ended this chapter by providing the reader with a background on semantic integration: We presented the common terminology we used throughout the rest of this thesis; we discussed the different forms and levels in which heterogeneity can appear; we presented a state of the art on ontology matching, a very important part of the integration process, and finally provided an overview of the most important existing integration frameworks.

5.2.2 Approach & Methodology

We started this chapter by introducing the two cornerstones of our proposed integration approach: Mapping reuse and application-specific mappings. We identified several scenarios that are problematic for the existing integration frameworks, but are much better handled using our methodology. In the second section of this chapter, we elaborated in more detail on our methodology. We discussed in detail the two phases of our integration approach: The Mapping phase and the Commitment phase. We discussed the tree most significant characteristics of our approach and methodology: Uniform and context-aware mappings, and a community-driven mapping process. In the third section we defined the precise semantics of the conceptual data model used by our methodology: We provided exact definitions of the Context, the Community, the Mapping, the Model, the Entity, and the Relation Definition used in our approach. To conclude the section, we elaborated on the motivation for our choices. We showed how our approach has significant benefits over existing approaches and how it accomplishes the goals we have set for this thesis.

5.2.3 The Platform

We started this chapter by presenting the high-level architecture of our integration platform. The platform consists of a Remote layer and a Client layer. In the client layer, a separation can be made between the two steps of our integration methodology: the *Mapping phase* and the *Commitment phase*. We identified the different steps that need to be followed in each phase and how these steps are translated into requirements for the Mapping Client. In the second section of this chapter, we discussed the Remote Layer of our platform in further detail. We presented the technology stack on which the remote layer is based: a three-tiered J2EE architecture. We further elaborated on each of the two remote tiers, discussing the implementation details using UML class and sequence diagrams and elaborating on the patterns we adopted. In the third section, we discussed the Mapping Client in further detail. We presented the architectural foundation of the tool. Next, we discussed how the Mapping Client supports the two methodological steps of our integration methodology. In the last section, we performed a preliminary evaluation of our platform and methodology using a real-life scenario. We showed how two heterogeneous ontologies can be integrated using our two-step methodology.

5.3 Discussion

In this thesis, we have presented our two-step methodology that splits up the traditional integration process into two completely separated phases. In the first phase, the Mapping phase, heterogeneous models are matched and mappings are created between corresponding entities of these models. The mappings are stored on a remote server in an application-neutral manner. We further introduced a community and contextual dimension for each mapping, which could, among others, be used as a grouping mechanism providing an extra level of scalability as the number of similar mappings can grow very large. In the second phase of our proposed methodology, the Commitment phase, a final, application-specific alignment is created by the actor. He creates this alignment by selecting from the Mapping Server a meaningful set of mappings depending on the intended use of the alignment. The tool will aid him in automatically interpreting the selected application-neutral mappings and translating them into the application-specific integration format.

As we argued in chapter 3, this methodology brings many of the advantages we found were lacking in current integration frameworks, but very necessary in our increasingly information-dependant, fast-moving, collaborative information society. By introducing an application-neutral, persistent state between the two steps of our methodology, mappings can be efficiently reused which enables a new level of scalability and efficiency:

- a mapping can be created once and used many times;
- when the model evolves, new mappings must only be created for the new and changed entities of the model;
- because the process of creating an application-specific alignment from a large number of existing application-neutral mappings is very efficient and requires only a small effort, users must no longer create general alignments, but can create alignments that are specifically tailored for their intended use without much overhead;

- storing application-neutral, conceptual mappings on a remote server enables a community of stakeholders to reason over, express their confidence in, and govern over these mappings. In this way, the mapping creation and management process encourages collaboration and can become community-driven, which may result in more and higher-quaility mappings.

While we argue that these advantages may result from adopting our methodology, this thesis misses a real case-study evaluating our claims. This could be one argument against this thesis. However, putting together such an evaluation study would need a large and controlled user base. This is very difficult to set up and as we did not have such resources to our disposal, we believe this falls outside the scope of this thesis and remains future work.

We have however done a preliminary evaluation of the platform we developed. We have shown that two real ontologies can be successfully integrated using our platform by following our methodology.

