DIP

Data, Information and Process Integration with Semantic Web Services

FP6 – 507483

Deliverable

D1.5

Framework for Representing Ontology Networks with Mappings that Deal with Conflicting and Complementary Concept Definitions

Livia Predoiu
Francisco Martín-Recuerda
Axel Polleres
Cristina Feier
Adrian Mocan
Jos de Bruijn
Fabio Porto
Doug Foxvog
Kerstin Zimmermann

Executive Summary

The aim of this deliverable is to present a framework for representing the interrelationships between distributed ontologies within a heterogeneous ontology network and dealing with mismatching or even conflicting concept definitions. For this purpose, we take advantage of related efforts within the SEKT and Knowledge Web projects. We employ a mapping language which we developed in a joint effort with the SEKT project to describe the connection between ontologies. Such a mapping language allows us to deal with complementary concept definitions.

Our approach to reason with the ontology network starts with importing the ontologies into a common format that eliminates the heterogeneity. We consider WSML Flight as an appropriate language for the common format because it shows an optimal trade-off between expressivity and tractability. After translating the ontologies into WSML Flight, we define mappings based on the mapping language defined in a joint effort with the SEKT project. Afterwards we insert the parts of the ontologies and the mappings that are relevant to the query into a so-called *reasoning space*. We detect the parts of the ontologies and the mappings that are relevant to the query by means of a Datalog dependency graph of the ontologies and the mappings. After setting up the reasoning space, the ensuing reasoning process takes account of inconsistencies that may have arisen due to mismatching or conflicting concept definitions in the reasoning space.

Document Information

IST Project Number	FP6 – 507483	**Acronym**	DIP
Full Title	Data, Information, and Process Integration with Semantic Web Services		
Project URL	`http://dip.semanticweb.org/`		
Document URL			
EU Project Officer	Brian Macklin		

Deliverable	**Number**	1.5	**Title**	Framework for Representing Ontology Networks with Mappings that Deal with Conflicting and Complementary Concept Definitions
Work Package	**Number**	1	**Title**	Ontology Reasoning and Querying

Date of Delivery	**Contractual**	M12	**Actual**	$Date$: 2004/12/1412 : 54 : 29
Status	Version $Version$		final ☐	
Nature	prototype ☐ report ☒ dissemination ☐			
Dissemination Level	public ☒ consortium ☐			

Authors (Partner)	Livia Predoiu, Francisco Martín-Recuerda, Axel Polleres, Cristina Feier, Kerstin Zimmermann (UIBK), Adrian Mocan, Doug Foxvoug (NUIG), Fabio Porto (EPFL)			
Resp. Author	Livia Predoiu		**E-mail**	livia.predoiu@deri.org
	Partner	UIBK	**Phone**	+43 (512) 507-6111

Abstract (for dissemination)	The overall goal of this deliverable is to present a framework for representing the interrelationship between distributed ontologies and dealing with mismatching or even conflicting concept definitions. For this purpose, we make advantage of related efforts within the SEKT and Knowledge Web projects.
Keywords	Ontology Networks, Ontology Mappings, Inconsistencies

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
2004/08/31	1	Axel Polleres	Initial Outline
2004/10/24	2	Livia Predoiu	Submitted Outline
2004/11/13	3	Livia Predoiu	First Draft
2004/12/22	4	Livia Predoiu	Final Version

Project Consortium Information

Partner	Acronym	Contact
National University of Galway	NUIG	Prof. Dr. Christoph Bussler Digital Enterprise Research Institute (DERI) National University of Ireland, Galway Galway Ireland E-mail: chris.bussler@deri.ie Tel: +353 91 512460
Fundacion De La Innovacion.Bankinter	Bankinter	Monica Martinez Montes Fundacion de la Innovation. BankInter, Paseo Castellana, 29 28046 Madrid, Spain Email: mmtnez@bankinter.es Tel: 916234238
Berlecon Research GmbH	Berelcon	Dr. Thorsten Wichmann Berlecon Research GmbH, Oranienburger Str. 32, 10117 Berlin, Germany E-mail: tw@berlecon.de Tel: +49 30 2852960
British Telecommunications Plc.	BT	Dr. Thorsten Wichmann Berlecon Research GmbH, Oranienburger Str. 32, 10117 Berlin, Germany E-mail: tw@berlecon.de Tel: +49 30 2852960
Swiss Federal Institute of Technology, Lausanne	EPFL	Prof. Karl Aberer Distributed Information Systems Laboratory École Polytechnique Féderale de Lausanne Bât. PSE-A 1015 Lausanne, Switzerland E-mail : Karl.Aberer@epfl.ch Tel: +41 21 693 4679
Essex County Council	Essex	Mary Rowlatt, Essex County Council, PO Box 11, County Hall, Duke Street, Chelmsford, Essex, CM1 1LX, United Kingdom. E-mail: maryr@essexcc.gov.uk Tel: +44 (0)1245 436524
Forschungszentrum Informatik	FZI	Andreas Abecker Forschungszentrum Informatik Haid-und-Neu Strasse 10-14 76131 Karlsruhe Germany E-mail: abecker@fzi.de Tel: +49 721 96540

Institut für Informatik, Leopold-Franzens Universität Innsbruck	UIBK 	Prof. Dieter Fensel Institute of computer science University of Innsbruck Technikerstr. 25 A-6020 Innsbruck, Austria E-mail: dieter.fensel@deri.org Phone: +43 512 507 6488/6485
ILOG SA	ILOG 	Christian de Sainte Marie 9 Rue de Verdun, 94253, Gentilly, France E-mail: csma@ilog.fr Tel: +33 1 49082981
inubit AG	inubit 	Torsten Schmale, inubit AG, Lützowstraße 105-106 D-10785 Berlin, Germany E-mail: ts@inubit.com Tel: +49 30726112 0
Intelligent Software Components, S.A.	iSOCO 	Dr. V. Richard Benjamins, Director R&D Intelligent Software Components, S.A. Pedro de Valdivia 10 28006 Madrid, Spain E-mail: rbenjamins@isoco.com Tel. +34 913 349 797
Net Dynamics Internet Technologies GmbH u. Co KG	Net Dynamics 	Peter Smolle Net Dynamics Internet Technologies GmbH u. Co KG Prinz-Eugen-Strasse 68-70 A-1040 Wien, Austria E-mail: peter.smolle@netdynamicstech.com Tel.: +43 1 503982615
The Open University	OU 	Dr. John Domingue Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK E-mail: j.b.domingue@open.ac.uk Tel.: +44 1908 655014
SAP AG	SAP 	Dr. Elmar Dorner SAP Research, CEC Karlsruhe SAP AG Vincenz-Priessnitz-Str. 1 76131 Karlsruhe, Germany E-mail: elmar.dorner@sap.com Tel: +49 721 6902 31
Sirma AI Ltd.	Sirma 	Atanas Kiryakov, Ontotext Lab, - Sirma AI EAD, Office Express IT Centre, 3rd Floor 135 Tzarigradsko Chausse, Sofia 1784, Bulgaria E-mail: atanas.kiryakov@sirma.bg Tel.: +359 2 9768 303

Tiscali Österreich GmbH	Tiscali 	Dr Dieter Haacker Tiscali Österreich Gmbh. Diefenbachgasse 35, A-1150 Vienna, Austria E-mail: Dieter.Haacker@at.tiscali.com Tel: +43 1 899 33 160
Unicorn Solution Ltd.	Unicorn 	Jeff Eisenberg Unicorn Solutions Ltd, Malcha Technology Park 1 Jerusalem 96951, Israel E-mail: Jeff.Eisenberg@unicorn.com Tel.: +972 2 6491111
Vrije Universiteit Brussel	VUB 	Carlo Wouters, Starlab- VUB Vrije Universiteit Brussel Pleinlaan 2, G-10 1050 Brussel, Belgium E-mail: carlo.wouters@vub.ac.be Tel.: +32 (0) 2 629 3719

Table of Contents

1 Introduction

On the DIP project web page[1] the following description for its objective is given: "the project aims at providing an automated environment for delivering a wide variety of e-commerce and business-to-business services and applications. Such services and applications will communicate and interoperate in a world composed of Web-accessible programs and databases, and interface wirelessly with many smart devices and sensors." In order to realize this objective, DIP strongly relies on the use of ontologies as a formal basis for describing domains in general and web services and user goals in particular. The spectrum of e-commerce and business-to-business applications can only be handled if scalability, distribution and heterogeneity issues are considered by DIP as real constraints on the use of Ontologies.

In this deliverable we present a framework for representing heterogeneous ontology networks and consider reasoning within them. Because the network is heterogeneous the ontologies are not all described in the same language. In this deliverable we consider only the OWL variants and the WSML variants as ontology languages[2]. We restrict ourselves to these languages because OWL on the one hand is the prevalent ontology description language for the Semantic Web and WSML on the other hand is the language for describing Semantic Web Services developed in the course of the DIP project.

The different ontologies may describe similar domains that overlap. Therefore, the information that is spread in the network has to be integrated in order to be able to deal with it in a cohesive manner. For this purpose we have to introduce mappings between the ontologies. The mappings can be described by means of a mapping language. Such a mapping language should consist of built-in constructs and mapping patterns that can be combined to yield more expressive phrases. This holds especially because in this way the mappings remain easily comprehendible. To this end, we employ the mapping language that has been devised in a joint effort with the SEKT project [28, 27]. By describing the relationships between the ontology elements, we can complement the definitions of ontologies. Though, there may also be correspondences between the ontologies that are conflicting because the individual ontologies in the network have been devised by different ontology engineers that made different modelling decisions. E.g. imagine two different ontology engineers that have to model an automobile domain, then the one may model the manufacturing process and introduce a general concept called car while the other may model the domain of a vendor and introduce a concept for each type of car the vendor sales.

When dealing with networks of ontologies, we want to gather new implicit information from the combination of these ontologies. By using mappings that connect the different ontologies in the network, we can resolve heterogeneities and apply similar reasoning techniques like the ones that we would use with single ontologies. The prerequisite is that all information is encoded in the same logical language. Therefore, the ontologies need to be preprocessed before assigning the mappings between them and translated into a common format, i.e. into the same logical language. Furthermore, because of possible conceptual inconsistencies within the network that cannot be resolved by mappings alone or are even introduced by the mappings, we need to consider

[1] http://dip.semanticweb.org/

[2] Note that we preprocess all ontologies by translating them into a common format in order to be able to find mappings and in order to enable standard reasoning

how to reason with inconsistencies. For this purpose we make use of an approach that has been devised in the course of the SEKT project [59]. This approach is iteratively looking for a consistent subset of an overall inconsistent theory for answering a query.

The remainder of this deliverable is organized as follows: In the following section we will set up a convention about the terminology used throughout the deliverable.

Chapter 2 presents an overview of the Semantic Web (Services) ontology languages that we consider within this deliverable, namely the OWL variants and the WSML variants.

Chapter 3 provides an overview of ontology modularization and ontology networks. Chapter 4 introduces an integrated classification for heterogeneities that can occur within ontology networks together with some examples.

Chapter 5 presents the mapping language which has been devised in collaboration with the SEKT project. We also present a direct translation of this high-level mapping language to the Web Service Modeling Language WSML which will be further detailed in Deliverable [63]. More precisely, we provide a translation into WSML Flight with function symbols which semantically corresponds to Datalog with function symbols. One reason for choosing the WSML Flight variant of the WSML language for the grounding of the mapping language is that it shows the optimal trade-off between expressivity and tractability. Note that a grounding of the mapping language to OWL can be found in [27].

Chapter 6 provides an overview of different ways to deal with inconsistencies within logics. We provide a general discussion about inconsistencies and discuss how to identify inconsistencies. We also survey literature about how to debug inconsistent ontologies, e.g. by means of belief revision. We briefly present the approach for dealing with inconsistencies devised within the SEKT project [59] which we adapt for our needs in chapter 7.

Chapter 7 introduces a framework for reasoning within ontology networks. There we describe how we proceed the ontologies and the mappings in order to be able to reason with them. As already mentioned, we translate the ontologies into a common format which in this case is WSML Flight. Then we set up the mappings. We identify then by means of the dependency graph the parts of the ontologies and the mappings that are relevant to the query. These parts are inserted into an initially empty so-called *reasoning space*. Within this reasoning space we reason by means of an inconsistency reasoner.

Finally, chapter 8 provides conclusions and an outlook on further research work and possible ways to implement this framework.

1.1 Terminology

This section sets up a convention for the terminology that is used throughout the deliverable. For a more in-depth discussion of the fundamental concepts concerning **Ontology Merging**, **Ontology Aligning** and **Relating Ontologies**, see for instance [32, 37].

- **Term**
  A *term* is a name of entity of an ontology, e.g. the name of a concept or an instance.

- **Ontology**
  An *ontology* is a 4-tuple $< C, R, I, A >$ [91]. At this, C is a set of concepts, R is a set of relations, I is a set of instances and A is a set of axioms and A is a set of axioms. An ontology has to be specified in a logical language, i.e. a formal language with a well-defined semantics. An *Ontology component* is an element c that is either a concept, a relation, a instance or an axiom and is defined in an ontology. Binary relations or distinguished subsets of binary relations are often called properties or attributes.

- **Instance Base**
  Instances are a part of an ontology. We deem it as useful to distinguish between an ontology that describes a set of instances and the set of instances described by an ontology. I.e., we regard an ontology as a schema and look at the set of instances independently from the ontology. In the following, we call the set of instances an *Instance Base*.

- **Ontology Language**
  An *Ontology Language* is a language that can be used to represent an ontology. In general, the Resource Description Framework Schema (RDFS) [18] and the Web Ontology Language (OWL) [36] are ontology languages that are used in the area of the Semantic Web in general. In the scope of this deliverable we will deal with OWL and the Web Service Modelling Language (WSML) [31]. A survey of these languages can be found in section .

- **Ontology Mediation**
  Ontology Mediation is the process of establishing relationships between two or more different ontologies. Without loss of generality, we consider only mediation between two ontologies. It is straightforward to scale this up to more than two ontologies. Ontology mediation can mainly be done in two different ways. Either mappings between the elements of the individual ontologies are discovered and specified or the ontologies are merged. Therefore, correspondences between the elements of the individual ontologies have to be detected in order to merge the ontologies appropriately.
  In this deliverable we dealing with Ontology Mediation based on mappings.

- **Ontology Mapping**
  An *Ontology Mapping* M is a (declarative) specification of the semantic overlap between two ontologies O_S (the source ontology) and O_T (the target ontology). This mapping can be *one-way* or *two-way*. In a one-way mapping it is specified how terms in O_T can be expressed using terms from O_S in an irreversible way. In contrast, a two-way mapping is reversible.

- **Mapping Language**
  A *mapping language* is the language that is used to represent the mapping M. We discriminate between a mapping language and a specification of similarities between ontologies: While the latter consists generally of a probability expressing the level of confidence of the similarity of entities, the former specifies and concretises the relationship between the entities in the ontologies. Often, mapping languages allow arbitrary transformations between ontologies and arbitrary value transformations and are based on rule-based formalisms.

- **Mapping Pattern**
  A *Mapping Pattern* is a predefined built-in mapping that can be used to construct a larger mapping. Typically, a mapping pattern is a mapping that occurs very often. Mapping Patterns are not often used in current ontology mediation approaches. Nonetheless, they are very helpful for the specification of ontology mappings. The advantage of using mapping patterns lies in a more concise representation of mappings and this yields in a greater understandability and fewer errors in the creation and usage of mappings.

- **Matching**
  We define *ontology matching* as the process of discovering similarities between two source ontologies. The result of a matching operation is a specification of similarities between two ontologies. Ontology matching is done through application of the *Match* operator (cf. [88]). Any schema matching or ontology matching algorithm can be used to implement the *Match* operator, e.g. [38, 48, 74, 78].

  We adopt here the definition of *Match* given in [88]: "[*Match* is an operation] which takes two schemas [or ontologies] as input and produces a mapping between elements of the two schemas that correspond semantically to each other".

- **Ontology Merging**
  Ontology Merging is the process of creating one new ontology from two or more ontologies. Then, the new ontology will unify and replace the original ontologies. This often required considerable adaptation and extension, especially when dealing with very big ontologies.
  When building up the new ontology one has to deal with redundant information. It is most suitable to start with one ontology and convert and add the other ontology or ontologies. The process of deleting and adding of ontology entities to an ontology is very error-prone and susceptible to inconsistencies.

- **Ontology Aligning**
  Ontology Aligning is the process of bringing ontologies into mutual agreement. Here, the ontologies are kept separate, but at least one of the original ontologies is adapted such that the conceptualization and the vocabulary match in overlapping parts of the ontologies. However, the ontologies might describe different parts of a domain in different levels of detail.

- **Relating Ontologies**
  Relating Ontologies is the process of specifying how the concepts in different ontologies are related in a logical sense. This means that the original ontologies have not changed, but that additional axioms describe the relationship between the concepts. According to [37] leaving the original ontologies unchanged often implies that only a part of the integration can be done, because major differences require adaption of the ontologies. Notice that *Mapping Ontologies* is just another term for relating Ontologies because it is defined in the same way, namely to specify the relationship between two ontologies.

- **Ontology Mismatches**
  An *Ontology Mismatch* is a difference between two ontologies that contradicts the semantic correspondence between the ontology entities at hand. The kinds

of mismatches that can and do occur have to be identified in order to resolve these mismatches in the mapping or the merge of the ontologies.

- **Ontology Conflicts**
  An *Ontology Conflict* is an ontology mismatch that requires a sophisticated solution or can not be solved at all.

- **Local Components**
  A *Local Component*, with respect to an ontology O_i, is an entity c of type concept,relation, axiom or instance that appears in O_i as part of its language.

- **external components**
  An *external component*, in respect to an ontology O_i, is a local component of an ontology O_j, with $i \neq j$.

- **Ontology space**
  Let I be a set of indexes, standing for a set of URIs of ontologies. For instance, I contains http://www.w3.org/2002/O7/owl. An *Ontology Space* is a family of ontologies $< i, O_i >_{i \in I}$, such that for every ontology component c either c is a local component of O_i or is an external component defined in O_j, $i \neq j$ and $i, j \in I$.

2 Ontology and mapping languages

This chapter presents an overview on the ontology languages that are considered as possible languages for the ontologies within this deliverable, i.e. on OWL and WSML. We We restrict ourselves to these two languages because they are the most important languages for the Semantic Web and for Semantic Web Services, respectively. Below we describe their peculiarities. This chapter also considers briefly the peculiarities of mapping languages in general.

2.1 General Considerations

In the following, we state some general considerations concerning languages that are suitable for describing mappings between ontologies.

2.1.1 Expressivity and Language Mismatches

In the simplest case we assume that we only want to establish mappings between ontologies using the same ontology language.

In a more complicated scenario you might also want to deal with networks of different ontologies using languages of different expressivity. Here, when mapping between a language with richer language features to a language with less expressivity information loss might be inevitable. The other way round should be less problematic. Even worse there might be semantic differences with languages which only overlap semantically, i.e. either language has some features not expressible in the other one and vice versa.

2.1.2 Decidability

Mappings are typically expressed by some form of rules. Here, logic programming style rules, offering the expressivity of a powerful query language are a natural choice. There is a whole bunch of literature on combining rule languages with ontology languages (cf. [73, 39, 51, 81, 41]). Note that a mapping language might need much more features than expressible in simple Horn rules. While Horn-rules alone are sufficient to express conjunctive queries on Relational Databases, for several advanced mappings additional features such as an exhaustive set of pre-defined functions, aggregations, negation, etc. might be needed. For instance, functions to concatenate strings or extracting substrings (let's say for extracting or combining a first and last name from/to a full name) or for aggregating attributes of a class in one ontology which in the other is only given by its instances (let's say, mapping to an instance representing a department with some average salary in one ontology while the single employees with their indivual salaries and associated department are specified in another one, etc.). These features are offered by powerful database and XML query languages such as SQL or XQuery, but are missing in almost all logical languages.

In most of the approaches to combine rules with ontology languages decidability issues are a major concern, as first discussed in [73]. Particularly, inferencing in Ontologies expressed in Description Logics style might become undecidable when rules are simply added to the language without restrictions, an example of such an undecidable combination of rules with an ontology language is the rule language SWRL [57]

as an extension of the Web ontology language OWL (more specifically its OWL DL fragment).

On the contrary the ontology language itself might already be undecidable which is for instance the case for OWL full. So, in order to retain decidability, one has to find a reasonable tradeoff between expressivity of the mapping language and the ontology language.

In the context of this deliverable we will focus on two particular ontology languages, namely the Web Ontology Language (OWL) [36] proposed as a recommendation of the World Wide Web Consortium and the Web Service Modelling Language (WSML) [31] proposed by the WSML working group of the SDK cluster.

2.2 OWL

OWL has three dialects, namely OWL Lite, OWL DL and OWL Full, each of which with increasing less expressivity. Since OWL DL is the most expressive decidable variant of these languages, we will focus on OWL DL here.

OWL DL can be seen as an alternate notation for the Description Logic language $\mathcal{SHOIN}(\mathbf{D})$ [56].

The most important aspect of Description Logics, from a modeler's point of view, is that roles are specified independently from concepts. This allows for automatically determining the class of an individual based on the roles specified for that individual. This is conceptually a nice feature, but not always intuitive, because humans tend to think in roles belonging to concepts and individuals specified directly as belonging to specific concepts. Furthermore, property restrictions are interpreted as class descriptions.

OWL has different syntaxes, the most prominent being the RDF/XML [8] syntax. Furthermore, there is a more readable abstract syntax [85] usable for the OWL DL subset of OWL.

OWL DL has a direct model-theoretic semantics [85]. However, this semantics is hard to understand compared with standard Description Logic semantics. It was shown that entailment in OWL DL (checking whether one ontology is logically entailed by another) can be reduced to satisfiability checking in Description Logics [56]. Since all interesting reasoning tasks can be reduced to entailment, it is safe to say that OWL DL is a notational variant of a Description Logic. Furthermore, it was shown that Description Logic is a subset of the First-Order Logic [15], thus OWL DL is also a notational variant for a subset of First-Order Logic.

While basically being definable as a Description Logic variant OWL adds some peculiarities such as for instance the distinction between abstract and concrete values: OWL allows as instance identifiers URLs which represent objects, and literals which represent atomic values of a concrete domain, which are no further decomposable.

As for mediation and combination of different ontologies, OWL supports a simple import mechanism. This feature allows for physical modularization of the ontologies but has no consequences on the semantics: The importing ontology is semantically equivalent to the union of all (recursively) imported ontologies. Subclass, class equivalences or subproperty relations as well as equivalence/disjointness between instances can be defined within the importing ontology in order relate concepts, relations and

instances of the imported ontologies. This allows for the formalization of simple mappings.

2.3 WSML

Recently proposed, the WSML family of languages for modelling ontologies and Web service descriptions aims (in its ontology modeling facilities) on tackling several problems in OWL DL.

As opposed to OWL, WSML does not commit to the Description Logics style of modeling but is more grounded in the Logic Programming paradigm which is an orthogonal decidable fragment of First-Order Logic. Similar to OWL, WSML defines several dialects. WSML-Core determines the least common subset of Datalog, that is Logic Programming without function symbols, with the DL world.

WSML-Flight (based on a Logic Programming based variant of OWL, called OWL Flight [34]) extends this with some more features from Logic Programming, namely default negation and database-like integrity constraints.

Due to their construction WSML-Core and WSML Flight both combine naturally with rule extensions.

WSML-Rule which is currently under development will be an extension of WSML-Flight in the direction of Logic Programming. The language will capture several extensions such as the use of function symbols and possibly extensions based on HiLog [24] and Transaction Logic [14], which are required for rich Web Service discovery [64]. Another possible extension for WSML-Rule is disjunction in the head of the rule, as in Disjunctive Datalog [40].

WSML-DL is an extension of WSML-Core which fully captures the Description Logic ($SHOIN$), which underlies the (DL species of the) Web Ontology Language OWL. The language can be seen as an alternate syntax for OWL DL, based on the WSMO conceptual model.

Finally, WSML Full which hasn't been defined yet should provide a common umbrella for OWL and the other WSML variants which amounts to full First-Order Logic with nonmonotonic extensions. Note that WSML is intended to serve not only as an ontology language but to describe in general all relevant features of ontologies, web services, and mediators. However, when speaking about WSML here, we only restrict ourselves to its ontology modeling facilities.

The normative syntax for WSML is a human readable syntax as defined in [30]. It is planned to also provide an RDF/XML exchange syntax for all WSML variants like for OWL, as well as a more concise native XML syntax which is closer to its human readable syntax.

Like OWL, WSML provides facilities for importing other ontologies. Additionally WSML includes the concept of mediators which define the interoperability between different WSMO components. As for interoperability between ontologies, in WSML it is also allowed to import mediators which link other ontologies not directly but via some conversion, defined in the capability of a so-called mediation service. The mapping language defined in this deliverable shall serve as the basis for the formal description of such mediators.

2.3.1 Relation to OWL

OWL has several drawbacks, as pointed out in [20]:

1. *Abstract vs. concrete values* are treated differently, in the sense that OWL does use the unique name assumption for concrete values (i.e. literals), whereas it does not for abstract values.

2. *Restrictions vs. constraints*: Given a property of a certain class in an ontology, OWL allows only for restrictions of the range to another class in the sense that membership of the range class is inferred for any filler of the respective property. Similarly, if the cardinality of a property is restricted, whenever more role fillers are specified equality between two or more of therm is inferred. This can often lead to unintuitive results. Particularly, in OWL is not possible to define constraints on the instance data of an ontology wrt. to a local knowledge base. This feature would be particularly useful when importing legacy data or mapping between ontologies.

3. *Dytatypes*: OWL has a very limited support for datatypes and pre-defined predicates over simple datatypes such as Strings or Numbers. As mentioned above pre-defined predicates and datatypes, (for instance for string concatenation or extraction of substrings) might be necessary for sufficiently describing mappings involving value conversions resolving heterogeneities between different ontologies. Extensions of OWL such as OWL-E [84] and the pre-defined predicates suggested for SWRL [57] take this into account, but have not yet found their way into the OWL proposal. WSML includes a set of pre-defined predicates listed in [30] which shall be supported by any implementation.

When defining mappings between OWL and languages treating these issues differently such as OWL these semantic differences have to be taken into account.

Further details on the Ontology language used will be given in Deliverables 2.7 [63].

3 Ontology Modularization and Ontology Networks

This chapter presents the concept of Ontology Networks (ON). In an ON context, independently developed, distributed and heterogeneous ontologies are seen by applications as an homogeneous reasoning space. (Sub)networks of ontology modules may be intensionally designed to enable reasoning using only applicable parts of an ontology, or to ease integration with external ontologies. Differences in language and semantic heterogeneity among ontologies are hidden from users through an ontology module abstraction, which also offers transparent access to distributed entities. Ontology modules offer an elegant abstraction for managing a ontology space allowing for the source ontologies of entities in an ontology network to be completely transparent to user applications.

3.1 Modularization Overview

Modularization is a well-known technique for managing complexity of many areas of computer science, such as algorithms, software engineering, programming languages and databases. In algorithms, the design technique of *divide and conquer* synthesizes the principle that a large problem is better understood if divided into small parts.

In the ontology context, the modularization approach finds its motivation in two main aspects [96]:

- **distribution**: In a distributed context, such as the Semantic Web, ontologies are created by different groups for the purpose of covering particular domains and are expressed using a variety of terms and ontology languages. Applications that would benefit by access to the various ontologies relating to a given subject require a uniform method of accessing entities in the set of ontologies they reference.

  Direct access or merging of ontologies is not possible because of the different languages used. Even in the case of ontologies using the same language, the terminology and forms defined in different ontologies for the same meanings would be incredibly unlikely to completely match.

  Modularization of these disparate ontologies breaks them down into smaller chunks which can individually be more easily mapped to other ontologies covering similar concepts. Once such modular ontologies are publicly available, future ontologies become more likely to be built upon existing ontology modules, cutting down on the heterogeneity of the growing ontology network.

  Applications would be tied into existing ontology modules and the original heterogeneity of the ontology network becomes less noticeable as entities that originally were defined in different languages and using different schemes become accessible in the same system using the same language.

- **scalability**: Another important aspect for modularization is the size of the component ontologies. Some existing ontologies include tens of thousands of concepts and relations and hundreds of thousands of instances. At some point large ontologies can cause performance issues for reasoners.

Modularization is an important approach for dealing with this problem. Instance databases can be separated from concept data bases and only used as necessary. For example, a reasoner trying to find the shortest route from Innsbruck to Berlin need not consider the tens of thousands of cities in Asia, Africa, or the Americas.

Downloading huge ontologies should not be a major issue since such downloads are not expected to be performed at run time; rather when reasoning is expected to be performed using an ontology, that reasoning is performed by a system which already has access to the ontology.

This section builds upon the motivation presented above to propose an ontology modularization approach to deal with distributed and heterogeneous ontologies as a single common view given for reasoning. The remainder of this chapter details the proposed approach and is structured as follows. Subsection 3.2 raises some requirements for the modularization approach. Next, subsection 3.3 introduces a proposal for dealing with ontology modules in DIP. Finally, in subsection 3.4, some related work is discussed.

3.2 Requirements for Ontology Modularization

Ontology modularization is the process of separating a single ontology - whether currently existing or in the process of being created - into multiple smaller interconnected ontologies each focused on some narrower topic. The modules are selected such that an appreciable amount of the necessary reasoning can be performed using only a subset of the modules, thereby reducing the combinatorics of the reasoning process.

In order to specify a model for ontology modularization, one needs to consider how to define and use ontology modules, the correctness criteria for individual modules and for composing multiple modules and the basis for distinguishing ontology modules in general. Issues such as ontology language do not arise since the designer of a single ontology would not consider using multiple languages within the same ontology.

The result of ontology modularization is typically a set of ontology modules, the more specific of which import some of the more general modules. A sensible structure for such a modularized ontology would be to have one or a few very general "upper-level" ontologies which define very general concepts and would be imported by the more specific ontology modules.

The basic intuition behind the design of individual ontology modules is simplification. The individual modules concern a narrowly defined topic (limited by either breadth or depth). As such they have few statements and the statements they include are expected to show regularities. Each module has a standard interface - way that it it accesses components of other ontologies and ways that other ontologies access its components.

Each ontology module is expected to be self-consistent (even after importing the other ontologies that it necessarily imports. Inconsistency between two modules, neither of which imports the other is no problem so long as no other module imports both of them. See Chapter 6 for a discussion of dealing with conflicts within ontologies.

Reasoning in an ontology network would produce different answers to the same query depending upon the portion(s) of the network that were accessed to answer the query. Thus, care must be taken so that the modules that specify the necessary and sufficient terms, rules, and appropriate instance-level data are included in the context

in which a query is posed. Chapter 7 presents a general framework for reasoning with ontology networks.

3.3 Dealing with Ontology Modules in DIP

In the context of DIP, ontology modules are far more likely to be pre-existing ontologies being brought together, as considered by *Wonderweb* or *C-OWL*, than being a decomposition of a large pre-existing ontology such as *Cyc*, or being built up modularly from scratch.

An ontology module becomes a tool for abstracting semantics from a section of an ontology network as well as merely a local ontology used by a single process. Its main component is a local ontology with its internal import links to other ontologies. The set of entities specified in an ontology module's local ontology and the external ones it imports defines a reasoning space.

In addition, an ontology module includes a description written in a natural language, has a single associated ontology language, one or more associated topics, and an associated URI as a unique global identifier.

Ontology modules are linked to each other by ontology-to-ontology mediators (ooMediators), which are pieces of software responsible for translating components from the form of one ontology to that of another.

A formal ontology module definition is presented below.

Definition 2.1 (Ontology Module): An ontology module (hereafter module) is defined as $OM =< id, D, T*, LN, O >$, where:

- id - is an unique ontology module identifier (e.g. a URI);
- D - is the module description written in natural language;
- $T*$ - is a set of zero or more topics expressed in terms defined in O (or in an ontology imported by O);
- LN - is the ontology language used in the ontology;
- O - is the local ontology;

The expression of mappings from external components is the basic principle used in extending the reasoning space with external components. Such extension can be achieved either by expanding an existing ontology module or by creating a new module, importing the ontology of the module to be extended into the new ontology, and bringing in the new terminology. In either case for each (desired) term in the source ontology that has no corresponding term in the new ontology, a term is created in the target ontology; and whether or not the term is new, a mapping is created in the ooMediator that relates the two ontology modules. Once the ooMediator is completely defined, sentences from the source ontology that can be mapped using the ooMediator into sentences in the target ontology.

3.4 Related work

The field of ontology modularization has become a research topic as large and distributed ontologies have been developed. In this section, we will summarize some of the recent work in this field.

C-OWL [16] aims to support the scale-up of large and distributed ontologies by specifying an ontology as the result of linking autonomous developed ontologies. In C-OWL, a set of independent ontologies form a context OWL space, where each ontology O_i is enriched by components defined on external ontologies O_j mapped according to O_i interpretation. A bridging language is specified for defining mappings (coordination) with some predefined associations like: subsumption (c-owl:into), equivalence(c-owl:equivalence), containement(c-owl:onto), disjunction (c-owl:incompatible) and intersection (c-owl:compatible). A consequence of C-OWL approach is that by defining their own mapping rules, a local ontology may get inconsistent but would not affect the consistency of the remaining ontologies.

The *wonderweb* project [96] presents an approach for developing ontology networks. The requirements suggested in the work comprise loose coupling of existing ontologies, self-containment, and integrity. Loose coupling describes links between pre-existing ontologies that share topics but may be implemented in different languages – instances in one ontology may be imported into another without bringing over everything from the source ontology. Self-containment recognizes that reasoning can proceed independently in any of the pre-existing ontologies, but can benefit by importing additional information. Integrity is associated with correct reasoning in the presence of autonomous modules. Based on these principles, the work proposes a modularization approach where self-contained modules are cross connected through materialized views expressed as conjunctive queries. The connection of modules thereof is obtained by defining an equivalence relation between a concept in a module (local ontology) and the result of evaluating a query on an external ontology. The result produced by evaluating the connection view is materialized into the local ontology as new axioms, contributing to the definition of a self-contained module. A procedure for managing updates in a external ontology definition is also proposed. The authors argue that the proposed mapping language, expressed as conjunctive queries, is more expressive than standard methods of directing referencing objects in an external ontology, such as adopted in OWL import strategy [7].

The Cyc Project has divided its ontology of well over 100,000 classes, relations, and individuals into a hierarchical network of "contexts", each of which is, in effect, an ontology that inherits the statements in the contexts above it in the hierarchy. Initially, the contexts were not defined systematically. However, as the number of contexts grew into the thousands, a set of principles for subdivision of a massive ontology into these contexts was desired.

Doug Lenat proposed that contexts vary along a limited number of dimensions (with some of those having subdimensions) and that by specifying the dimensions for a statement or query, its placement in context space could be reasonably accurately determined [70]. Every context (module) is proposed to have values defined along all of the defined dimensions. The dimensions help to determine which contexts inherit from which other contexts. The superstructure envisioned allows specific information to be located so as to inherit definitions and information from a large number of related contexts, while a much larger number are ignored. Meanwhile, the definitions

of the general classes are made in contexts where much more of the knowledge base is inaccessible.

The major proposed dimensions are time, type of time, geographical location, type of place, culture, topic, and granularity, with other dimensions dealing with how the context is dealt with by users (hypothetical, justification and argument preference, modality, sophistication/security). One dimension used but not defined as such in the paper is that of general term definition/ theory/ data. For example, a general economic ontology could define the basic economic concepts upon which everyone agrees, separate ontologies which inherit those terms would have the rules for competing economic theories, and other ontologies would have data about the real world and inherit (at least) from the term definition contexts.

Ontology modularization is not only useful for re-organizing and improving existing ontologies but is seen as a strategy for developing new ontologies [60].

3.5 Summary

This chapter presented an approach for ontology mapping and networks. Although ontology networks in the future may include ontologies written in different languages, in the context of this paper we concentrate on networks that are homogeneous in terms of the language used.

In the ontology space context, each ontology has associated metadata, (such as language, URI, and topics) and may normally import other ontologies in the ontology net, and as such is considered an ontology module, for which a formal definition was presented.

An ontology module abstracts physical aspects regarding its entities definitions from reasoners, providing a common view for applications. By doing this, a module presents an opportunity for ontology management, where issues like scalability and distribution can be dealt with by decomposition and mappings.

We presented a formal definition for a ontology module as an extension to the context ontology approach [16].

As future work, we should investigate a correctness criteria for ontologies formed by local and external components, as well as the problem of query answering in the presence of ontology modules.

4 Heterogeneities in Ontology Networks

In this chapter we provide an exhaustive classification of heterogeneity mismatches and conflicts. Here, as defined in the terminology section 1.1, mismatches correspond to differences between the ontologies at hand and are not necessarily conflicts. A conflict is a mismatch that requires a sophisticated solution or that does not have a solution at all. Mismatches and conflicts can occur when mapping or integrating two or more ontologies. It is necessary to have an exhaustive classification of heterogeneity mismatches in order to excogitate a possible solution.

First we will survey some literature that provides either a classification of ontology mismatches or deals in other ways with data integration. We also have a look at some new publications in this ares. At the end of each section we comment on the potential impact for our scheme. After that we present an integrated classification of conflicts that we elaborate mainly based on the work of Klein [68] and Wache [101]. We will also provide an example for each mismatch.

4.1 Survey of Literature

In this section we provide a survey of some literature dealing with heterogeneities within knowledge representation. Notably important are the classifications of mismatches or conflicts, respectively, presented by Klein and Wache. We will integrate these two classifications into one scheme in section 4.2.

4.1.1 Klein's classification of mismatches between ontologies

Klein [68] provides a classification of mismatches between ontologies. He elaborates his classification based on the work of Chalupsky [23], Visser et. al. [99] and Wiederhold [102].

He basically distinguishes between *language level mismatches* and *ontology level mismatches*.

- **Language (or meta) level mismatches**
  Language mismatches emerge when ontologies that are described in different ontology languages are combined.

  - **Syntax mismatch**
    Obviously, different ontology languages often use different syntaxes. For example in RDF Schema one uses <rdfs:Class ID="Person"> while in LOOM one uses (defconcept Chair) to express the same.
    A typical example of a 'syntax only' mismatch is on an ontology language that has several syntactical representations.

  - **Mismatch in the logical representation**
    A logical notion is represented in different ways. E.g. in one language it is possible to define disjointness of classes in a straightforward way (e.g., disjoint A B) while in another you need negation in subclass statements (e.g., A subclass-of (NOT B), B subclass-of (NOT A)). Therefore, translation

rules from one logical representation to another one are needed.

  - **Semantics of primitives**
    Sometimes the same name is used for a language construct in two languages, but the semantics may differ, e.g. there are several interpretations of A = B. Even when two ontologies seem to use the same syntax, the semantics can differ. For example, the OIL RDF Schema syntax [19] interprets multiple <rdfs:domain> statements as the intersection of the arguments, whereas RDF Schema itself uses union semantics.
  - Mismatch in the language expressivity
    The mismatch with the most impact is the difference in expressivity between two languages. For example, some language have the constructs to express negation, others have not.

- **Ontology (or model) level mismatches**
  Analogous to Visser [99], Klein distinguishes between *conceptualization mismatches* and *explication mismatches*. He describes a conceptualization mismatch as a difference in the way a domain is interpreted or conceptualized. This yields different ontological concepts or different relations between those concepts. In contrast, an explication mismatch is a difference in the way a conceptualization is specified. This yields mismatches in definitions, mismatches in terms and combinations of both.

  - **conceptualization mismatches**
    - **Scope**
      Two classes seem to represent the same concept, but do not have exactly the same instances, although they intersect.
    - **Model coverage and granularity**
      This is a mismatch in the part of the domain that is covered or the level of detail to which that domain is modelled.
  - **Explication mismatches**
    - Explicit choices of the modeller about **the style of modelling** yield the following two mismatches:
      - **Paradigm**
        Different paradigms can be used to represent concepts such as time, actions, plans, causality, propositional attitudes, E.g. temporal representations can be modelled via intervals or time points.
      - **Concept description**
        For the modelling of concepts there are several choices. E.g. distinctions between 2 classes can be modelled by using a qualifying attribute in the same class or by introducing a separate class. Another example is different class hierarchies.
    - **terminological mismatches**
      - **Synonym terms**
        The same concepts have different names. E.g. ”car” vs. ”automobile”. The solution seems relatively simple as it involves the use of

thesauri, but requires a lot of effort and several semantic problems can also occur. Especially, one must be very careful not to oversee a scope difference.

- **Homonym terms**
  Different concepts have the same name. Human interaction is needed to solve this conflict.

* **Encoding**
  This mismatch arises because values in different ontologies can be encoded in different formats, e.g. a date may be represented as 'dd/mm/yyyy' or as 'mm-dd-yy', instances are described in miles or kilometers, etc. This is a relatively easy to solve problem. It requires the usage of a converting function that is relatively easy to determine according to Klein.