5.4 Future Work

In this thesis, we have presented our novel methodology and platform that should allow for a more efficient, scalable, and collaborative integration process. However, more research should be conducted in order to make this a robust, repeatable and well-tested process. We acknowledge future work is needed on two aspects of our approach: The methodology needs more research and evaluation to become more robust and well-defined. On the other side, better tool support will need to be developed to aid the users and increase the efficiency of our two-step approach. We identify a number of issues for which future work could further increase the quality and efficiency of the integration process:

Integration Methodology

With respect to our integration methodology, we identify the areas where future work could directly strengthen the methodology:

Collaboration More research is needed on the way users and communities of users interact to come to a larger number of high-quality mappings. Certain policies of thrust and confidence must be introduced and social network analysing techniques may be appropriate to evaluate the quality and thrust in a mapping. This may lead to a complete community-driven integration process which could further increase the scalability and efficiency of the platform, and quality of the resulting alignments.

Evolution More research may be necessary to develop a robust methodology to support evolution of models and hence the mappings between them. Problems like versioning and governance will need to be looked into.

Integration Platform

An integration platform that supports our methodology is almost just as important as the methodology itself. It is critical that the actor has a powerful tool suite available that will enable him to integrate heterogeneous models using our methodology as efficiently as possible.

Integrating existing matching frameworks Many matching algorithms and frameworks have already been developed. These should be integrated in our platform as the Mapping phase of our methodology has a matching step itself: The actors must match the corresponding entities from the heterogeneous models to create the mappings. When matching algorithms would be integrated in the client, they could propose the mappings to the user. This could lead to a semi-automated mapping phase which would further increase the efficiency of our approach. When the entire process becomes community-driven, a fully automated Mapping phase might be possible: The mappings could be created fully automatically after which the community could express their confidence and thrust in the best mappings. This would fully exploit the *Wisdom of Crowds* and *Collective Intelligence* we referred to in our introduction.

Interpretation of application-neutral mappings In the Commitment phase of our methodology, the application-neutral mappings must be translated into application-specific mappings conform the given alignment format. It is important that the tool supports the user as much as possible in this process. This translation should be as transparent as possible for the user. Several interpreters will need to be developed to understand the conceptual relation from the mappings on the server and translate them into the corresponding relations or rules in the different alignment formats.

Bibliography

[1] Ontology definition metamodel. Request for proposal, Object Management Group, March 2003.

[2] Topic maps data model. Technical Report 13250-2, ISO/IEC, December 2005.

[3] Meta object facility (mof) core specification. OMG Available Specification Version 2.0, Object Management Group, January 2006.

[4] Unified modeling language. Technical Report Version 2.0, Object Management Group, March 2006.

[5] D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with coma++. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 906–908, New York, NY, USA, 2005. ACM Press.

[6] J. Barwise and J. Seligman. *Information Flow: the Logic of distributed systems*, volume Cambridge Tracts in Theoretical Computer Science 44. Cambridge University Press, 1997.

[7] M. Benerecetti, P. Bouquet, and C. Ghindini. Contextual reasoning destilled. *Journal of Theoretical and Experimental Artificial Intelligence*, 12(3):279–305, 2000.

[8] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American 284(5)*, pages 34–43, 2001.

[9] J. Bézivin and O. Gerbé. Towards a precise definition of the omg/mda framework. In *Automated Software Engineering*, San Diego, USA, November 2001.

[10] J. D. Bo, P. Spyns, and R. Meersman. Assisting ontology integration with existing thesauri. In Z. T. e. a. Meersman R., editor, *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE (part I)*, volume 3290 of *LNCS*, pages 801 – 818. Springer Verlag, 2004.

[11] H. Boley. The rule markup language: Rdf-xml data model, xml schema hierarchy, and xsl transformations. In *Web Knowledge Management and Decision Support: 14th International Conference on Applications of Prolog, INAP*, volume 2543/2003 of *Lecture Notes in Computer Science*, pages 5–22, Tokyo, Japan, October 2001. Springer Berlin / Heidelberg.

[12] P. Bouquet, M. Ehrig, J. Euzenat, E. Franconi, P. Hitzler, and et al. D2.2.1 specification of a common framework for characterizing alignment. KnowledgeWeb Deliverable, February 2005.

[13] D. Brickley and R. V. Guha. Rdf vocabulary description language 1.0: Rdf schema. W3c recommandation, W3C, February 2004.

[14] A. Brown. An introduction to model driven architecture - part 1: Mda and today's systems.

[15] R. Colomb, K. Raymond, L. Hart, P. Emery, C. Welty, G. T. Xie, and E. Kendall. The object management group ontology definition metamodel. In F. Ruiz, C. Calero, and M. Piattini, editors, *Ontologies for Software Engineering and Technology*. Springer, 2006.

[16] S. Crawley, S. Davis, J. Indulska, S. McBride, and K. Raymond. Meta-meta is better-better. In *Workshop on Distributed Applications and Interoperable Systems (DAIS)*, 1997.