Klein's distinctions between the two levels: language vs. ontology (meta level) is very important. We will keep this in mind. Furthermore we will reconsider the finer classification of explication.

4.1.2 Wache's classification of mismatches between database systems

In contrast to Klein, who is concerned with ontologies, Wache [101] provides a classification of mismatches that occur at integrating two or more heterogeneous database systems. Wache elaborated his classification of mismatches based on the work of Kim and Seo [66], Kashyap and Seth [61] and Goh [49]. As databases are knowledge representation systems and Wache looks at all possible database systems (i.e. relational ones, deductive ones and object oriented ones), this work provides some insights into ontology mismatches as well. However, we do not mention mismatches that are contained in Wache's classification but that do not occur between ontologies.

Wache calls the mismatches *integrity conflicts*. The mentioned conflicts are described in detail in the following subsections.

Structural heterogeneity mismatches

Structural heterogeneity mismatches concern structural mismatches between two different databases, i.e. database schema mismatches. Carried forward to ontologies these kinds of mismatches correspond to mismatches on the layer of conceptualization, i.e. concepts, attributes and relations.

Wache defines a *value* as the simplest structural element which can be a number or a string. Furthermore, he defines an *entity* as a complex structural element which corresponds to a real object and that records different kinds of information. According to Wache, an entity can have attributes that contain features of the entity; these attributes can refer to values or other entities.

According to Wache, it is not allowed to compare arbitrary structural elements with each other. A comparison is only admissible if the elements are semantically equivalent. I.e. first one has to identify semantically equivalent information.

Wache distinguishes between two different kinds of structural heterogeneity mismatches: *mismatches with bilateral correspondences*, i.e. two different structural

elements correspond to each other, and *mismatches with multilateral correspondences*, i.e. two different sets of structural elements correspond to each other.

- **mismatches with bilateral correspondences**
  - name mismatches
    The two individual structural elements differ in their names but are semantically equivalent, i.e. they are synonyms.
  - data type mismatches
    The data types of the two individual structural elements can differ. Such mismatches occur often together with a semantic heterogeneity conflict. E.g. the attribute value of one entity can be stored as an integer or as a double. Instead of using a number, one can also store the attribute value as a string.
  - integrity mismatches
    - different default values
    - contradicting integrity constraints (the integrity constraints in the different systems can be contradicting)
    - key mismatches (heterogeneous information systems access their data via different (primary) keys).

  Integrity mismatches usually come together with semantic heterogeneity mismatches. I.e. they have to be solved on the semantic layer, too. In general it can be assumed that there are no semantic correspondences between the structural elements if the integrity mismatches cannot be solved.

- mismatches with multilateral correspondences

  These kinds of mismatches occur when coherent information is split on several structural elements.

  - mismatches with multilateral attribute correspondences
    Here the relevant attributes only point to data and not to entities. Otherwise the corresponding mismatches are covered by the class *mismatches at multilateral entity correspondences*.
    As an example for such a conflict consider the distribution of the information of one attribute to several ones, e.g. an entity *Person* which can have an attribute *Name* or two attributes *First Name* and *Last Name*.
  - mismatches with multilateral entity correspondences.
    Analogous to *mismatches with multilateral attribute correspondences*, these mismatches occur when the same information is split to different sets of entities, i.e. we have a $n : m$- relationship, in general. E.g. the information about a person can be stored in one table or it can be split over several tables, e.g. one which stores the information about his work (social insurance number, his salary,) and another one which stores his medical record (vaccinations, illnesses, ...).

  - mismatches due to missing information
    Sometimes there are some entities missing which are required to make a match possible. E.g. assume that two stock exchange market databases should be mapped to each other. But they miss the information to which stock market they belong.

- discrepancies in the meta layer
  These mismatches occur when disparate structural elements correspond to each other. E.g. in one information system the information is encoded in an attribute and in another one it is encoded as an entity. There exist three kinds of meta layer discrepancies:

  - mismatches at data attribute correspondences (= the value of an attribute in one database corresponds to an attribute in another database)
  - mismatches at data entity correspondences (= the value of an attribute in one database corresponds to a entity (e.g. a relation) in another database)
  - mismatches at attribute entity mismatches

Semantic heterogeneity mismatches

Semantic heterogeneity mismatches do not refer to the structure, but to the interpretation of information. There are 2 kinds of semantic heterogeneity mismatches:

- semantic data heterogeneity mismatches (the interpretation of the data is different)

  - mismatches with scaling and units
    Two numerical values have the same semantics but differ in the value characteristic, e.g. the one system deals with values which are implicitly always thousands of and the other one deals with regular numbers. Or imagine a stock market database that stores the stocks in euro and another one that stores them in dollar.
  - representation mismatches
    Symbolic values can also differ in their peculiarity even when they have the same semantics. Here, it is assumed that a symbolic value can be transferred directly into another value and that there exists a bijective mapping between the different representations.
    As an example imagine a database which stores the values in the attribute date as "month/day/year" and another one stores it as "day.month.year".
  - surjective mapping mismatches
    In contrast to the *representation mismatches* and the *scaling and unit mismatches* here value sets of varying dimensions are mapped to each other. In general, the mapping function is not injective anymore but the surjectivity is preserved. I.e. all elements of the range are mapped to elements of the target domain. In the practice it can happen that not all elements of the range can be mapped because to one value of the range no corresponding value of the target domain can be+ identified. As an example imagine a broker database which assesses the value of a stock from the assessment of

another database and the market price aim of the stock (= the aim is not allowed to be higher than 50% more than the market price of the last three days).

- semantic domain heterogeneity mismatches (the semantic of entities and attributes is different)
  Semantic data heterogeneities occur when information systems have different conceptualizations. Domain heterogeneities can occur with attributes, entities or whole Information Systems. Without loss of generality here only the entities are covered.

  – subsumption conflict
    A subsumption conflict occurs when the set of instances of an entity is contained in the set of instances of another entity, e.g. a table stores only a subset of the tupels of another one.
  – overlapping conflict
    An overlapping conflict occurs when the instance sets of the entities overlap, i.e. in both sets there are elements that are not contained in the other set.
    As an example imagine three stock markets A, B and C and two databases which both store information about stocks sold at A and each stores additionally information about stocks sold at either A or B.
  – incompatibility
    At this kind of conflict the instance sets of the entities are completely different. There one has first to clarify if the entities (semantically) correspond at all. If an incompatibility can be confirmed then the cause is usually the different abstraction layers. E.g. NY and Frankfurt represent information about stocks. Therefore we can talk about semantical correspondence. But at the NY stock market and at the Frankfurter stock market they deal with completely different stocks. On this layer we can talk about incompatibility. The solution lies then in the decision which is layer the adequate one for the integration.

  – aggregation mismatches
    Aggregation mismatches occur when two Information Systems (IS) conceptualize a domain with different degrees of detail. An IS contains then a set of data sets that are comprised in one single data set in the other IS.
    As an example imagine one IS which stores information about ships while another one comprises the ships to convoys

Inconsistency and redundancy mismatches

In contrast to structural and semantic heterogeneity mismatches, here it is dealt with problems on the layer of data values. These problems arise from inconsistencies or redundancies of the data values and therefore depend of the state of the information system.

- Data inaccuracy mismatches
  Data inaccuracy mismatches occur when two data values that should represent

the same value, differ from each other.
Such mismatches can arise e.g. when the value gathering was error-prone.

- Temporal inconsistencies (different current data)
  Temporal inconsistencies occur when the data in one system is older that the data in another system.

- mismatches due to missing data
  Note that this conflict is not the same like the conflict caused by missing information. While the former deals with missing structural elements, the latter deals with missing data values. The problem here is that one can unforeseen encounter missing data values which is especially a problem if this value is needed for the further selection of data sets.

- redundancy problems
  This conflict occurs when different Information Systems contain the same information.

The more heterogeneous the Information Systems are, the more integration mismatches occur. Furthermore, it is very likely that a structural element is involved in more than one integration conflict. Then a **combination of integration mismatches occur**.
Usually certain structural heterogeneity mismatches come together with semantic mismatches. The combination of integration mismatches challenges the integration additionally.

Wache also makes a distinction between two categories at the first level. He separates structural heterogeneity vs. semantic heterogeneity which can be seen as a parallel to Klein, but then Wache does more refinement on the next levels.

4.1.3 Mismatches by Euzenat

Euzenat [42] proposes a definition of semantic interoperability based on model theory, i.e. based on the set of models belonging to the expressions. He defines *semantic interoperability* as the faculty to ascribe to each imported piece of knowledge the correct interpretation or set of models.

He provides a classification of possible interoperability requirements. Neglecting them can cause mismatches.

Levels of interoperability

- encoding
  ability to divide the representation into characters

- lexical
  ability to divide the representation into words (or symbols)

- syntactic
  ability to structure the representation in structured sentences (or formulas or assertions)

- semantic
  being able to construct the propositional meaning of the representation
- semiotic
  being able to construct the pragmatic meaning of the representation (or its meaning context)

These levels are layered on each other in the order of been mentioned. Note that each level cannot be achieved if the previous ones have not been completed. The first three levels can easily be achieved by the use of a formal language. Euzenat considers in his paper only semantic interoperability. *Semantic interoperability* is defined as beeing able to construct the propositional meaning of the representation. Within this requirement, he considers only *consequence preservation* which he defines as $\forall \delta, r \models_L \delta \Rightarrow \tau(r) \models_{L'} \tau(\delta)$. Here L and L' are two languages and r is a representation in L that has to be transferred into L' by means of the transformation τ.

The Euzenat paper was published in the same year as Klein's but his focus is on semantic interoperability for which he develops a logic based on model theory. Euzenat considers only transfers between different languages. More precisely, he considers only mismatches on the meta or language layer as classified by Klein. So we assume not to taking him into account further on.

4.1.4 Mismatches by Benerecetti et.al.

In their paper *Contextual Reasoning Distilled* Benerecetti et.al. [10] provide a theory of contextual reasoning from the perspective of knowledge representation. They present three general forms and its classification. A any representation in context dependent. So they introduces the metaphor of the box: Inside you have the linguistic expression, outside there are a collection of parameters $P_1 ... P_n$ and a value V_i for each parameter P_i. Forms of contextual reasoning can be splitted in three parts:

Forms of contextual reasoning

- Localized reasoning
  Information and its representation depend on a collection of contextual parameters and their values. Localized reasoning happens when a particular collection of parameters are fixed.
- Push and pop
  Push is the operation of adding contextual parameters to the collection whereas pop removes them. With this mechanism one can vary the degree of approximation.
- Shifting
  This means changing the value of contextual parameters. So a 'translation' of a representation into another is done.

This three forms correspond to operating on three fundamental dimensions. These are:

Dimensions of context dependence

- Partiality
  The portion of the world is taken into account here. A representation is partial when it describes only a subset of a more comprehensive state of affair.
- Approximation
  The level of detail is addressed. A representation is approximate when it abstracts away some aspects of a given state of affair.
- Perspective
  This is the point of view. A representation is perspectival when it encodes a spatio-temporal, logical and cognitive point of view on a state of affair.

Based on this they distil the principles of a logic afterwards.

Benerecetti et.al. based their work on natural languages constructs. This is very general. With ontologies we are more specific and therefore refer to Klein and Wache.

4.1.5 Mismatches by Ghidini and Giunchigla

In their paper Ghidini and Giunchiglia [21] present a model-theoretic formalization of abstraction. They define an abstraction as a mapping between two representations of a problem, which are a pair of formal languages. Then they restrict themselves to abstractions which are total, effective and surjective. After defining atomic abstraction they give a classification of abstraction which are:
- symbol abstraction operate on symbols and collapse them together
- arity abstraction operates an arities and lowering them
- truth abstraction operates on predicates and mapping them into the symbol for truth

Their key idea for a semantic of abstraction is to use domain relations and compatibility relations to model, at a semantic level, the syntactic abstraction relation between terms and formulae of the ground and abstract language.

Logic and abstraction can be done on a high theoretical level. Ontologies can be seen as one specific representation of the general problem. The mismatches are described in a more domain specific way here. This level is too general for our approach.

4.2 A Classification of Mismatches

In this section we present a classification of mismatches which bases on Klein's classification which we agree upon in principle. But in order to round it off we extend it by making some finer grained distinctions based on Wache's classification. and is extended with elements of Wache's classification. Thus, analogous to Klein we distinguish between *language (or meta-model) mismatches* and *ontology (or model) mismatches*.

Figure 4.1 shows the integrated classification of mismatches.

- **language (or meta-model) mismatches**
  We adopted the language mismatches from Klein, i.e. *syntax mismatches*, *mismatches in the logical representation*, *semantics of primitives* and *language expressivity*. A detailed description can be found in section 4.1.1

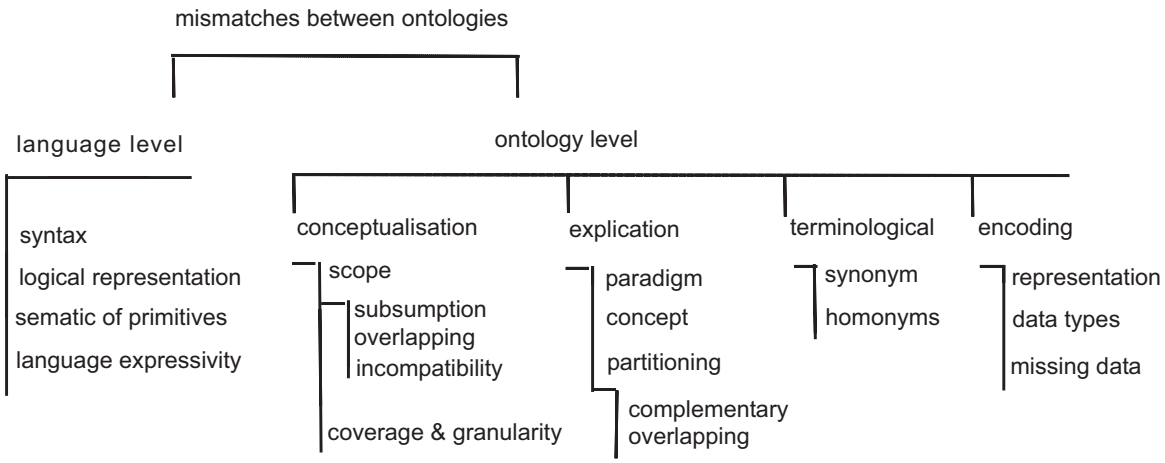

Figure 4.1: An overview of the integrated mismatches

In the mapping process the first step consists of importing the ontologies into a common format (cf. [32]). Here, the ontologies are transferred either into one of the OWL species or into one of the WSML species. Clearly, the translation into a common format eliminates the differences in *syntax*.

Differences in the *logical representation* occur when syntactically different, but logically equivalent statements are used to represent the same notion. E.g. consider the way disjointness is expressed in the OWL Lite species of the Web Ontology Language OWL [36] , compared to the way disjointness is usually expressed in the OWL DL species[1].

This is not really an issue with the language itself, but rather an issue with the use of the language. However, when a language allows the user to model the same thing in different ways, it is easy for a user to mistakenly model certain things in an inconvenient way and it is harder for a user to understand the model created by a different user or indeed created by him/herself in the past. When the ontology language used by the technique/tool/system allows such different logical representations of equivalent statements, this mismatch still needs to be taken into account in the ontology mapping process. In order to overcome these issues, one could think of a normalization step before the start of the mapping process or reasoning during the mapping process in order to detect equivalence in logical expressions.

When the *semantics of primitives* is different in different ontology languages, i.e. a syntactically equivalent construct has a different meaning in the different languages, the translation to the common representation needs to take this into account. Fortunately, this problem can be resolved in the translation to the common representation. If both ontologies already use the common representation and this common representation does not allow ambiguous statements, this mismatch does not occur.

Differences in *expressivity of the languages* are resolved in the translation to the common representation language. However, if the expressivity of the common

[1]The OWL Lite statement `Class(owl:Nothing complete` A B`)`, although also valid in OWL DL, is usually modeled as `DisjointClasses(`A B`)` in OWL DL

representation language is not a superset of the language of the source ontology, some semantics might get lost in the translation, as was pointed out in [79].

- **ontology (or model) mismatches**

  These are mismatches or differences which lie in the modelling of the information by means of ontologies. Analogous to Klein, we distinguish between *conceptualization mismatches*, *explication mismatches* and *encoding mismatches*.

  - **conceptualization mismatches**

    * Scope
      According to Klein, these kind of mismatches occur when two classes seem to represent the same concept, but do not have exactly the same instances although they intersect. Wache's distinction is finer grained as he also distinguishes between subsumption, overlapping and incompatibility of the domain of concepts.

      · subsumption
      The instance set of one class is subsumed by the extension of the other class. See below an example of this mismatch in OWL.

      Ontology 1:

```
Class(Pc complete unionOf(Desktop, Laptop))
Individual(ibmR40 type (Laptop))
Individual(compaqPresario200 type (Desktop))
Class(PcOffice complete intersectionOf(Pc, complementOf(Desktop)))
```

      Ontology 2:

```
Class(Pc complete unionOf(Desktop, Laptop, Server))
Individual(ibmR40 type (Laptop))
Individual(compaqPresario200 type (Desktop))
Individual(dellPoweredge001 type (Server)).
Class(PcOffice complete intersectionOf(Pc, complementOf(Desktop)))
```

      Obviously, the instance set belonging to the class PcOffice from the first ontology is subsumed by the instance set belonging to the class PcOffice from the second ontology.

      · overlapping
      The instance sets of two classes overlap. See below an example of this mismatch in OWL.

      Ontology 1:

```
Class(Pc complete unionOf(Desktop, Laptop, Server))
Individual(ibmR40 type (Laptop))
Individual(compaqPresario200 type (Desktop))
Individual(dellPoweredge001 type (Server)).
Class(PcOffice complete intersectionOf(Pc, complementOf(Desktop)))
```

      Ontology 2:

```
Class(Pc complete unionOf(Desktop, Laptop, Server))
Individual(ibmR40 type (Laptop))
Individual(compaqPresario200 type (Desktop))
Individual(dellPoweredge001 type (Server)).
Class(PcOffice complete intersectionOf(Pc, complementOf(Server)))
```

Obviously, the instance set of the class PcOffice from the first ontology and the instance set of the same class from the second ontology intersect in the instances of type Laptop. Furthermore, each instance set contains instances either of type Server or of type Desktop.

- incompatibility
  The instance sets are completely disjoint. This does not mean necessarily that the classes do not correspond at all to each other. Maybe we have to switch to a higher abstraction level. See below an example of this mismatch in OWL: Imagine two ontologies o1 and o2 as described below:

  Ontology 1:

```
Class(Pc complete unionOf(Desktop, Laptop))
Individual(ibmR40 type (Laptop))
Individual(compaqPresario200 type (Desktop))
Class(PcOffice complete intersectionOf(Pc, complementOf(Laptop)))
```

  Ontology 2:

```
Class(Pc complete unionOf(Laptop, Server))
Individual(ibmR40 type(Laptop))
Individual(dellPoweredge001 type (Server)).
Class(PcOffice complete intersectionOf(Pc, complementOf(Laptop)))
```

  Obviously, the instance set of the class PcOffice from the first ontology and the instance set of the same class in the second ontology do not intersect at all.

* model coverage and granularity
  Klein describes this kind of mismatch as a mismatch in the part of the domain that is covered or the level of detail to which that domain is modelled. Wache considers only granularity mismatches and calls this kind of mismatches *aggregation mismatch*.
  See below an example of this kind of mismatches in OWL: Imagine two ontologies o1 and o2 as described below. There it can be seen that o1 considers only Pcs as subclass of computers while o2 makes finer grained difference between laptops and desktop pcs.

  Ontology 1:

```
Class(Pc rdfs:subClassOf(Computer))
Individual(ibmR40 type(Pc))
Individual(compaqPresario200 type(Pc))
```

  Ontology 2:

```
Class(Pc rdfs:subClassOf(Computer))
Class(Laptop rdfs:subClassOf(Pc))
Class(Desktop rdfs:subClassOf(Pc))
Individual(ibmR40 type(Laptop))
Individual(compaqPresario200 type(Desktop))
```

We distinguish a special case of these kinds of mismatches which are *mismatches due to missing information* as mentioned by Wache. An example of this kind of mismatch are e.g. two ontologies that model the stock information of a stock market in New York and the stock information of a stock market in Frankfurt, respectively. Here, imagine that there is no class or attribute which specifies the location of the respective stock markets.

– **explication mismatches**
These kinds of mismatches are caused by explicit choices of the modeller about the style of modelling.
Wache considers a kind of mismatch that he calls *discrepancies at the meta-layer*.
An example of this kind of mismatches would be for example, if the food duration in ontology is expressed using the best before date while in another ontology the number of days after the production is used.

Ontology 1:

```
Class(Food rdfs:subClassOf(Product))
rdfs:Property(bestBefore Food xsd:date)
```

Ontology 2:

```
Class(Food rdfs:subClassOf(Product))
rdfs:Property(DaysDuration Food xsd:integer)
```

∗ Paradigm
Different paradigms can be used to represent concepts such as time, actions, plans, causality, propositional attitudes, E.g. temporal representations can be modelled via intervals or time points.

∗ Concept description
For the modelling of concepts there are several choices. E.g. distinctions between 2 classes can be modelled by using a qualifying attribute in the same class or by introducing a separate class. Another example is different class hierarchies. E.g., imagine two ontologies o1 and o2 as described below (cf. [68]):

Ontology 1:

```
Class(Book rdfs:subclassOf(scientific publication))
Class(Dissertation rdfs:subclassOf(Book))
```

Ontology 2:

```
Class(Book rdfs:subclassOf(publication))
Class(ScientificBook rdfs:subclassOf(Book))
Class(Dissertation rdfs:subclassOf(scientificBook))
```

* ontology elements partitioning
  This kind of mismatch has not been considered by Klein. Wache considers *mismatches with multilateral correspondences* which inspired us to this kind of mismatch. Mismatches in *ontology elements partitioning* are differences in the sets of modelling elements for one piece of information. E.g. in one ontology the concept family models a whole family while in another one there exist several concepts that form a family like father, mother, child, grandfather and the like. These concepts have to be modelled in the family concept of the first ontology at least as attributes to avoid a aggregation mismatch. Another example would be the attribute fullname in an OWL ontology which is modelled in another OWL ontology as firstname and familyname.

  Ontology 1:
  ```
  Class(Person rdfs:subClassOf(Human))
  rdfs:Property(FullName Person xsd:string)
  ```

  Ontology 2:
  ```
  Class(Person rdfs:subClassOf(Human))
  rdfs:Property(FirstName Person xsd:string)
  rdfs:Property(LastName Person xsd:string)
  ```

– **terminological mismatches**
  Wache summarizes these kinds of mismatches under the name *name mismatches*.

  * synonym terms The same concepts have different names. E.g. "car" vs. "automobile". The solution seems relatively simple as it involves the use of thesauri, but requires a lot of effort and several semantic problems can also occur. Especially, one must be very careful not to oversee a scope difference. As an example, imagine two ontologies o1 and o2 as described below. The same concept is called Car in o2 and Automobile in o1.

    Ontology 1:
    ```
    Class(Automobile SubClassOf(MotorVehicle))
    ```

    Ontology 2:
    ```
    Class(Car SubClassOf(MotorVehicle))
    ```

  * homonym terms Different concepts have the same name. Human interaction is needed to solve this conflict. As an example, imagine two ontologies o1 and o2 as described below. Although both ontologies deal about *Heads*, the obviously do not mean the same concepts.

    Ontology 1:

```
Class(Head subclassOf(Body))
```

Ontology 2:

```
class(Head subClassOf(Employee))
```

– **Encoding**
Klein describes this mismatch as a mismatch that arises because values in different ontologies can be encoded in different formats. This is a relatively easy to solve problem. It requires the usage of a converting function that is relatively easy to determine according to Klein.
Wache calls this kind of mismatches *representation mismatches*.
He also introduces *data type mismatches* which we see as a special case of the encoding mismatches as well as *scaling and units* and *data inaccuracy* and *mismatches due to missing data*.

∗ representation encoding
As an example imagine two ontologies o1 and o2 as described below. They have the same ontology statements but while in o1 the distance is stored in kilometers in o2 it stored in miles.

Ontology 1 and Ontology 2:

```
Class(Trip rdfs:subClassOf(travel))
rdfs:Property(distance Trip xsd:Integer)
```

∗ data types
The datatypes of different corresponding attributes can differ. This kind of mismatch can be solved by specifying an adequate conversion. But it has to be taken care as such mismatches can also occur together with a scope mismatch.
A special case of these kinds of mismatches are mismatches with scaling and units, which is also mentioned by Wache. But in his classification, he distinguished it from the representation mismatches. Also *scaling and units* is a special case of this mismatch.
As an example imagine two ontologies as described below. There, the age of a person in o1 is stored as an integer while in o2 it is stored as a double.

Ontology 1:

```
Class(Person rdfs:subClassOf(Human))
rdfs:Property(Age Person xsd:Integer)
```

Ontology 2:

```
Class(Person rdfs:subClassOf(Human))
rdfs:Property(Age Person xsd:Double)
```

∗ mismatches due to missing data
As an example of this kind of mismatches imagine an instance in one ontology which hasn't fully specified its attributes. Therefore it is not clear if it can be mapped to an instance in another ontology which has the same values in the specified attributes.

Due to time constraints, we were not able to consider the integrity mismatches mentioned by Wache. Although, in WSML there are integrity constraints and therefore integrity mismatches can occur.

5 Representing Ontology mappings

This chapter presents how the ontology mappings can be described using the mapping language developed as part of a joint effort between DIP and SEKT [27, 28] projects. We will start by presenting a general overview of the mapping language as well as the requirements it has to meet, and we will continue by describing the syntax and the semantics of this language. In the end we will try to illustrate the usage of this language by representing the required mappings for some of examples.

5.1 Mapping Language

5.1.1 General Considerations

The mapping language described here was meant to be independent of any formalism that may be used for modelling certain ontologies. Even if it was designed having in mind WSML-Flight [30] and OWL [36] it doesn't commit to any of the peculiarities of these two languages. According to the particular usage of the mapping language, one can devise an appropriate formal semantics to it. Here, as we deal mainly with WSML in the DIP project, we provide in section 5.2 a grounding of the mapping language to WSML. As mentioned in section 1.1 the ontologies to be mapped are considered to be quadruples $<C, R, I, A>$ where C stands for classes, R for relations, I for instances and A for axioms, but no assumptions are made related to the types of class expressions, attribute expressions and relation expressions used in the language. The type of mapping conditions, represented by the preconditions to be fulfilled in order to enable the mappings, is also left open.

As a consequence, an associated syntax for our mapping language is defined in section (5.1.3).

The mapping language was developed in close relation with a set of elementary mapping patterns, identified in SEKT D4.3.1 [28]. In order to keep the language as compact as possible, not all the mapping patterns presented in D4.3.1 have a direct correspondent construct in the language, but only the most general ones. The general constructs might be combined with several additional constructs, in order to represent the more specific patterns, too.

5.1.2 Requirements

In the following, we present some of the requirements a mapping language should fulfil, as identified in [33], and how these are covered by presented mapping language.

- Mapping on the Semantic Web

  A mapping language should be able to map between ontologies on the Web and the mapping itself made available on the Web. As mentioned above, the mapping language we propose was designed having in mind OWL (one of the current standards for specifying the ontologies on the Web) and WSML-Flight (which is part of the family languages that formalizes the Web Service Modelling Ontology [91] specification) to be used within the DIP project. This shall guarantee the fulfillment of this requirement for a wide range of applications in the areas of the Semantic Web and Semantic Web Services.

- Mapping between Description Logic Ontologies

  The mapping language to be used does not provide any formal grounding, leaving open the choice of the appropriate semantics for the mappings; as a consequence, a formal semantics compatible with Description Logics can be associated with the mappings. Just as well, a formal semantics corresponding to Logic Programming can be associated with the mappings.

- Specify instance transformations

  The instance transformation will be performed based on the provided *built-in functions*. They will allow both *structural* transformations (changing the structure of the given instance) and *value* transformations (only the value of the instance is changed) for particular instances. These built-ins could be made available by the underlying mapping language implementation itself, or through URIs pointing to existing realizations of these functions as web services.

- One mapping for all tasks

  A proper mapping language should provide means for supporting various tasks such as ontology merging, query rewriting, instance unifications and transformations. The mapping language adopted, offers constructs for class mappings, relations mappings and individual mappings which, together with other additional constructs, are able to accommodate the needs of various mediation related tasks.

- Use of mapping patterns

  As we previously described, the mapping language offers explicit constructs for the most general mapping patterns identified in [28] and a set of additional constructs for covering the most specific mapping patterns, too.

- Versioning support

  This requirement addresses the problem of mappings in the context of evolving ontologies, implying support for versioning from the mapping language side too. The mapping language we chose, offers the possibility of introducing the version identifier in a special construct which corresponds to the non-functional properties in WSMO [91]. However, versioning goes beyond the scope of this deliverable.

- Treating classes as instances

  This requirement is due to the fact that different modelling decisions may be taken when developing ontologies for related or even similar domains. The most obvious example is the one in which we have a certain entity modelled as a concept in one ontology and as an instance in the other ontology. As a consequence a mapping language should be able to accommodate all these mismatches that may appear due to different modelling strategies. To this end we have to offer support for mappings between any types of entities from the two ontologies, e.g. classes, relations, or individuals. In this direction, [28] presents a hierarchical organizations of the mapping patterns that contains a set of patterns which covers this requirement.

- Mappings of different cardinalities

  In order to meet this requirement, a set of patterns that can be expressed using the constructs of our mapping language are introduced by [28].

We want to conclude this section with one additional requirement:

- A rule language for mappings

  A rule language is required for expressing mappings between ontologies (a motivation is offered in [27]). The mapping language we chose is a based on a Horn subset of F-Logic [65] and adds support for the special symbols *true* and *false*, integrity constraints, default negation (*not*), the inequality predicate $\neq$ (either as a built-in or by axiomatizing it in the language), the use of function symbols, and a set of built-in functions which need to be supported by the respective implementation.

5.1.3 Abstract Syntax

Appendix C of [28] provides an abstract syntax written in EBNF, for the proposed mapping language. For the purpose of the self containment of this document and for a better understanding of the examples presented in the next section, in the following, we briefly review this syntax. Elements presented between square brackets '[' and ']' are optional, while the elements between curly brackets '{' and '}' can have zero or multiple occurrences.

The URIs that identify all the elements presented in an ontology (classes, attributes, instances, relations) are denoted in the abstract syntax with the name **URIReference**. As a consequence, we have the following identifiers:

mappingID ::= **URIReference**
ontologyID ::= **URIReference**
classID ::= **URIReference**
propertyID ::= **URIReference**
attributeID ::= **URIReference**
relationID ::= **URIReference**
individualID ::= **URIReference**

The abstract syntax for concrete data values is taken from the OWL abstract syntax:

dataLiteral ::= **typedLiteral** |**plainliteral**
typedLiteral ::= **lexicalForm**ˆˆ**URIReference**
plainLiteral ::= **lexicalForm**['@'**languageTag**]

A mapping is expressed in the following way:

mapping ::= 'Mapping(' [**mappingID**]
{ 'source(' { **ontologyID** } ')' }
'target(' **ontologyID** ')'
{ **directive** } ')'

The **directive** is defined as follows:

directive ::= **annotation**
|**expression**

annotation ::= 'Annotation(' **propertyID propertyValue** ')'

The **annotation** corresponds to the WSMO [91] non-functional properties and **expression** corresponds to class mappings, attributes mappings, relation mappings or instance mappings. The attributes can be considered as binary relations with a certain range and a certain that denotes the classes they are associated with. The **expression** for a class mapping is defined as follows:

expression ::= 'classMapping(' ['one-way'|'two-way']
classExpr classExpr
{**attributeMapping**} { **classCondition** }
['{' **logicalExpression**'}'] ')'

attributeMapping ::= 'attributeMapping(' ['one-way'|'two-way']
attributeExpr attributeExpr
{ **attributeCondition** } ')'

expression ::= 'attributeMapping('['one-way'|'two-way']
attributeExpr attributeExpr
{**attributeCondition** } ['{' **logicalExpression** '}'] ')'

As illustrated above, an attribute mapping can be either nested inside of the class mapping or it can appear outside of the class mapping. A **classCondition** represents an axiom which define the general conditions of the mappings, and plays the role of the preconditions that have to be fulfilled in order to perform the given mapping. Also, arbitrary axioms are allowed in the form of **logical expression**. The axioms are very much dependent of the logical language being used for the mappings and they are not going to be further specified in here. Note that for nested attribute mappings no logical expressions are required since the logical expressions defined for the class mapping cover the nested attribute mappings as well.

Accordingly, the expressions for the other types of mappings are defined in a similar manner as the class mapping:

expression ::= 'relationMapping(' ['one-way'|'two-way']
relationExpr relationExpr
{**relationCondition** } ['{' **logicalExpression** '}'] ')'

expression ::= 'instanceMapping(' **individualID individualID**')'

expression ::= 'classAttributeMapping('['one-way'|'two-way']
(**classExpr attributeExpr**)|(**attributeExpr classExpr**)
{**classAttributeMapping**} {**attributeMapping**}
{**classCondition**} {**attributeCondition**}
['{' **logicalExpression** '}'] ')'

expression ::= 'classRelationMapping(' ['one-way'|'two-way']
(**classExpr relationExpr**)|(**relationExpr classExpr**)
{**classCondition** } {**relationCondition**}
['{' **logicalExpression** '}'] ')'

Each of the mappings can be either biderectional ('two-way') or unidirectional ('one-way'); the default is consider to be 'two-way'.

The **classAttributeMapping** inherits some of the characteristics of **classMapping** and **attributeMapping**: it can nest **attributeMapping**s and it can be nested in another **classAttributeMapping**. As a consequence we also have to define:

classAttributeMapping ::= 'classAttributeMapping(' ['one-way'|'two-way']
(**classExpr attributeExpr**)|(**attributeExpr classExpr**)
{**classAttributeMapping**} {**attributeMapping**}
{ **attributeCondition** } { **classCondition** }')'

The **classExpr** allows basic boolean algebra, and it has the following definition:

classExpr ::= **classID**
|'and(' **classExpr classExpr** {**classExpr** } ')'
|'or(' **classExpr classExpr** {**classExpr** } ')'
|'not(' **classExpr** ')'
|'join(' **classExpr classExpr** { **classExpr** }
['{' **logicalExpression** '}'] ')'

and stands for intersection, *or* for union, *not* for different (*C-D* corresponds to *and(C, not(D))*) and *join* for join expression.

The **attributeExpr** is defined as such allowing for inverse, transitive closure, symmetric closure, and reflexive closure:

attributeExpr ::= **attributeID**
|'and(' **attributeExpr attributeExpr** { **attributeExpr** } ')'
|'or(' **attributeExpr attributeExpr** { **attributeExpr** } ')'
|'not(' **attributeExpr** ')'
|'inverse(' **attributeExpr** ')'
|'symmetric(' **attributeExpr** ')'
|'reflexive(' **attributeExpr** ')'
|'trans(' **attributeExpr** ')'
|'join(' **attributeExpr attributeExpr** { **attributeExpr** }
['{' **logicalExpression** '}'] ')'

Relation expressions are defined in a similar manner with the class expressions:

relationExpr ::= **relationID**
|'and(' **relationExpr relationExpr** { **relationExpr** } ')'
|'or(' **relationExpr relationExpr** { **relationExpr** } ')'
|'not(' **relationExpr** ')'
|'join(' **relationExpr relationExpr** { **relationExpr** }
['{' **logicalExpression** '}']

The rest of the definition that haven't been introduced yet are presented below:

classCondition ::= 'attributeValueCondition(' **attributeID**
(**individualID** | **dataLiteral** |**classExpr**)')'

classCondition ::= 'attributeOccurrenceCondition(' **attributeID** ')'

attributeCondition ::= 'valueCondition(' (**individualID** | **dataLiteral**
| **classExpr**) ')'

attributeCondition ::= 'expressionCondition(' **attributeExpr** ')'

relationCondition ::= 'parameterCondition(' (**individualID** | **dataLiteral**
| **classExpr**) ')'

relationCondition ::= 'expressionCondition(' **relationExpr** ')'

5.2 Grounding to WSML

In order to allow for reasoning with the mapping language we formally ground the semantics of the mapping language to Logic Programming and, more specifically, to WSML-Flight [31] (extended with function symbols), which is an ontology language developed in the context of the DIP project.

We translate the mapping language to a rule language and thus there are several restrictions on the type of rules which can be created. In general, a rule may only have one literal in the head. Rules with a conjunction can easily be rewritten into a number of rules with only a single literal in the head. However, a disjunction in the head cannot be handled. Furthermore, WSML-Flight does not allow for classical negation, only default negation. Default negation may only occur in the body of the rule; not in the head.

All the statements of the mapping language are translated to WSML by the function $t()$ which takes as argument a mapping language statement and returns the corresponding WSML construct. The symbol '$\mapsto$' makes the links between the function call and result returned by this function. Note that the constructs containing '$\mapsto$' are not rules, i.e. '$\mapsto$' does not separate a rule head and a rule body, as it might be suggested by the form of these constructs.

In the mapping, $?x$ and $?y$ are variables. $?x_{new}$ stands for a newly introduced variable. X and Y are meta-variables which are replaced with real variables during the translation. In the class, attribute, and relation expressions 'naf' stands for *negation as failure*.

The **URIReference**s used to identify the **mappingID**, **ontologyID**, **classID**, **propertyID**, **attributeID**, **relationID**, and **individulID** are represented by full URIs in WSML. As a consequence, we have:

$t(\textbf{URIReference}) \mapsto <$ "URIReference" $>$

The **typedLiteral**s and **plainLiteral**s (**dataLiteral**s) have direct correspondents in WSML language as plain literals and typed literals:

$t(\textbf{typedLiteral}) \mapsto$ typedliteral

$t(\textbf{plainLiteral}) \mapsto$ plainliteral

The **logicalExpression**s have also a direct correspondent in WSML logical expressions, namely:

$t(\textbf{logicalExpression}) \mapsto \textbf{expr}$

It is possible to use a meta-variable in logical expressions which are nested inside other mapping expression (for example, class mappings). The meta-variable is '?X' and is syntactically substituted in the translation of the mapping language to WSML-Flight:

$t(\textbf{logicalExpression}, X) \mapsto \textbf{expr}[?\text{X} := X]$

The annotations in the mapping language correspond to the non-functional properties in WSML; several annotations can be translated in one non-functional properties block:

$t(Annotation_1(P_1\ v_1)...Annotation_n(P_n\ v_n)) \mapsto$
nonFunctionalProperties
P_1 **hasValue** v_1
...
P_n **hasValue** v_n
endNonFunctionalProperties

5.2.1 Class mappings

Below, the translations of class mappings are specified. First, a two-way class mapping is translated into two one-way class mappings. Then, a one-way mapping is translated as a rule of subtranslations.

$t(classMapping(two\text{-}way\ \text{classExpr}_1\ \text{classExpr}_2$
$\text{attributeMapping}_1\ ...\ \text{attributeMapping}_n$
$\text{classCondition}_1\ ...\ \text{classCondition}_m$
$\text{logicalExpression}_1\ ...\ \text{logicalExpression}_q)) \mapsto$
$t(classMapping(one\text{-}way\ \text{classExpr}_1\ \text{classExpr}_2$
$\text{attributeMapping}_1\ ...\ \text{attributeMapping}_n$
$\text{classCondition}_1\ ...\ \text{classCondition}_m$
$\text{logicalExpression}_1\ ...\ \text{logicalExpression}_q))$

$t($*classMapping*$($*one-way* classExpr$_2$ classExpr$_1$
attributeMapping$_1$ ... attributeMapping$_n$
classCondition$_1$ ... classCondition$_m$
logicalExpression$_1$ ... logicalExpression$_q$))

$t($*classMapping*$($*one-way* classExpr$_1$ classExpr$_2$
attributeMapping$_1$ ... attributeMapping$_n$
classCondition$_1$ ... classCondition$_m$
logicalExpresion$_1$ ... logicalExpresion$_q$)) $\mapsto$
$t($classExpr$_2, ?x)$ **impliedBy** $t($classExpr$_1, ?x)$ 'and'
$t($attributeMapping$_1, ?x)$ 'and' ... 'and' $t($attributeMapping$_n, ?x)$
$t($classCondition$_1$, $?x)$ 'and' ... 'and' $t($classCondition$_m$, $?x)$
$t($logicalExpression$_1, ?x)$ ... $t($logicalExpression$_q, ?x)$ '.'

In the mapping language, for different class expressions different translations are required. There are no explicit constructs for representing the intersection, union, difference and join operations in WSML. Therefore, we have to create a new concept and to write for it the WSML logical expression that defines the intersection, union, complement, and join, respectively. Note that the or() construct may only be used in the source of a mapping rule and may not be used in a two-way mapping rule.