[17] J. De Bo, P. Spyns, and R. Meersman. Creating a "dogmatic" multilingual ontology infrastructure to support a semantic portal. In Z. T. e. a. R. Meersman, editor, *On the Move to Meaningful Internet Systems 2003: OTM 2003 Workshops*, volume 2889 of *LNCS*, pages 253 – 266. Springer Verlag, 2003.

[18] P. De Leenheer and A. de Moor. Context-driven disambiguation in ontology elicitation. In P. Shvaiko and J. Euzenat, editors, *Context and Ontologies: Theory, Practice and Applications*, volume WS-05-01 of *AAAI Technical Report*, pages 17–24, Pittsburg USA, 7 2005. AAAI Press.

[19] P. De Leenheer, A. de Moor, and R. Meersman. Context dependency management in ontology engineering: a formal approach. *Journal on Data Semantics VIII*, LNCS(4380):26–56, 2007.

[20] P. De Leenheer and R. Meersman. Towards a formal foundation of dogma ontology: part i. Technical Report STAR-2005-06, VUB STARLab, Brussel, 2005.

[21] A. de Moor. Ontology-guided meaning negotiation in communities of practice. In P. Mambrey and W. Graether, editors, *Proceedings of the Workshop on the Design for Large-Scale Digital Communities at the 2nd International Conference on Communities and Technologies (C&T 2005)*, pages 21 – 28, Milano, 2005.

[22] A. de Moor, P. De Leenheer, and R. Meersman. DOGMA-MESS: A meaning evolution support system for interorganizational ontology engineering. In *Proceedings of the 14th International Conference on Conceptual Structures, (ICCS 2006), Aalborg, Denmark*, Lecture Notes in Computer Science. Springer-Verlag, 2006.

[23] M. Dean and G. Schreiber. Owl web ontology language reference. W3c recommandation, Word Wide Web Consortium (W3C), February 2004.

[24] H. H. Do and E. Rahm. Coma - a system for flexible combinations of schema matching approaches. *Proceedings of the Very Large Data Bases conference (VLDB)*, pages 610–621, 2001.

[25] K. Duddy. Uml2 must enable a family of languages. *Communications of the ACM*, 45(11):73 – 75, 2002.

[26] J. Euzenat. An api for ontology alignment. In *The Semantic Web, ISWC 2004: Third International Semantic Web Conference*, volume 3298/2004 of *Lecture Notes in Computer Science*, pages 698–712. Springer Berlin / Heidelberg, 2004.

[27] B. Ganter and R. Wille. *Formal Concept Analysis: mathematical foundations*. Springer, 1999.

[28] O. Gerbé and B. Kerhervé. Modeling and metamodeling requirements for knowledge management. In J. Bézivin, J. Ernst, and W. Pidcock, editors, *Proceedings of OOPSLA Workshop on Model Engineering with CDIF*, Vancouver, Canada, October 1998.

[29] F. Giunchiglia and P. Shvaiko. Semantic matching. *The Knowledge Engineering Review Journal (KER)*, 18(3):265–280, 2003.

[30] F. Giunchiglia and P. Shvaiko. S-match: an algorithm and an implementation of semantic matching. In *Proceedings of the European Semantic Web Symposium (ESWS)*, pages 61–75, 2004.

[31] F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence*, 57(2-3):323–390, 1992.

[32] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[33] N. Guarino and P. Giaretta. Ontologies and knowledge bases: Towards a terminological clarification. In N. Mars, editor, *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, pages 25 – 32, Amsterdam, 1995. IOS Press.

[34] P. Hayes. Rdf semantics. Technical report, W3C, 2004.

[35] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabel, B. Grosof, and M. Dean. Swrl: A semantic web rule language combining owl and ruleml. W3c member submission, World Wide Web Consortium, 2004.

[36] ISO/IEC. Information technology - common logic (cl) - a framework for a family of logic-based languages. Official ISO FCD Draft 24707, ISO/IEC, April 2006.

[37] E. J. Towards a principled approach to semantic interoperability. In G.-P. A., G. M., and S. H, editors, *Proceedings IJCAI 2001 Workshop on ontology and ifnormation sharing*, pages 19 – 25, 2001.

[38] Y. Kalfoglou, B. Hu, D. Reynolds, and N. Shadbolt. Capturing representing and operationalising semantic integration. 6th month deliverable: Semantic integration technologies survey, The University of Southampton, Hewlett Packard Laboratories@Bristol, 2005.