$t($and$($**classExpr**$_1$ ... **classExpr**$_n), X) \mapsto$
$t($**classExpr**$_1, X)$ and ... and $t($**classExpr**$_n, X)$

$t($or$($**classExpr**$_1$ ... **classExpr**$_n), X) \mapsto$
$t($**classExpr**$_1, X)$ or ... or $t($**classExpr**$_n, X)$

$t($not$($**classExpr**$), X) \mapsto$ 'naf' $t($**classExpr**$, X)$

$t($join$($**classExpr**$_1$ ... **classExpr**$_n$ **logicalExpression**$_1$ ...
logicalExpression$_n$)) $\mapsto$
$t($**classExpr**$_1, f(?x_2, ..., ?x_n))$ 'impliedBy' $t($**classExpr**$_2, ?x_2)$ 'and' ... 'and'
$t($**classExpr**$_n, ?x_n)$ 'and' $t($**logicalExpression**$_1, \{f(?x_2, ..., ?x_n), ?x_2, ..., ?x_n\})$
'and' ... 'and' $t($**logicalExpression**$_1, \{f(?x_2, ..., ?x_n), ?x_2, ..., ?x_n\})$ '.'

One more transformation function is required, for the case when the **classExpr** is a **classID**:

$t($**classID**$, X) \mapsto X$ 'memberOf' $t($**classID**$)$

5.2.2 Attribute mappings

Below the translations of attribute mappings are specified. For the two-way attribute mapping we distinguish three cases: (1) no variables are given as parameters, (2) one variable is given and (3) two variables are given.

$t($*attributeMapping*$($*two-way* attributeExpr$_1$ attributeExpr$_2$
attributeCondition$_1$ ... attributeCondition$_n$
logicalExpresion$_1$ ... logicalExpresion$_m$)) $\mapsto$
$t($*attributeMapping*$($*one-way* attributeExpr$_1$ attributeExpr$_2$

attributeCondition$_1$ ... attributeCondition$_n$
logicalExpresion$_1$ ... logicalExpresion$_m$))
t(*attributeMapping*(*one-way* attributeExpr$_2$ attributeExpr$_{12}$
attributeCondition$_1$ ... attributeCondition$_n$
logicalExpresion$_1$ ... logicalExpresion$_m$))

t(*attributeMapping*(*two-way* attributeExpr$_1$ attributeExpr$_2$
attributeCondition$_1$ ... attributeCondition$_n$
logicalExpresion$_1$ ... logicalExpresion$_m$), X) $\mapsto$
t(*attributeMapping*(*one-way* attributeExpr$_1$ attributeExpr$_2$
attributeCondition$_1$ ... attributeCondition$_n$
logicalExpresion$_1$ ... logicalExpresion$_m$), X)
t(*attributeMapping*(*one-way* attributeExpr$_2$ attributeExpr$_{12}$
attributeCondition$_1$ ... attributeCondition$_n$
logicalExpresion$_1$ ... logicalExpresion$_m$), X)

t(*attributeMapping*(*two-way* attributeExpr$_1$ attributeExpr$_2$
attributeCondition$_1$ ... attributeCondition$_n$
logicalExpresion$_1$ ... logicalExpresion$_m$), X, Y) $\mapsto$
t(*attributeMapping*(*one-way* attributeExpr$_1$ attributeExpr$_2$
attributeCondition$_1$ ... attributeCondition$_n$
logicalExpresion$_1$ ... logicalExpresion$_m$), X, Y)
t(*attributeMapping*(*one-way* attributeExpr$_2$ attributeExpr$_{12}$
attributeCondition$_1$ ... attributeCondition$_n$
logicalExpresion$_1$ ... logicalExpresion$_m$), X, Y)

t(*attributeMapping*(*one-way* attributeExpr$_1$ attributeExpr$_2$
attributeCondition$_1$ ... attributeCondition$_n$
logicalExpresion$_1$ ... logicalExpresion$_m$)) $\mapsto$
t(*attributeMapping*(*one-way* attributeExpr$_1$ attributeExpr$_2$
attributeCondition$_1$ ... attributeCondition$_n$
logicalExpresion$_1$ ... logicalExpresion$_m$), x_{new})

t(*attributeMapping*(*one-way* attributeExpr$_1$ attributeExpr$_2$
attributeCondition$_1$ ... attributeCondition$_n$
logicalExpresion$_1$ ... logicalExpresion$_m$), X) $\mapsto$
t(*attributeMapping*(*one-way* attributeExpr$_1$ attributeExpr$_2$
attributeCondition$_1$ ... attributeCondition$_n$
logicalExpresion$_1$ ... logicalExpresion$_m$), X, x_{new})

t(*attributeMapping*(*one-way* attributeExpr$_1$ attributeExpr$_2$
attributeCondition$_1$ ... attributeCondition$_n$
logicalExpression$_1$ ... logicalExpression$_m$), X, Y) $\mapsto$
t(attributeExpr$_1$, X, Y) 'impliedBy' t(attributeExpr$_1$, X, Y)
'and' t(attributeCondition$_1$, X, t(attributeID$_1$) 'and' ... 'and'
t(attributeCondition$_n$, X, t(attributeID$_1$))
'and' t(logicalExpression$_1$, X, Y) 'and' ... 'and'
t(logicalExpression$_m$, X, Y)

The mappings for **attributeExpr**s implies the usage of a transformation function having three parameters: the attribute expression to be transformed, the **ID** of an instance of the concept owning this attribute, and the value of the attribute.

$t(\textbf{attributeID}, X, Y) \mapsto X[\ t(\text{attributeID})$ 'hasValue' $Y]$

$t(\text{inverse}(\textbf{attributeExpr}), X, Y) \mapsto$
$t(\textbf{attributeExpr}, Y, X)$

$t(\text{symmetric}(\textbf{attributeExpr}), X, Y) \mapsto$
$t(\textbf{attributeExpr}, X, Y)$ 'and' $t(\textbf{attributeExpr}, Y, X)$

$t(\text{reflexive}(\textbf{attributeExpr}), X, Y) \mapsto$
$t(\textbf{attributeExpr}, X, Y)$ 'and' $t(\textbf{attributeExpr}, X, X)$

$t(\text{trans}(\textbf{attributeExpr}), X, Y) \mapsto t(\textbf{attributeExpr}, X, Y)$ 'impliedBy'
$t(\textbf{attributeExpr}, X, ?z)$ 'and' $t(\textbf{attributeExpr}, ?z, Y)$

$t(\text{and}(\textbf{attributeExpr}_1 \ldots \textbf{attributeExpr}_n), X, Y) \mapsto$
$t(\textbf{attributeExpr}_1, X, Y)$ 'and' ... 'and' $t(\textbf{attributeExpr}_n, X, Y)$

$t(\text{or}(\textbf{attributeExpr}_1 \ldots \textbf{attributeExpr}_n), X, Y) \mapsto$
$t(\textbf{attributeExpr}_1, X, Y)$ 'or' ... 'or' $t(\textbf{attributeExpr}_n, X, Y)$

$t(\text{not}(\textbf{attributeExpr}, X, Y) \mapsto$
'naf' $t(\textbf{attributeExpr}, X, Y)$

The transformation function for **classCondition**s and **attributeCondition**s have the following definitions ($attID$ is a meta-identifier which is replaced with the actual attribute identifier during translation):

$t(\text{attributeValueCondition}(\textbf{attributeID}, \textbf{individualID}), X) \mapsto$
X[t(attributeID) 'hasValue' t(individualID))]

$t(\text{attributeValueCondition}(\textbf{attributeID}, \textbf{dataLiteral}), X) \mapsto$
X[$t(\textbf{attributeID})$ 'hasValue' $t(\textbf{dataLiteral})$)]

$t(\text{attributeValueCondition}(\textbf{attributeID}, \textbf{classExpr}), X) \mapsto$
X[$t(\textbf{attributeID})$ 'hasValue' $?y$] **and** $t(\textbf{classExpr}, y)$

$t(\text{attributeOccurenceCondition}(\textbf{attributeID}), X) \mapsto$
$X[t(\textbf{attributeID})$ 'hasValue' $?x_{new}]$

$t(\text{valueCondition}(\textbf{individualID}), X, attID) \mapsto$
$X[attID$ 'hasValue' $t(\textbf{individualID})]$

$t(\text{valueCondition}(\textbf{dataLiteral}), X, attID) \mapsto$
$X[attID$ 'hasValue' $t(\textbf{dataLiteral})]$

$t(\text{valueCondition}(\textbf{classExpr}, X, attID) \mapsto$
$X[attID$ 'hasValue' $?y]$ 'and' $t(\textbf{classExpr}, ?y)$

$t(\text{expressionCondition}(\textbf{attributeExpr}),\ X, attID) \mapsto$
$t(\text{attributeExpr},\ X, attID)$

Having defined the transformations of expressions and conditions in WSML we can start defining the transformations for the actual mappings.

5.2.3 Relation mappings

Below the translations of relation mappings into WSML-Flight are specified. A two-way relation mapping is translated to two one-way mappings:

$t(relationMapping(\textit{two-way}\ \text{relationExpr}_1\ \text{relationExpr}_2$
$\text{relationCondition}_1\ \ldots\ \text{relationCondition}_m$
$\text{logicalExpression}_1\ \ldots\ \text{logicalExpression}_n)) \mapsto$
$t(relationMapping(\textit{one-way}\ \text{relationExpr}_1\ \text{relationExpr}_2$
$\text{relationCondition}_1\ \ldots\ \text{relationCondition}_m$
$\text{logicalExpression}_1\ \ldots\ \text{logicalExpression}_n))$
$t(relationMapping(\textit{one-way}\ \text{relationExpr}_2\ \text{relationExpr}_1$
$\text{relationCondition}_1\ \ldots\ \text{relationCondition}_m$
$\text{logicalExpression}_1\ \ldots\ \text{logicalExpression}_n))$

For each relation mapping, n new variables $(?x_1, ..., ?x_n)$ are introduced, where n is the arity of the relations. Notice that all relations in a relation mapping must have the same arity.

$t(relationMapping(\textit{one-way}\ \text{relationExpr}_1\ \text{relationExpr}_2$
$\text{relationCondition}_1\ \ldots\ \text{relationCondition}_m$
$\text{logicalExpresion}_1\ \ldots\ \text{logicalExpresion}_q)) \mapsto$
$t(\text{relationExpr}_1, ?x_1, ..., ?x_n)$ **impliedBy** $t(\text{relationExpr}_2, ?x_1, ..., ?x_n))$ 'and'
$t(\text{relationCondition}_1, ?x_1, ..., ?x_n, t(\text{relationID}_1))$ 'and' ... 'and'
$t(\text{relationCondition}_m, ?x_1, ..., ?x_n, t(\text{relationID}_m))$ 'and'
$t(\text{logicalExpression}_1, ?x_1, ..., ?x_n)$ 'and' ... 'and'
$t(\text{logicalExpression}_n, ?x_1, ..., ?x_n)$ '.'

The transformation functions for **relationExpr** and **relationCondition**s are the followings ($relID$ is a meta-identifier which is replaced with the actual attribute identifier during translation):

$t(\text{and}(\text{relationExpr}_1, ..., \text{relationExpr}_n), ?x_1, ..., ?x_n) \mapsto$
$t(\text{relationExpr}_1, ?x_1, ..., ?x_n)$ 'and' ... 'and' $t(\text{relationExpr}_1, ?x_1, ..., ?x_n)$

$t(\text{or}(\text{relationExpr}_1, ..., \text{relationExpr}_n), ?x_1, ..., ?x_n) \mapsto$
$t(\text{relationExpr}_1, ?x_1, ..., ?x_n)$ 'or' ... 'or' $t(\text{relationExpr}_1, ?x_1, ..., ?x_n)$

$t(\text{not}(\text{relationExpr}), ?x_1, ..., ?x_n) \mapsto$ 'naf' $t(\text{relationExpr}, ?x_1, ..., ?x_n)$

$t(\text{relationID}, ?x_1, ..., ?x_n) \mapsto \text{relationID}(?x_1, ..., ?x_n)$

$t(\text{parameterCondition}(\textbf{individualID}), ?x_1, ..., ?x_k, ..., ?x_n, \text{relID}) \mapsto$
$\text{relID}(?x_1, ..., ?x_k, ..., ?x_n)$ 'and' $?x_k = t(\textbf{individualID})$

$t(\text{parameterCondition}(\textbf{dataLiteral}), ?x_1, ..., ?x_k, ..., ?x_n, \text{relID}) \mapsto$
$\text{relID}(?x_1, ..., ?x_k, ..., ?x_n)$ 'and' $?x_k = t(\textbf{dataLiteral})$

$t(\text{parameterCondition}(\textbf{classExpr}), ?x_1, ..., ?x_k, ..., ?x_n, \text{relID}) \mapsto$
$\text{relID}(?x_1, ..., ?x_k, ..., ?x_n)$ 'and' $t(\textbf{classExpr}, ?x_k)$

$t(\text{expressionCondition}(\textbf{relationExpr}), ?x_1, ..., ?x_n, \text{relID}) \mapsto$
$t(\text{relID}, ?x_1, ..., ?x_n)$

5.2.4 Instance mappings

Instance mappings are not allowed in WSML-Flight, because equality in the head is not allowed in WSML-Flight.

5.2.5 Class-attribute mappings

Below the translations of class-attribute mappings into WSML-Flight are specified. Again, two-way mappings are translated into two one-way mappings.

$t($*classAttributeMapping*$($*two-way* classExpr attributeExpr
classAttributeMapping$_1$ ... classAttributeMapping$_n$
attributeMapping$_1$ ... attributeMapping$_m$
classCondition$_1$ ... classCondition$_p$
attributeCondition$_1$ ... attributeCondition$_q$
logicalExpresion$_1$ ... logicalExpresion$_s$)) $\mapsto$
$t($*classAttributeMapping*$($*one-way* classExpr attributeExpr
classAttributeMapping$_1$ ... classAttributeMapping$_n$
attributeMapping$_1$ ... attributeMapping$_m$
classCondition$_1$ ... classCondition$_p$
attributeCondition$_1$ ... attributeCondition$_q$
logicalExpresion$_1$ ... logicalExpresion$_s$))
$t($*classAttributeMapping*$($*one-way* attributeExpr classExpr
classAttributeMapping$_1$ ... classAttributeMapping$_n$
attributeMapping$_1$ ... attributeMapping$_m$

classCondition$_1$ ... classCondition$_p$
attributeCondition$_1$ ... attributeCondition$_q$
logicalExpresion$_1$ ... logicalExpresion$_s$))

t(*classAttributeMapping*(*one-way* classExpr attributeExpr
classAttributeMapping$_1$ ... classAttributeMapping$_n$
attributeMapping$_1$ ... attributeMapping$_m$
classCondition$_1$ ... classCondition$_p$
attributeCondition$_1$ ... attributeCondition$_q$
logicalExpresion$_1$ ... logicalExpresion$_s$)) $\mapsto$
(t(**attributeExpr**, $f(?x), ?y$) 'impliedBy' t(classExpr, $?x$) 'and
t(classCondition$_1$, $?x$) 'and' ... 'and' t(classCondition$_p$, $?x$) 'and'
t(attributeCondition$_1$, $?x$) 'and' ... 'and' t(attributeCondition$_n$, $?x$) 'and'
logicalExpresion$_1$ 'and' ... 'and' logicalExpresion$_s$) 'and'
t(classAttributeMapping$_1$, $?x, f(?x)$) 'and' ... 'and'
t(classAttributeMapping$_n$, $?x, f(?x)$) 'and'
t(attributeMapping$_1$, $?x, f(?x)$) 'and' ... 'and'
t(attributeMapping$_m$, $?x, f(?x)$) '.'

t(*classAttributeMapping*(*one-way* attributeExpr classExpr
classAttributeMapping$_1$ ... classAttributeMapping$_n$
attributeMapping$_1$ ... attributeMapping$_m$
classCondition$_1$ ... classCondition$_p$
attributeCondition$_1$ ... attributeCondition$_n$
logicalExpresion$_1$ ... logicalExpresion$_s$)) $\mapsto$
(t(classExpr, $f(?x)$) 'impliedBy' t(**attributeExpr**, $?x, ?y$) 'and
t(classCondition$_1$, $?x$) 'and' ... 'and' t(classCondition$_p$, $?x$) 'and'
t(attributeCondition$_1$, $?x$) 'and' ... 'and' t(attributeCondition$_n$, $?x$) 'and'
logicalExpresion$_1$ 'and' ... 'and' logicalExpresion$_s$) 'and'
t(classAttributeMapping$_1$, $?x, f(?x)$) 'and' ... 'and'
t(classAttributeMapping$_n$, $?x, f(?x)$) 'and'
t(attributeMapping$_1$, $?x, f(?x)$) 'and' ... 'and'
t(attributeMapping$_m$, $?x, f(?x)$) '.'

5.3 Mapping Examples

For a better understanding of the mapping language and of the way the mappings can be applied, this section provides an example of an ontologies mapping scenario, based on the running example presented in SEKT D4.3.1 [28].

In this example the ontologies to be mapped are expressed in WSML syntax and for the mappings both the abstract syntax, presented in section 5.1.3, and the WSML syntax are used.

5.3.1 Ontologies

We consider the following ontology as the source ontology:

concept Person
name **ofDataType** {0 1} xsd:string
age **ofDataType** {0 1} xsd:integer
hasGender **ofDataType** {0 1} gender
hasChild **ofType** Person
marriedTo **ofType** Person

datatype gender
range ofDataType xsd:string
definedBy
?x = "Male"^^xsd:string **or** ?x = "Female"^^xsd:string.

The target ontology is defined as follows:

concept Human
name **ofDataType** {0 1} xsd:string
age **ofDataType** {0 1} xsd:integer
noOfChildren **ofDataType** {0 1} xsd:integer

concept Marriage
hasParticipant **ofType** Human
date **ofDataType** {0 1} xsd:date
location **ofDataType** {0 1} xsd:string

concept Man **subConceptOf** Human

concept Woman **subConceptOf** Human

axiom ManWomanDisjointness
definedBy
constraint ?x **memberOf** Man **and memberOf** Woman.

The task is to conciliate the existing mismatches between these two ontologies by providing the appropriate set of mapping rules. First, the rules are represented by using the abstract syntax in order to illustrate the usage of our mapping language. Then, the rules are written in WSML for emphasizing the grounding of the mapping language to a formal representation language.

5.3.2 Ontology mappings represented by using abstract syntax

```
classMapping(o1:Person o2:Human
        attributeMapping(o1:name o2:name)
        attributeMapping(o1:age o2:age)
        )
```

```
classMapping(o1:Person o2:Man
        attributeValueCondition(hasGender "Male"^^xsd:string)
        )

classMapping (o1:Person o2:Woman
        attributeValueCondition(hasGender "Female"^^xsd:string)
        )

classAttributeMapping(o1:Person.marriedTo o2:Marriage
        classAttributeMapping(o1:Person o2:hasParticipant)
        attributeMapping(o1:marriedTo o2:hasParticipant)
        )
```

5.3.3 Ontology Mappings represented by using WSML syntax

?x **memberOf** o1:Person **implies**
?x **memberOf** o2:Human.
?x[o1:name **hasValue** ?y] **memberOf** o1:Person **implies**
?x[o2:name **hasValue** ?y]
?x[o1:age **hasValue** ?y] **memberOf** o1:Person **implies**
?x[o2:age **hasValue** ?y]

?x[o1:hasGender **hasValue** "Male" ^^xsd:string] **memberOf** o1:Person **implies**
?x **memberOf** o2:Man

?x[o1:hasGender **hasValue** "Female"^^xsd:string] **memberOf** o1:Person
implies ?x **memberOf** o2:Woman

?x[o1:marriedTo **hasValue** ?y] **memberOf** o1:Person **implies**
f(?x)[o2:hasParticipant **hasValue** {?x, ?y}] **memberOf** o2:Marriage

6 Resolving Ontology Heterogeneity Conflicts in Ontology Networks

In this chapter we review the existent reasoning approaches that deal with inconsistencies, analyze their pros and cons with respect to the purpose of this deliverable and finally conclude by choosing one of those approaches. In section 6.1, a general overview of inconsistencies within a Knowledge Base[1] (KB) is presented.

There are two main ways to deal with inconsistencies. The first one, called "*actual contradictions view*" [12], tolerates inconsistencies in the knowledge base by using non-standard reasoning methods to obtain meaningful answers (see [59] and [13]. Approaches that deal with inconsistencies in this way are described in section 6.2. The second approach called "*potential contradictions view*" [12] does not tolerate inconsistencies in a knowledge base. It assumes some mechanism to resolve conflicts, thereby obviating the potential of any contradiction. (see [94], [1], [44] and [3]). Approaches that deal with inconsistencies in the style of *potential contradictions view* are described in section 6.2.

6.1 Overview

Dealing with imperfect information is essential in most human intellectual activities [86]. [58] defines perfect information in the context of a Knowledge Base as follows:

Definition 5.1.1 (Perfect information): A Knowledge base KB is said to be perfect if and only if it is satisfied that

1. for no θ do we have θ , $\neg\theta \in$ KB *(consistent)*
2. if KB $\models \theta$ then $\theta \in$ KB *(exhaustive)*
3. for each θ, either $\theta \in$ KB or $\neg\theta \in$ KB *(crisp)*

Based on the previous definition [58] also identified three possible sources of imperfection:

1. inconsistency;
2. vagueness;
3. background ignorance

Following inverse order from the previous list, [62] characterizes "*background ignorance*" as the situation where the information of a concrete domain is incomplete. The second source of imperfection that [58] distinguished is "*vagueness*" defined by [95] as: "*Vagueness is standardly defined as the possession of borderline cases. When a term is applied to one of its borderline cases, the result is a statement that resists all attempts to settle whether it is true or false*". Three different kinds of vagueness are described in [62]:

[1] Here, a Knowledge Base is intended to be a set of first order logic formulas.

- *Ontic Vagueness*: assume that there are things or phenomena in the world that are themselves vague [53]
- *Cognitive vagueness*: support the fact that human perception is only an abstraction, it reflects a segmental piece of the world and subsequently it represents a vague "figure" of reality [52].
- *Linguistic vagueness*: language refers vaguely to objects or concepts (i.e. vague predicates like "he is tall") although they are not vague in nature. [92]

The last source of imperfection, inconsistency, is colloquially defined by "*the free dictionary*" [2] in the following way: "*inconsistency is the relation between propositions that cannot both be true at the same time*". Bremer [17]provides a formal characterization of an inconsistent set of statements: "A set of statements Γ is inconsistency iff it contains for some A both A and $\neg A$ as elements; i.e. $\exists A(A \in \Gamma \wedge \neg A \in \Gamma)$"

For each of the three sources of imperfection described above, [58] associates one or two logical approaches: *nonmonotonic logics* for background of ignorance; *many-valued logics* for vagueness; *paraconsistent logics and belief-revision* in the case of inconsistency.

Classical Logics have been designed to formalize mathematical reasoning where axioms are rarely expanded and inferences are long and complex. In mathematics, truth is not invalidated by the addition of new axioms: a theorem that was proven once, stays proven [46]. In everyday life, humans commonly use a different reasoning pattern called "*defeasible inference*" (or also "*common-sense reasoning*") where the reasoning process tries to achieve proper conclusions tentatively, reserving the right to retract them in the light of further information [2]. These inferences are called "*non-monotonic*" because the addition of new facts can reduce the set of logical conclusions. Such inferences are related with the attempts to clarify reasoning in taxonomic hierarchies [97], the meaning of the closed world assumption in databases [89], and the semantics of the negation as failure operator of logic programming [26]. More powerful and general non-monotonic reasoning formalisms like Circumscription [76], Default Logic [90], and Non-monotonic Modal Logics ([77] and [80] were proposed in the early eighties [46].

[58] stated that *many-valued logics* are the appropriate formalism to deal with Vagueness. Many-valued logics are non-classical logics. Like classical logics, they accept the principle of truth-functionality: "*the truth of a compound sentence is determined by the truth values of its component sentences (and so remains unaffected when one of its component sentences is replaced by another sentence with the same truth value)*" [50]. However, many-valued logics are significantly different from classical logics: "*Many-valued logics do not restrict the number of truth values to only two: they allow for a larger set W of truth degrees.*" [50]

Finally, [87] and [17] (among others) claim that classical, monotonic logic is inappropriate to handle the issue of inconsistency because they are explosive, i.e. from an inconsistent set of logical formulae everything can be concluded. A logic (i.e. classical logic and intuitionist logic) is said to be *explosive* (i.e. not para-consistent) if any formula is a logical consequence of a contradiction or in other terms:

[2] http://www.thefreedictionary.com/

"Let ⊨ be a relation of logical consequence, defined either semantically or proof-theoretically. Let us say that ⊨ is *explosive* iff for every formula A and B, {A, ¬A} ⊨ B [87]

There are two main ways to deal with inconsistencies. The first one, called "*actual contradictions*" [12], tolerates inconsistencies in the knowledge base by using non-standard reasoning methods to obtain meaningful answers (see [59] and [13]. The second approach called "*potential contradictions*" [12], assumes some mechanism to resolve conflicts, thereby obviating the potential of any contradiction (see [94], [1], [44] and [3]).

6.2 Actual-Contradictions View

The approach of tolerating inconsistencies in the knowledge base by using non-standard reasoning methods to obtain meaningful answers is the core pattern that underlies many paraconsistent logics [13]. For instance, Levesque's limited reference [72] allows the interpretation of a language in which a term and its negation can belong to truth assignments. Extending the idea of Levesque's limited reference, Schaerf and Cadoli [93] propose S-3-entailment and S-1-entailment for approximate reasoning which was intended to accelerate the reasoning process. The main idea of Schaerf and Cadoli's approach is to identify a subset S (called approximation set) of the language used and restrict the reasoning space to the set of formulas that are specified by this subset S. Although the proposal of Schaerf and Cadoli can be extended to adopt a more classical reasoning procedure, their approach remains incomplete because they do not propose a proper methodology for constructing and extending the approximation sets.

A new framework for reasoning by inconsistencies was recently introduced by Marquis and Porquet [75]. Resource-bounded paraconsistent inference [75] extends Schaerf and Cadoli's S-3-entailment to avoid some of its limitations. The main elements of this new approach are: the substitution of S-3 logic to a more expressive paraconsistent language called J3; and the use of coherence based reasoning for inconsistency handling. The core idea of coherence based reasoning is to start with an inconsistent knowledge base and to apply two successive mechanisms, namely, a consolidation operation which generates and selects several consistent subsets of the knowledge base and an entailment relation which uses classical logic on the consistent subsets in order to deduce nontrivial conclusions. In [75], Marquis and Porquet propose several policies, e.g., the linear order policy and the lexicographic policy, for the selection of the consistent subsets.

As a part of the project SEKT, a method for reasoning with inconsistencies was developed, described by Huang et. al in [59]. In a nutshell, Huang et al's proposal for reasoning with inconsistencies selects a consistent subset from an inconsistent ontology, and then they apply either standard reasoning or S-3-entailment on the selected sub-theory to find meaningful answers. If the result is not satisfactory, the subset is extended while keeping consistency, and the reasoning process is executed again. For the selection process that results a new selection function, they investigate To identify these consistent subsets, they study different solutions proposed in the area of computational linguistic based in syntactic and semantic relevance to measure the relationship between two formulas.

6.2.1 Reasoning with an Inconsistent Ontology: a close look

In [59], the first step is to verify if the ontology is indeed inconsistent because if it is consistent, a reasoning procedure based on classical logics can be used. Given a query θ, the decision tree depicted in figure 6.1 is employed in order to check whether an ontology Σ is consistent. If both consequences $\Sigma \models \theta$ and $\Sigma \models \neg\theta$ can be generated, the ontology is inconsistent.

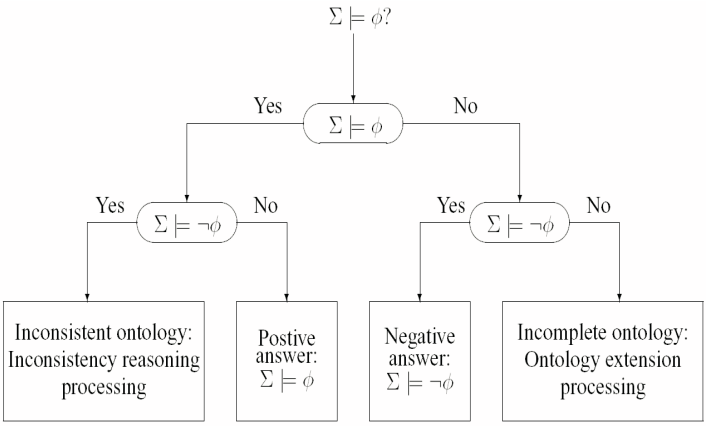

Figure 6.1: An overview of the integrated mismatches [59]

When the inconsistency of an ontology is determined, the reasoning process has to be adapted in order to enable the proper dealing with the inconsistency. Huang et. al. suggest two iterative approaches: linear extension with either classical entailment or S-3 entailment.

The strategy of linear extension with classical entailment depicted in figure 6.2 requires a monotone selection function that always returns increasing consistent subsets of the ontology. In an iterative process, the system tries to obtain a consistent subset, Σ", of the ontology Σ: if that is not possible the system returns the answer "*unknown*" and the process is finished; in the other case, the system uses a classical reasoner to prove the query, θ, in this consistent subset. If the query can be proved in the selected consistent subset, the system returns a positive answer and the process ends. If the system cannot prove the query, it can try to prove the negation of the query, $\neg\theta$ with two possible cases: if the system can prove the negation of the query, the system returns the negative answer and terminates; if the system cannot also prove the negation of the query, the system applies the selection function to extract a consistent super-set of the previous subset Σ". This new set it will be always subsumed by the ontology Σ, and the strategy described before (and graphically represented in figure 6.2) is executed again.

Similarly with Marquis and Porquet's approach [75], they use Belnap's four valued logic [9]:

- Over-determined $\Sigma \mid\!\approx \theta$ and $\Sigma \mid\!\approx \neg\theta$
- Accepted $\Sigma \mid\!\approx \theta$ and $\Sigma \mid\!\not\approx \neg\theta$
- Rejected $\Sigma \mid\!\not\approx \neg\theta$ and $\Sigma \mid\!\approx \neg\theta$
- Undetermined $\Sigma \mid\!\not\approx \neg\theta$ and $\Sigma \mid\!\not\approx \neg\theta$

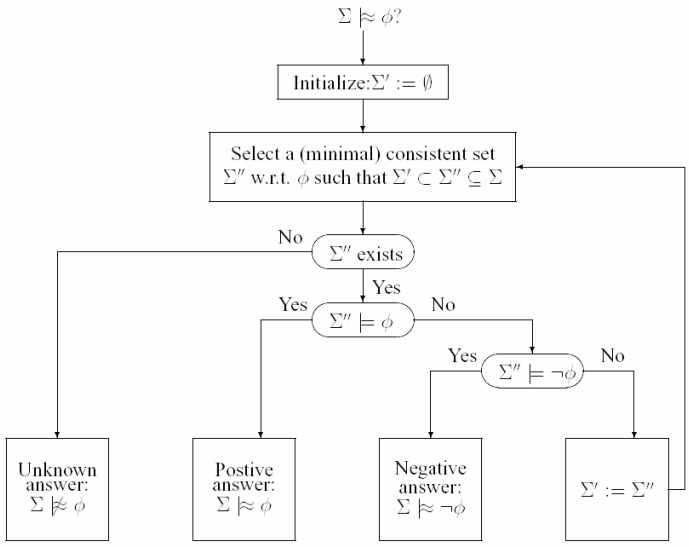

Figure 6.2: Reasoning with Inconsistencies: Linear extension with classical reasoning [59]

Huang et al highlight as a main disadvantage of the linear extension strategy that it can be undetermined, and they recommend to take into account an extension of their strategy by backtracking and heuristics evaluation in order to improve the results of the strategy.

The second method proposed to deal with inconsistencies is called "Linear extension with S-3 entailment" and is shown in figure 6.3. Opposed to the previous method, [59] proposes also to use S-3 entailment [93] for approximate reasoning in inconsistent ontologies. Given an inconsistent ontology Σ specified using a finite language $L(\Theta_0)$ which is generated on a primitive proposition set Θ_0, they called S an "*approximation set*" of Θ_0 if S is a subset of Θ_0. Depending on how S is defined the S-3 entailment can be appropriate to handle inconsistencies, and it is suitable for a selection function based on concept relevance. Moreover, Schaerf and Cadoli [93] show that the approximation of classical reasoning can be equivalent to approximate reasoning using S-3 entailment in the following proposition:

Proposition 5.2.1 ([93]): Let simplify -3(Σ,S) be the result of deleting all clauses of Σ which contain an atom outside S. Σ is S-3-satisfiable iff simplify -3(Σ,S) is classically satisfiable. S-3 entailment requires that formulas should be rewritten into negation normal forms (NNF). Using Marquis and Porquet's approach, there is no need to this restriction.

Both methods (linear extension with classical entailment and linear extension with S-3 entailment) satisfy the following properties: never over-determined, may be undetermined, always sound, always meaningful, always locally complete, may be not be maximal, and always locally sound [59].

Also, both methods require the existence of a selection function that on the one hand extracts increasing consistent subsets, and on the other hand identifies bigger approximation sets. Based on Chopra, Parikh, and Wassermann's relevance approach [25], Huang et al study also different solutions proposed in the area of computational linguistic based in syntactic and semantic relevance to measure the relationship between two formulas.

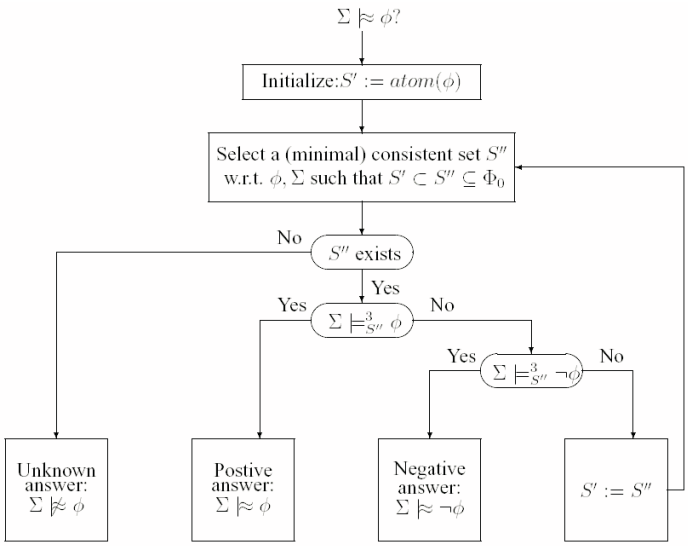

Figure 6.3: Reasoning with Inconsistencies: Linear extension with S-3-Entailment [59]

6.3 Potential-Contradictions View

Contrary to the actual-contradictions view, according to which the systematic eradication of contradictions, by detecting and remedying them, is not desired, the potential-contradictions view does not tolerate any inconsistencies[12]. In case contradictory statements still appear there exist two possibilities to deal with them:

- some of the statements involved in the contradiction can be found responsible for the contradiction. There is no indication in [12] about what can be done with those statements.

- the potential for contradiction is anticipated, the statements involved in the contradiction being indicative of conflicting arguments whose relative value is clear. The corresponding notion of inference is developed accordingly, in order to avoid contradictions actually taking place. Reasoning consists in trying to identify the "strongest argument" where all arguments that oppose it are discarded. If no strongest argument can be found, then all competing arguments can be discarded. This kind of reasoning is called *defeasible reasoning*. Nonmonotonic logics, like default logic[90] and defeasible logic[5, 4] are formalizing this kind of reasoning.

This section presents and discusses several approaches that deal with inconsistencies in the manner described by the potential-contradictions view. The presented approaches may be useful for identifying critical mappings and/or conflicting concept/relation definitions in the mapped ontologies.

In section 6.3.1 is discussed why solutions based on nonmonotonic logics, like those prescribed by the second possibility of the potential-contradictions view, have no use for our purpose.

A reasoning mechanism related with that of nonmonotonic logics is the mechanism associated to *belief revision*. An evaluation of belief revision is given in section 6.3.2.

In the next two sections we restrict ourselves at considering approaches that deal with the identification of contradictions, and either leave the task of correcting them

to the knowledge engineer or provide automatic means of correction that are external to the reasoning mechanism of the underlying logic.

In section 6.3.3 we present an approach that identifies minimal subsets of axioms responsible for contradictions that make part from unfoldable TBoxes in the language $\mathcal{ALC}$ DL. Also, a more precise localisation of contradictions is provided, by detecting the parts of these axioms that are actually responsible for contradictions. In section 6.3.4 the field of knowledge bases validation and refinement is explored, more specifically an approach that deals with the problem of evaluating and restoring the knowledge base coherency.

6.3.1 Ontology Heterogeneity Conflicts and Nonmonotonic Logics

Using non-monotonic logics for overcoming inconsistencies implies that classes of contradictory statements together with the solutions of their conflicts must be foreseen. Contradictions in our setting can appear only when parts of ontology modules are merged using mappings. We assume that no ontology module is inconsistent by itself. Though we may imagine scenarios where a priori knowledge that encodes preferences between statements from different ontologies is available [71], usually this is not the case. This excludes the use of nonmonotonic logics for solving inconsistencies between ontologies in the general case, because such knowledge would be required. However, we must acknowledge that when ontologies are expressed in a nonmonotonic formalism, some of the contradictions may be solved by the combination of defeasible rules from one ontology and the mappings from an external ontology to that ontology.

6.3.2 Belief Revision

Belief revision is the process of updating a knowledge base when new information is acquired. In case the new information is consistent with the previous knowledge base, the update consists just of adding the new information to the knowledge base. Otherwise some information has to be discarded in order for the new knowledge base to be consistent. It is this part of belief revision that interests us in the context of this deliverable: how information is discarded in case of an inconsistent resulting knowledge base.

One well-known framework for belief revision is the AGM framework [1, 43]. The framework is based on the *Principle of Minimal Change* that says that as much information should be conserved as possible in accordance with an underlying preference relation. Four types of changes are formalized in this framework: *contraction*, *withdrawal*, *expansion* and *revision*. These change functions are defined axiomatically, by describing their properties. For obtaining constructive definitions for each of the change functions, a preference relation named *epistemic entrenchment ordering* is needed. This provides extralogical information required for deciding which information should be discarded. It was shown by Gärdenfors and Makinson that for a finite language, an epistemic entrenchment of a theory is determined by the order of dual atoms (maximal disjunctions) in that theory. Such an order must be established in advance.

One problem that is frequently posed related to belief revision is how to perform iterated belief revision. In [3], a solution for enabling iterated revision within the AGM

framework is presented. This solution is based on the existence of a *finite partial entrenchment ranking* that grades the content of a finite knowledge base according to its epistemic importance. As discussed in [69] for all the attempts to logically characterize iterated belief change behavior, the belief base framework is not sufficient to encompass iterated revision. Additional information, like *epistemic states* to encode the agent beliefs but also its relative confidence in alternative states of the world is needed.

Thus, belief revision like the other approaches presented in this section is based on the principle that any representation should be free of contradictions[3]. Like in the case of nonmonotonic inferences, the semantics of belief revision can be described by a mechanism that selects minimal models in accordance with some underlying preference ordering. But also, as for defeasible logics, it is unrealistic to assume that rankings between statements from different knowledge bases exist. Still, it's interesting to note that as shown in [11], paraconsistent logics can be used for pinpointing contradictions, and thus taking part in the revision process. In this case they can be seen as an enrichment to the revision process, and not a rival approach.

6.3.3 Identifying Inconsistencies in $\mathcal{ALC}$ DL Unfoldable TBoxes

In this section we review a non-standard reasoning service for pinpointing to logical contradictions that was devised during the development of the medical terminology DICE [94]. No solution for correcting the contradictions is provided. We assume that the knowledge engineer is responsible for this task.

Since DICE was migrated to DL, more specifically to an unfoldable TBox in the $\mathcal{ALC}$ DL species, procedures for identifying the precise positions of errors within such a TBox were developed.

$\mathcal{ALC}$ is a simple yet relatively expressive DL with conjunction ($C \sqcap D$), disjunction ($C \sqcup D$), negation($\neg C$) and universal ($\forall r.C$) and existential quantification ($\exists r.C$). A TBox is called unfoldable if the left-hand sides of the axioms (the defined concepts) are atomic, and if the right-hand sides (the definitions) contain no direct or indirect reference to the defined concept [82].

A TBox is incoherent if there is at least one unsatisfiable concept that is interpreted as the empty set in all the models of the TBox. Two concepts were introduced for pinpointing to minimal subsets of axioms responsible for the incoherency of a TBox:

- *minimal unsatisfiability-preserving sub-TBoxes (MUPS)*: the smallest subsets of axioms of an incoherent terminology preserving unsatisfiability of a particular concept.
- *minimal incoherence-preserving sub-TBoxes (MIPS)*: the smallest subsets of axioms of an incoherent terminology preserving unsatisfiability of at least one concept.