[39] M. Klein. Combining and relating ontologies: an analysis of problems and solutions. In A. Gomez-Perez, M. Gruniger, H. Stuckenschmidt, and M. Ushold, editors, *Proceedings IJCAI 2001 Workshop on ontology and ifnormation sharing*, WA USA, 2001.

[40] G. Klyne and J. J. Carroll. Resource description framework (rdf): Concepts and abstract syntax. Technical report, W3C, 2004.

[41] G. E. Krasner and S. T. Pope. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *J. Object Oriented Program.*, 1(3):26–49, 1988.

[42] O. Lassila and R. R. Swick. Resource description framework (rdf) model and syntax specification. Technical report, World Wide Web Consortium, January 1999.

[43] D. Lenat. The dimensions of context space.

[44] J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with cupid. In P. M. G. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. T. Snodgrass, editors, *Proceedings of 27th International Conference on Very Large Data Bases*, pages 49–58, 2001.

[45] J. Madhavan, P. A. Bernstein, A. Doan, and A. Halevy. Corpus-based schema matching. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, pages 57–68, Washington, DC, USA, 2005. IEEE Computer Society.

[46] N. Martin. *Wine Journal: Chateau Cheval-Blanc*. `http://www.wine-journal.com/cheval.html`, 2004.

[47] D. McComb. *Semantics in Business Systems*. Morgan Kaufmann Publishers Inc., 2003.

[48] R. Meersman. The ridl conceptual language. *Control Data DMRL*, 1982.

[49] P. Mika. Ontologies are us: A unified model of social networks and semantics. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(1):5–15, 2007.

[50] G. A. Miller. Wordnet: a lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.

[51] I. Niles and A. Pease. Towards a standard upper ontology. In C. Welty and B. Smith, editors, *FOIS '01: Proceedings of the international conference on Formal Ontology in Information Systems*, pages 2–9, New York, NY, USA, 2001. ACM Press.

[52] N. Noy and M. A. Musen. Anchor-prompt: Using non-local context for semantic matching. In *In Proceedings of IJCAI workshop on Ontologies and Information Sharing*, page 63ñ70, 2001.

[53] C. Ogden and I. A. Richards. The meaning of meaning: A study of the influence of language upon thought and of the science of symbolism. *8th ed. 1923. Reprint New York: Harcourt Brace Jovanovich*, 1923.

[54] E. R. Philip A. Bernstein. Data warehouse scenarios for model management. In *Lecture Notes in Computer Science*, volume 1920, pages 1–15. Springer Berlin / Heidelberg, 2000.

[55] G. Pór. Blog of collective intelligence, 2007.

[56] A. J. Pretorius. Lexon visualisation: Visualising binary fact types in ontology bases. Unpublished MSc Thesis, Brussels, Vrije Universiteit Brussel, 2004.

[57] E. Prud'hommeaux and A. Seaborne. Sparql query language for rdf. W3c candidate recommendation, Word Wide Web Consortium (W3C), 2006.

[58] M. R. and J. M. An architecture and toolset for practical ontology engineering and deployment: the dogma approach. Technical Report 06, STAR Lab, Brussels, 2002.

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

[60] R. Reiter. Towards a logical reconstruction of relational database theory. *In Brodie, M., Mylopoulos, J., Schmidt, J. (eds.), On Conceptual Modelling*, pages 191–233, 1984.

[61] B. Rosario. Classification of the semantic relations in noun compounds.

[62] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics*, IV, 2005.

[63] P. Spyns and R. Meersman. A survey on ontology alignment and merging. OntoBasis Deliverable #D1.5, STAR Lab, Brussel, 2003.

[64] P. Spyns, R. Meersman, and M. Jarrar. Data modelling versus ontology engineering. *SIGMOD Record Special Issue on Semantic Web, Database Management and Information Systems*, 31(4):12–17, 2002.

[65] J. Surowiecki. *The Wisdom of Crowds*. Anchor, August 2005.

[66] M. Ushold and M. Gruninger. Ontologies: Principles, methods and applications. In *The Knowledge Engineering Review*, 1996.

[67] H. Wache, T. Vogele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hüber. Ontology-based integration of information – a survey of existing approaches. In *Proceedings of the IJCAI-Workshop Ontologies and Information Sharing*, pages 108–117, Seattle, WA, 2001.

[68] A. V. Zhdanova, R. Krummenacher, J. Henke, and D. Fensel. Community-driven ontology management: Deri case study. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 73–79, Washington, DC, USA, 2005. IEEE Computer Society.