Below are the formal definitions for these concepts:

Definition 5.3.1 Let A be a concept which is unsatisfiable in a TBox T. A set $T' \subseteq T$ is a *minimal unsatisfiability-preserving sub-TBox (MUPS)* of T if A is

unsatisfiable in T', and A is satisfiable in every sub-TBox $T'' \subseteq T'$.

Definition 5.3.2 Let T be an incoherent TBox. A TBox $T' \subseteq T$ is a *minimal incoherence-preserving sub-TBox (MIPS)* of T if T' is incoherent, and every sub-TBox $T'' \subseteq T'$ is coherent.

The most interesting axioms are those that appear in more MIPS, sets of such axioms being named *cores*. The arity of a core is defined as the number of MIPS in which it appears. Cores that have larger arities or maximal size are preferred.

An algorithm was designed for the computation of MUPS based on Boolean minimisation of terminological axioms needed to close a standard tableau ([6] Chapter 2). MIPS are calculated relatively simple using the computed MUPS.

Once the minimal sets of inconsistent axioms were identified, the next task is to pinpoint the axiom parts that are responsible for inconsistencies. The idea is to regard inconsistencies as being caused by concept overspecification. *Generalised incoherence-preserving terminologies* (GITS) are defined as TBoxes where the defining concepts of the axioms are maximally generalised without losing incoherence and the generalized definitions are syntactically related to the original axioms. By considering a specific syntactic relation (see [94] for details), all but one axiom per GIT can be trivially generalized to $A_i \dot{\sqsubseteq} \top$.

Definition 5.3.3 Let $T = \{C_1 \dot{\sqsubseteq} D_1, \ldots, C_n \dot{\sqsubseteq} D_n\}$ be an incoherent TBox. An incoherent TBox $T' = \{C_1 \dot{\sqsubseteq} D'_1, \ldots, C_n \dot{\sqsubseteq} D'_n\}$ is a *generalised incoherence-preserving terminology (GIT)* of T if, and only if,

- each D'_i is a syntactic generalization of $D_i (1 \leq i \leq n)$,
- every TBox $T'' = \{C_1 \dot{\sqsubseteq} D''_1, \ldots, C_n \dot{\sqsubseteq} D''_n\}$ with a syntactic generalization D''_i of D_i where $D'_i \dot{\sqsubseteq} D''_i$ and $D''_i \dot{\not\sqsubseteq} D'_i$ (for some $1 \leq i \leq n$) is coherent.

GITs with minimal size are preferred, where the size of a GIT refers either to the size of axioms or to the number of concept names occuring in it. Their computation is simplified by the observation that the set of GITs of a TBox $\mathcal{T}$ is equivalent to the union of GITs for the MIPS of $\mathcal{T}$.

The main disadvantage of this approach is that it deals only with a formalism with limited expressivity. It's obvious that it cannot be applied for formalisms that do not fall inside any DL subset like WSML Flight, WSML Rule, WSML Full and OWL Full. Since WSML Core corresponds to a subset of OWL Lite and WSML DL is envisaged as an alternate syntax for OWL DL, it remains to investigate whether this approach may be extended for the DL species that underlie OWL Lite and OWL DL, $\mathcal{SHIF}(\mathcal{D})$ DL, respectively $\mathcal{SHOIN}(\mathcal{D})$ DL. The authors conjecture that it is not difficult to extend the algorithms for MIPS and GITs computations for more expressive languages and for general TBoxes.

An algorithm for checking the consistency of an ABox with respect to a TBox in the $\mathcal{ALCN}$ DL is given in [6]. In [55] an algorithm for combined TBox and ABox reasoning in the $\mathcal{SHIQ}$ description logic is presented.

However the question remains how something like *minimum inconsistency preserving sets* can be computed. For $\mathcal{ALCN}$ DL, a possible solution can be developed using the approach from [54] where the consistency problem for $\mathcal{ALCN}$-ABoxes is reduced to satisfiability of $\mathcal{ALCN}$-concept descriptions. More specifically, in a preprocessing step, one applies the transformation rules used in the process of constructing a tableau for an ABox only to individuals present in the original ABox. Afterwards the role assertions are ignored and for each individual name in the initial ABox, the satisfiability algorithm is applied to the conjunction of its concept assertions. This is useful because it allows for locating more precisely the source(s) of inconsistencies for an inconsistent ABox.

6.3.4 Validation and Refinement in Knowledge Bases

Another approach related with the problem of identification of inconsistencies in a knowledge base is the *validation* and *refinement* of a knowledge base [35]. Validation concerns the evaluation of the knowledge base quality. Refinement is the process of locating causes of the KB inadequacy and finding potential modifications in order to improve its quality.

Different criteria can be considered for measuring the quality, but the approach discussed in this section analyzes KB *coherency*. With respect to this, refinement is a way to modify an incoherent knowledge base in order to restore its coherency.

In the validation field, the knowledge base is seen as a set of rules, while a set of facts constitutes the input. For validation purposes constraints are provided that specify which inputs should be never dealt with or what output must be obtained from a given input. The latter type of constraints are called *test constraints*.

The rules from a KB are *if-then rules* of the form "if $L_1 \wedge \ldots \wedge L_n$ then L_m", , where L_i are literals (propositional variables or their negations) or the symbol $\perp$. Each rule has associated a logical formula: "$L_1, \ldots, L_n \rightarrow L_m$". A KB has two types of rules, *expert rules* (rules that can be modified), denoted by the symbol r and *constraints* (rules that cannot be modified), denoted by the symbol R. An input is a literal. A test constraint (external to the KB) has the form "if $f_1 \wedge \ldots \wedge f_n$ then O", where $\{f_1, \ldots, f_n\}$ is an input fact base and O is a non-input fact. The inference relation is modus ponens.

Checking the coherence of a knowledge base amounts to checking the consistency of the knowledge base taken with each possible set of valid inputs and checking the test constraints. The definition below is a slightly simplified version of the definition for *VT_coherency* from [35].

Definition 5.3.4 A knowledge base KB is *VT_coherent* with respect to a set of test constraints R_{test} if:

- for each input fact base F_i satisfying the constraints, $R \cup R_{test}$: $F_i \cup KB \nvdash \perp$ (checking the consistency of the KB with each possible input).

- for each test constraint "if $f_1 \wedge \ldots \wedge f_n$ then O": $\{f_1, \ldots f_n\} \cup O \cup KB \nvdash \bot$ (checking test constraints).

The notion of coherence, as defined in this context, is not interesting per se for the scope of this deliverable. What's worthy to investigate is how refinement is performed, how the sources of incoherency are identified and repaired.

A possible cause of an incoherency is called *diagnosis*. A diagnosis is composed of a set of missing constraints (when the set of valid inputs is not sufficiently restricted) and a set of bad expert rules (when a test constraint is violated) such that, when the missing constraints are added to the KB and the bad rules are deleted from the KB, coherency is restored. Such a missing constraint is a conjunction of the input facts from an input fact base found to be inconsistent with the KB that implies false.

Minimal diagnoses are preferred. A particular class of such minimal diagnoses are defined. Compared with the approach described in the previous subsection, a minimal diagnosis is similar with a MIPS.

Repairing given a diagnosis consists of eliminating the bad rules and introducing the missing constraints. In this respect, this approach automatizes the repairing process. In a rather rudimental way, though.

Abstracting from the specific knowledge representation used in the presented approach, the remaining ideas are that sets of instances (inputs) are checked with respect to the schema (the set of rules) and the schema is checked with respect to a set of constraints. A set of instances that is not consistent with the schema is discarded (introducing a constraint that hinders the apparition of the input). Likewise a set of axioms that is not consistent with a distinguished set of the schema (constraints) is discarded.

Returning to our setting, a possible idea induced by this approach is to distinguish between a part of an ontology that can be discarded (the *expert* part) and a part that cannot be discarded (the *constraint* part). The distinction should be done by the ontology designer. Some parts of the inconsistencies may be solved in this way. When, as a result of a mapping, an inconsistency like an instance belonging to two disjoint classes arises, it may be the case that the statement that the instance belongs to a class makes part from the *expert* part of one ontology and the statement that the instance belongs to the disjoint class makes part from the *constraint* part of the other ontology. In this case the statement from the first ontology may be discarded. Still, this is a very limited way of solving inconsistencies: no solution can be applied if both statements have equal status and previous annotations of the ontologies statements must be available.

6.4 Rationale for the adopted approach

After a thorough study of the available approaches for dealing with inconsistencies we decided to adopt the approach of Huang et al [59]. We discard the approaches of the *potential contradictions view* because we don't intend to change the ontologies and the instance bases themselves. We want to stay in the very general environment where we don't have the possibility to change the ontologies and the instance bases, e.g. when we can only access them for reading and not for writing because they

belong to different organizations. The advantage of an approach which corrects the ontologies and the instance bases before processing queries, is that the processing of consecutive queries can be done much faster than with an approach like the one from Huang et al. There, for each query a very tedious process of iteratively searching for a consistent subset that answers the query has to be executed. This of course has a much higher complexity. But as said before, changing the ontologies is not possible in many scenarios and we want to stay at the most general level as possible.

Furthermore, as we are dealing with WSML Flight as our language to which each ontology is translated, the approach of Schlobach and Cornet [94] which deals with Description Logics is not appropriate for our purposes.

7 A General Framework for Reasoning with Ontology Networks

This chapter provides a framework for reasoning with ontology networks. First we analyze the usual reasoning tasks in the context of ontological reasoning within Description Logics as well as within Logic Programming based reasoners in Section 7.1. Since we have a grounding to WSML Flight for the mapping language, we mainly focus on reasoning within WSML Flight which is closer to Logic programming than to Description Logics[1].

A naive way to reason with ontology networks and map between the individual ontologies would be to insert all ontologies and the mappings between them into one common *reasoning space* and to apply either standard reasoning or the reasoning with inconsistencies framework outlined in Section 6.2. We decided to employ a method that can be more efficient though has the same worst case complexity. This method consists of traversing the ontology network and the mappings in order to find those parts relevant for the query. The method is described in section 7.2.

Finally, in section 7.3, we present a reasoning framework that integrates the traversal algorithm of section 7.2 and the reasoning with inconsistencies framework of Huang et al. [59].

7.1 General Reasoning Tasks

In this section we discuss general reasoning tasks in the context of Description Logics reasoners as well as Logic Programming Engines. Such an overview is in order since most commonly used ontology languages and reasoning engines used in the field of ontologies such as OWL, F-Logic and particularly WSML are based on these two logical paradigms. Both research areas, logic programming and Description logics, have identified decidable subsets of first order logics and several reasoners have been implemented for performing infernece in these frameworks.

Remarkably, WSML Flight, the ontology language we are focusing at in this work, tries to combine these two paradigms as follows:

- Take the Description Horn Logic $\mathcal{DHL}$ fragment of the Description Logic $\mathcal{SHOIN}$ which fits into the Datalog fragment of Logic Programming, (i.e. function-free Horn Logic). The sub language of WSML Flight which corresponds exactly to this fragment is WSML Core.
- Extend this fragment with other useful features of Logic Programming i.e. extending WSML Core to Datalog with integrity constraints, inequality in rule bodies and default negation.

For further details, we refer to the DIP Deliverable D 2.7 [63].

It is useful to compare different reasoning tasks addressed in the context of Logic Programming and Description Logics. As it runs out, both logic families are tailored

[1]WSML Core is a very restricted ontology language based on the Description Horn Logic fragment which roughly marks the intersection between the Description Logic $\mathcal{SHOIN}$ and the Datalog fragment of Logic Programming. WSML Flight is an extension of WSML Core towards Datalog with features such as rich datatype support, integrity constraints, inequality and default negation.

for different tasks. Whereas Description logics reasoners mainly focus on consistency (i.e. satisfiability) checking of a given Knowledge Base, retrieval of answers (instances) to a query is usually very expensive. Logic Progamming engines, closely related to the field of Deductive Databases, on the other hand, is tailored towards answering queries efficiently.

7.1.1 Reasoning with Description Logics

For Description Logics we distinguish TBox (Schema information) and ABox (instance information) reasoning. In general, every reasoning task except for instance retrieval can be reduced to satisfiability of a Description Logics Knowledge Base (i.e. ABox $\cup$ TBox).

- TBox Reasoning Services
  - Satisfiability checks whether a knowledge base is consistent, i.e. there is no concept which inferred to be necessarily empty [6].
  - Entailment checks whether some axiom A logically follows from the knowledge base KB, written $KB \models A$. Entailment can be reduced to checking satisfiability of $KB \cup \neg A$.
  - Strongly related to Satisfiability and Entailment are subsumption, equivalence and disjointness of concepts: Subsumption checks whether the concept C (subsumee) always denotes a subset of the concept D (subsumer), or in other words every instance of C is necessarily an instance of D [6]. Subsumption between concepts $C \sqsubseteq D$ wrt. a knowledge base KB can be reduced to checking entailment: $KB \models C \sqsubseteq D$. Equivalence of concepts or deciding disjointness between concepts can equally be reduced to subsumption checking, thus entailment respectively.
  - Classification is a useful reasoning service when we are interested to modify the knowledge base by automatically adding new concept expressions in the proper place in a taxonomic hierarchy of concepts. Given a concept C, identify its most specific subsumer and the most general subsumee in the hierarchy [6]. Certain DL reasoners such as RACER [2] can make use of classification in order to make subsumption queries more efficient. When all named classes are classified, subsumption is trivial.
- Abox Reasoning Services:
  - Checking consistency of an ABox w.r.t. a TBox
  - Instance checking (instantiation), which verifies whether a given individual (object), a, is an instance of (belongs to) a specified concept C. This task can be reduced to entailment and therefore to satisfiability ($KB \models C(a)$).
  - Retrieval service is to find the individuals in the knowledge base that are instances of a given concept (which of course includes all instances of subsumed concepts) [6]. This problem, as opposed to the others is not easily reducible to satisfiability. Actually, for finding all instances of a class in DL

[2] http://www.racer-systems.com/

one has to perform instance checking for *each* individual in the Knowledge Base. Since instance checking and thus entailment are paricularly expensive resoning tasks in DL (usually EXPTIME complete, typically implemented using optimized Tableaux techniques) usual Description logics reasoners perform badly on this task.

– Realization, which finds the most specific concepts (wrt. subsumption) that the individual is an instance of [6].

The TBox Reasoning Services, satisfiability and classification, can be easily reduced to subsumption. On the other hand, classification is a practical application of subsumption service between different concepts of a hierarchy [6].

Note that ABox and TBox reasoning is not as clearly separated as it might seem above. PArticularly in expressive Description Logics such as $\mathcal{SHOIN}$ (underlying OWL) which allow nominals and enumerated classes, the distinction is not that clear any longer.

7.1.2 Reasoning with Logic Programming

In this section we describe the reasoning tasks of Logic Programming. Logic Programming is optimized towards query answering where there exist two different types of queries, namely ground queries and open queries. A ground query consists of a knowledge base and a ground fact. The inference task is to check whether the ground fact is entailed by the knowledge base. An open query is a formula with free variables. The query answer consists of a number of substitutions for the variables with values from the knowledge base. An open query can be reduced to a number of ground queries, namely, a ground query for each of the facts in the knowledge base.

The particular reasoning tasks outlined above in the context of Description Logics relate to Logic Programming Engines as follows:

- instance check, i.e. checking if a ground fact is entailed by a Knowledge Base (given as a logic program): Instance checking can be viewed in the context of logic programming as query answering of a ground query.
- instance retrieval (find all individuals that have a set of characteristics): To some extent instance retrieval for a concept can be seen as a special case for answering monadic queries, when viewing concepts as unary predicates and the underlying ontology language is based in Logic Programming rather than DL. Similarly, retrieval of (binary) relation instances can be viewed as query answering.
- Consistency of the Knowledge Base: Logic Programming without extensions does not deal with inconsistency: Horn programs always have a single unique Herbrand model. When extending with integrity constraints, these are usually formulated as queries, where the knowledge base is viewed to be inconsistent whenever one of these queries is satisfiable. Pure Logic Programming also has been extended by non-classical negation, so-called *negation as failure*, for which several semantics exist, the well-founded semantics [45] and the stable model semantics [47] being the most well-known.[3]

[3]Note that under the well-founded semantics there is no inconsistency at all (i.e., every logic program has a well-founded model), whereas the stable model semantics allows a notion of inconsistency, i.e. a program having no model at all.

- Query containment: The corresponding task to subsumption checking for Logic Programs is query containment for monadic queries. However, Logic Programming engines do not provide support for these kinds of checks. In fact for extensions of logic programming above pure Datalog query containment is undecidable. However, when staying within Datalog, for checking subsumption of named classes one can simply introduce a new individual, assert membership of the subsumed class and query for membership of the subsuming class [51]. In order to check subsumption for complex descriptions, a new rule need to be added to the knowledge base. These techniques [51, 100] effectively reduce subsumption reasoning to query answering in a deductive database but in general DL resoners are better suited for subsumption checking.

For an overview of reasoners for Description Logics and Logic Programming engines and their strengths we refer to[29].

7.2 Reasoning over Ontology Networks

This section discusses how we can reason with an ontology network and the mappings. Finally we decide to traverse the ontology network and the mappings and set up a so-called *reasoning space* which contains only the parts that are relevant for the query and which is used for answering the query. Given a network of ontologies connected by mappings, i.e. a set of ontology modules, we want to see how reasoning can be performed using these distributed sources of knowledge. The prerequisite is that all the ontologies are imported into a common language. We use WSML Flight as the common language because we have provided a WSML Flight grounding for the mapping language in section 5.2 and we want to reason with both mappings and ontological knowledge. We discriminate between three approaches for reasoning over such networks:

1. query rewriting
2. reasoning over all the ontologies from an ontology space
3. reasoning over subsets of ontologies related to a given query - creating a reasoning space

A description for each approach together with the rationale for considering it or discarding it for the purpose of this deliverable is given in the next subsections. We discovered that the third approach offers the most advantages and we therefore provide a reasoning framework using this approach for reasoning over ontology networks in section 7.3.

7.2.1 Query rewriting

The approach discussed here is based on a paper by Calvanese et. al. [22] regarding query answering over information nodes connected by mappings. The paper shows how query answering using query rewriting over such information nodes, called there *peers*, should be performed. For simplicity, systems composed of only two peers are

considered in the paper. Among the two peers, one is called *the local peer*, and the other one *the remote peer*, the local one being the one to which the query is addressed.

Each peer P has the form $P = \langle K, V, M \rangle$ which means that it has associated a knowledge base K and exports a part of this knowledge base as a so-called view V to the clients willing to use the peer. Additionally, the peers have attached a set of mappings M. But in the paper the easier case is considered where only the local peer has a non-empty set of mappings. Each peer provides a query answering service that accepts queries expressed in a given query language over V. The mappings of the local peer have the form $q_r \rightsquigarrow q_l$, where q_r and q_l are queries of the same arity, the first one being a query accepted by the remote peer and the second one a query accepted by the local peer. A mapping assertion $q_r \rightsquigarrow q_l$ has an immediate interpretation in first order logic that states that $\forall x_1, ..., x_n \phi_r(x_1, ...x_n) \supset \phi_l(x_1, ..., x_n)$.

The task is to compute the certain answers $cert(q, T)$ to a query q over the theory T addressed to the local peer, by taking into account both peers knowledge bases and the mappings. Here, the set of certain answers is defined as $cert(q, T) = \{(c_1, ..., c_n) | (c_1, ..., c_n) \in q^I$ for all I such that $I \models T\}$. The idea is to rewrite this query into a set of queries accepted by the remote peer using the mappings in such a way that: the union of

1. the certain answers of the query with respect to the local peer's knowledge base and
2. of the certain answers of each of the rewritten queries with respect to the remote peer's knowledge base

is exactly the result of the query by taking into account all the information sources. I.e. $cert(q, K_l \cup K_r \cup M_l) = cert(q, K_l) \cup \bigcup_{i=1}^{n} cert(q_r^i, K_r)$.

An algorithm for query rewriting is provided that yields this result for a certain specialization of the framework. This specialization of the framework consists of the usage of

- the language L_K^O as the language for encoding the knowledge bases and
- the language of conjunctive queries as a query language.

L_K^O is a subset of FOL that captures features of frame-based knowledge representation formalisms and of ontology languages for the Semantic Web. The alphabet of the language is composed of constants and of unary and binary predicates, called *classes* and *roles*. The intensional component allows typical ontology constructs, like typing of roles, mandatory participation to roles for objects in a class, functionality of direct and inverse roles, and subsumption between classes. The extensional component allows facts and existential formulas, each assertion having one of the forms: $C(a)$, $R(a_1, a_2)$, $\exists x.C(x)$, $\exists x_1, x_2.R(x_1, x_2)$, $\exists x.R(x, a)$ where C and R are a class and a role and a, a_1 and a_2 are constants. Mappings are restricted to have one of the forms: $q_r \rightsquigarrow \{x | C(x)\}$ or $q_r \rightsquigarrow \{x_1, x_2 | R(x_1, x_2)\}$, where q_r is a query of arity 1 (resp. 2) over the exported fragment V of the remote peer, and C(resp. R) is a concept (resp. role) of the local peer.

The advantages of this approach are:

- for the specified languages the reasoning is correct

- for small queries the reasoning is scalable

The disadvantages of this approach are:

- the addressed queries are only conjunctive queries, while the envisioned type of queries for our framework is more varied.
- within the whole paper only consistent theries are considered, e.g. it is assumed that the theory formed by the two knowledge bases and the mappings is consistent. In contrast, this deliverable aims at presenting a way for reasoning in the presence of inconsistencies.
- it was shown by its authors that by allowing role hierarchies to appear within the intensional component of one peer, the described procedure for computing the result of the query does not work any more. In contrast, even the least expressive variants of our chosen languages OWL and WSML, OWL Lite and WSML Core, allow the appearance or relation hierarchies.
- with query rewriting it is not possible to answer a query in relation to all possible answers, e.g. it is not possible to compute the transitive hull of a predicate because the query is processed only locally within each ontology and then the answers are only unified.

Since we want to support a variety of queries in WSML Flight and this deliverable aims at offering a solution for answering queries even in the presence of inconsistencies, the disadvantages of this approach are very severe and rule it out of the set of approaches that are applicable in our setting. c

7.2.2 Reasoning with all ontologies and mappings as a whole

A straightforward approach that will yield correct and complete reasoning is to insert all the ontologies from the ontology network and the mappings that connect them into the *reasoning space*, which is the space of logical formulas we consider for answering the query. An alternative would be to consider only the external ontologies to the ontology the query is addressed to.

The advantages of this approach are:

- the reasoning is correct
- the approach works for all kinds of queries
- the approach works for WSML Flight and OWL DL as well.

The disadvantage of this approach is that it has a very high worst case complexity. There may be many ontologies, their size being relatively high, with many interconnections having to be represented by a lot of mappings. As we will see in the next section, there is a possibility to have much better best case complexity in considering only these parts of the ontologies and the mappings that are relevant to the query. Though, the worst case complexity remains the same because it is possible that everything is relevant to the query.

7.2.3 Reasoning over a reasoning space

In the following, an approach is presented which extracts the parts of the ontologies and the mappings that are relevant for the query. All the information pertinent to the query will be inserted into the initially empty *reasoning space*.

Recall that all ontologies have been translated to WSML Flight which can be rewritten into Datalog without loss of expressivity as it corresponds semantically to Datalog. Our mapping rules are written in WSML Flight with function symbols which means that we can translate them without loss of expressivity into Datalog with function symbols. Therefore, we can identify the query relevant parts of the ontologies and the mappings by means of a dependency graph[4] of the corresponding Datalog Programs. By taking the subgraph of the dependency graph which consists of all the predicates on which the query predicates transitively depend, we get all relevant predicates. By inserting all the rules and facts which contain relevant predicates into the reasoning space, we end up with a reasoning space which contains all parts of the ontologies and the mappings that are relevant to the query. Actually, we don't insert all the facts into the reasoning space when processing the dependency graph. We assume that the ontologies are already separated into an EDB and an IDB[5]. We don't have to import explicitly the instances of EDB which contain a relevant predicate, because we can retrieve them from the databases as need be while reasoning.

The process of gathering the relevant parts of the ontologies and the mappings has the complexity of $O(n)$ where n is the number of rules in the Datalog programs, i.e. the number of rules that are contained in all ontologies and in the mappings.

Compared with the previous approach, this approach can be much more efficient, e.g. when only a few ontologies are relevant for the query. Therefore, we choose this approach for setting up the reasoning space which we use for answering the query.

7.3 A Reasoning Framework

In this section we present a general framework for reasoning with ontology networks whose interrelationships are represented by means of mappings. Our framework also specifies how we can deal with inconsistencies due to conflicting and complementary modelling decisions of the ontology designers. Which kind of mismatches can occur has been described in chapter 4. We assume that the ontologies themselves are consistent and inconsistencies can only be introduced when reasoning with more than one ontology of the network.

In order to take into account that more than one ontology of the network models the same part of the domain, we first have to track down mappings that describe the overlaps between the individual ontologies in the network. Note that if there is no overlap, they are completely independent from each other. Then, on the one hand, we don't need mappings and on the other hand, no inconsistencies can arise. We describe the overlaps between the ontologies by means of the mapping language that is described in chapter 5.

[4] A dependency graph is a common construct in deductive databases (cf. [98]) which captures the dependency of the predicates in a logical program.

[5] In general deductive database notation an EDB is an abbreviation for extensional database which is the set of ground facts and IDB is the intensional database which is the set of rules

We deal with heterogeneous ontology networks but restrict the potentially used languages to the OWL and the WSML variants. In order to be able to reason with the ontologies and to facilitate the process of searching for similarities and mappings between the ontologies, we import them into a common format. Because we have a grounding of the mapping language into WSML Flight, we suggest to use WSML Flight as the common format.

After importing the ontologies into WSML Flight, the mappings are specified. The mappings can be generated either manually or semi-automatically or fully automatically. In each case first the similarities have to be detected and then mappings can be specified by means of the similarities. For finding similarities semi- or fully automatically, many systems use the *Match* operator (cf. section 1.1). After (potential) similarities between the ontologies of the Network have been found, the mappings can be specified semi-automatically by means of special tools, e.g. PROMPT (cf. [83]. It is also possible to use a learning algorithm to discover ontology similarities and mappings. For a thorough elaboration on ontology mappings please be referred to [32, 28, 27]. It goes beyond the scope of this deliverable to analyze and recommend a special technique for finding similarities and mappings between ontologies.

After having specified the mappings between the ontologies in the network, we can create a reasoning space to answer a query. By means of the Datalog dependency graph[6] we identify the parts of the ontologies and the mappings that are relevant to the query according to the technique presented in Section 7.2. The parts of the ontologies and the mappings that are identified as relevant for the query are inserted into a memory space which has been provided in advance and which we call *Reasoning Space*. In the worst case, we have to include the whole ontologies and mappings into the reasoning space. Here, we distinguish between the ontologies that model an instance base and the instance base itself. I.e. we assume that the instance base (or here, in the case of Datalog the EDB) is stored separately from the ontology schema (or here, in the case of Datalog the IDB). Therefore, we don't have to copy huge instance bases into our reasoning space although we reason with the instance bases as well.

After the reasoning space is filled up with all parts of all mappings and all ontologies of the network that affect the query, we can start to reason. As said above, we reason also with the instance bases of the ontologies but don't include them into the reasoning space because that would be too inefficient. Within the reasoning space, inconsistencies can occur although the parts of the ontologies are in themselves still consistent concerning the query. But as a whole they can bear inconsistencies. In order to be able to answer queries even when inconsistencies arise, we adopt the framework for reasoning with inconsistencies developed in the course of the SEKT project [59].

The framework for reasoning with inconsistencies consists of identifying a part of the inconsistent ontology that is consistent and can therefore give an answer to our query. The identification of the consistent part of the ontology that can give an answer to our query is iterative: one starts with a very small consistent set (either the empty set or the set $atom(\phi)$; cf. figures 6.2 and 6.3) and extends it linearly to the least larger consistent set that contains the old set. With the new set one tries to get a positive or negative answer. If this is not possible, the set has to be increased again. It is increased as long as there exists a least larger set that contains the old set. Huang et al developed two inconsistency reasoners: one that relies on classical reasoning and

[6]Recall that WSML Flight corresponds semantically to Datalog. Therefore, we can build a dependency graph for the ontologies and the mappings.

one that relies on reasoning with S-3-entailment [93]. These two reasoners are depicted in figure 6.2 and figure 6.3. By means of an in-depth empirical analysis Huang et al found out that the inconsistency reasoner based on S3-entailment didn't perform as well as the one based on classical entailment in the general case. Therefore, we choose the inconsistency reasoner based on classical entailment for our framework.

If the reasoning space is consistent, we don't need to execute the much more time consuming reasoning with inconsistencies. Instead, we can perform classical reasoning. Therefore, we first have to check if the reasoning space is indeed inconsistent. This preprocessing is done by means of a classical reasoner in the way that is depicted in figure 6.1.

A flow of the whole process is depicted in figure 7.1.

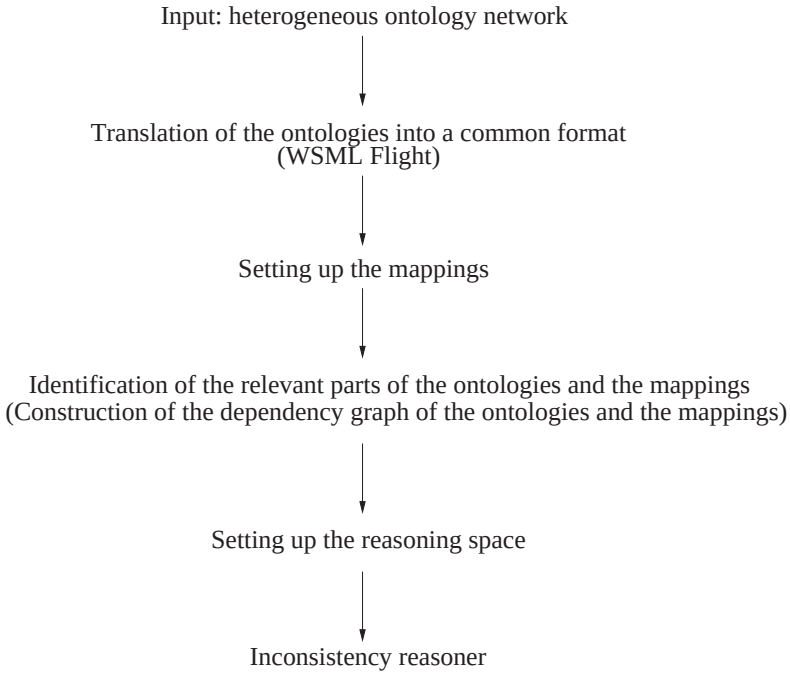

Figure 7.1: The processing flow for dealing with heterogeneous ontology networks

In section 7.1 a summary of several possible reasoning tasks is presented. Here, as our ontologies and the mappings are formulated in WSML Flight with function symbols or Datalog with function symbols, respectively, we consider only the typical reasoning tasks for logic programming. As mentioned above, reasoning with pure Description Logics is possible when using e.g. the grounding of the mapping language into OWL presented in [27, Section 5.5]. *Instance check* or query answering of a ground fact can be straightforwardly implemented by our framework by iteratively selecting increasing consistent subsets of the reasoning space until the instance is found. *Instance retrieval* is a little harder to realize. Here, obviously, we are not allowed to stop selecting increasing consistent subsets of the reasoning space until we haven't considered all monotonously increasing consistent subsets. Only if we consider monotonously increasing consistent subsets we can assure that the retrieved instances are not contradictory. Therefore, we are not allowed to do backtracking after having found the first instance and we have to proceed with the inconsistency reasoner as if we haven't found any instance at all.

8 Conclusions and outlook

In this deliverable we presented a way for representing ontology networks with mappings that deal with conflicting and complementary concept definitions. We did this by introducing a mapping language and by showing how to answer queries within heterogeneous networks with mappings. For this purpose we employed an inconsistency reasoner devised in the SEKT project which iteratively identifies consistent subsets of an inconsistent ontology.

In this deliverable, we provided a grounding of the mapping language to WSML Flight with function symbols. In the future a grounding to a higher WSML language can also be provided in order to retain even more expressibility. However, it has to be kept in mind that a grounding to a higher WSML language would worsen the tractability issue.

In D1.6 where an implementation of a WSML reasoner has to be provided, the Software modules presented in section 7.3 may be implemented. The reasoner can base on a Prolog engine, e.g. XSB. When implementing the framework it should be striven for an alignment of the ORDI framework [67] which is a framework for representing ontologies language independently. Since the framework stores the ontologies as pure strings and hands them over to the reasoners the integration of the ORDI framework into our reasoning with heterogeneous ontologies setting should not be a problem.

Acknowledgements

We would like to thank Francois Scharffe for devising the examples for the heterogeneity mismatches in section 4.2.

References

[1] C. E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510 – 530, 1985.

[2] A. Antonelli. Non-monotonic logic. Entry in the Stanford Encyclopedia of Philosophy, 2001. http://plato.stanford.edu/entries/logic-nonmonotonic/.

[3] Grigoris Antoniou. *Nonmonotonic Reasoning*. MIT Press, 1997.

[4] Grigoris Antoniou, David Billington, Guido Governatori, and Michael J. Maher. Efficient defeasible reasoning systems. *International Journal of Tools with Artificial Intelligence*, 10(4):483–501, 2001.

[5] Grigoris Antoniou, David Billington, Guido Governatori, and Michael J. Maher. Representation results for defeasible logic. *ACM Transactions on Computational Logic*, 2(2):255–287, 2001.

[6] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[7] Sean Bechhofer, Frank van Harmelen, Jim Hendlera, Ian Horrocks, and et all. Owl web ontology language reference, recommendation 10 feb. 2004. In *W3C Recomendation, http://www.w3.org/TR/owl-ref/*, 2004.

[8] Dave Beckett. RDF/XML syntax specification (revised). Recommendation 10 February 2004, W3C, 2003.

[9] N. Belnap. *Uses of Multiple-Valued Logic*, chapter A useful four-valued logic, pages 8–37. Reidel, Dordrecht, 1977.

[10] M. Benerecetti, P. Bouquet, and C. Ghidini. Contextual reasoning distilled. *Theoretical and Experimental Artificial Intelligence*, 12(3):279 – 305, 2000.

[11] P. Besnard and E. Laenens. A knowledge representation perspective: Logics for paraconsistent reasoning. *International Journal of Intelligent Systems*, 9:153 – 168, 1994.

[12] Phillipe Besnard and Anthony Hunter, editors. *Handbook of Defeasible Reasoning and Uncertainty Management Systems, Volume 2: Reasoning with Actual and Potential Contradictions*. Kluwer Academic Publishers, 1998.

[13] J. Beziau. *Frontiers of paraconsistent logic*, chapter What is paraconsistent logic, pages 95–111. Research Studies Press, Baldock, 2000.

[14] Anthony J. Bonner and Michael Kifer. A logic for programming database transactions. pages 117–166, 1998.

[15] Alex Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.

[16] Paolo Bouquet, Fausto Giunchiglia, Frank van Harmelen, Luciano Serafini, and Heiner Stuckenschmidt. C-OWL: Contextualizing ontologies. In K. Sekara and J. Mylopoulis, editors, *Proceedings of the Second International Semantic Web Conference*, number 2870 in Lecture Notes in Computer Science, pages 164–179. Springer Verlag, October 2003.

[17] M. Bremer. Lectures in paraconsistent logics, 2004. http://www.mbph.homepage.t-online.de/ParaLec.htm.

[18] Dan Brickley and Ramanathan V. Guha. RDF vocabulary description language 1.0: RDF schema. Recommendation 10 February 2004, W3C, 2004. Available from http://www.w3.org/TR/rdf-schema/.

[19] Jeen Broekstra, Michel C. A. Klein, Stefan Decker, Dieter Fensel, Frank van Harmelen, and Ian Horrocks. Enabling knowledge representation on the web by extending RDF schema. In *World Wide Web*, pages 467–478, 2001.

[20] Jos De Bruijn, Axel Polleres, Rubén Lara, and Dieter Fensel. OWL DL vs. OWL Flight: Conceptual modeling and reasoning for the semantic web. Technical Report DERI-TR-2004-11-10, DERI, November 2004.

[21] F. Giunchiglia C. Ghidini. A semantics for abstraction. In *Proceedings of the 16th European conference on Artificial Intelligence (ECAI-04) Valencia*, 2004.

[22] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Query reformulation over ontology-based peers. In *Proc. of the 12th Italian Conf. on Database Systems (SEBD 2004)*, 2004.

[23] Hans Chalupsky. Ontomorph: A translation system for symbolic knowledge. In *Principles of Knowledge Representation and Reasoning*, pages 471–482, 2000.

[24] Weidong Chen, Michael Kifer, and David Scott Warren. HILOG: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, 1993.

[25] S. Chopra, R. Parikh, and R. Wassermann. Approximate belief revision preliminary report. Journal of igpl, 2000.

[26] K. Clark. *Logic and Data Bases*, chapter Negation as failure, pages 293–322. Plenum, New York, 1978.

[27] Jos de Bruijn. Ontology mediation management. SEKT Project Deliverable D4.4.1, Digital Enterprise Research Institute, University of Innsbruck, 2004.

[28] Jos de Bruijn. Ontology mediation patterns. SEKT Project Deliverable D4.3.1, Digital Enterprise Research Institute, University of Innsbruck, 2004.

[29] Jos de Bruijn, Cristina Feier, Uwe Keller, Ruben Lara, Axel Polleres, and Livia Predoiu. WSML Reasoner Implementation. Deliverable D16.2v0.2, WSML, http://www.wsmo.org/wsml/, 2004. Available from http://www.wsmo.org/2004/d16.2/v0.2/.

[30] Jos de Bruijn, Douglas Foxvog, Holger Lausen, Eyal Oren, Dumitru Roman, and Dieter Fensel. The WSML Family of Representation Languages. Deliverable D16v0.2, WSML, http://www.wsmo.org/wsml/, 2004. Available from http://www.wsmo.org/2004/d16/v0.2/.

[31] Jos de Bruijn, Douglas Foxvog, Holger Lausen, Eyal Oren, Dumitru Roman, and Dieter Fensel. The WSML family of representation languages. Deliverable D16.0, WSML, http://www.wsmo.org/wsml/, 2004. Available from http://www.wsmo.org/2004/d16/v0.2/.

[32] Jos de Bruijn, Francisco Martin-Recuerda, Dimitar Manov, and Marc Ehrig. State-of-the-art survey on ontology merging and aligning v1. SEKT Project Deliverable D4.2.1, Digital Enterprise Research Institute, University of Innsbruck, 2004.

[33] Jos de Bruijn and Axel Polleres. Towards and ontology mapping language for the semantic web. Technical Report DERI-2004-06-30, DERI, June 2004.

[34] Jos de Bruijn, Axel Polleres, Rubén Lara, and Dieter Fensel. OWL flight. Deliverable D20.3v0.1, WSML, 2004.
Available from http://www.wsmo.org/2004/d20/d20.3/v0.1/.

[35] F. Dupin de Saint-Cyr and S. Loiseau. Validation and refinement versus revision. In Anca I. Vermesan and Frans Coenen, editors, *Validation and Verification of Knowledge Based Systems - Theory, Tools and Practice, Collected papers from EUROVAV '99, 5th European Symposium on Validation and Verification of Knowledge Based Systems,June 9-11, 199, Oslo, Norway*. Kluwer, 1999.

[36] Mike Dean and Guus Schreiber, editors. *OWL Web Ontology Language Reference*. 2004. W3C Recommendation 10 February 2004.

[37] Ying Ding, Dieter Fensel, Michel C. A. Klein, and Borys Omelayenko. The semantic web: yet another hip? *Data Knowledge Engineering*, 41(2-3):205–227, 2002.

[38] AnHai Doan, Jazant Madhaven, Pedro Domingos, and Alon Halevy. Ontology matching: A machine learning approach. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies in Information Systems*, pages 397–416. Springer-Verlag, 2004.

[39] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. AL-log: integrating datalog and description logics. *Journal of Intelligent Information Systems*, 10:227–252, 1998.

[40] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.

[41] Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. In *Proc. of the International Conference of Knowledge Representation and Reasoning (KR04)*, 2004.

[42] Jerome Euzenat. Towards a principled approach to semantic interoperability. In Asuncion Gomez-Perez, Michael Gruninger, Heiner Stuckenschmidt, and Michael Uschold, editors, *Workshop on Ontologies and Information Sharing, IJCAI'01*, page 2925, Seattle, USA, August 4–5, 2001.

[43] P. Gärdenfors, editor. *Knowledge in flux*. 1988. MIT Press.

[44] P. Gardenfors. *Belief Revision*. Cambridge University Press, Cambridge (MA), 1992.

[45] Allen Van Gelder, Kenneth Ross, and John S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proc. 7th ACM Symposium on Principles of Database Systems*, pages 221–230, Austin, Texas, 1988. ACM.

[46] M. Gelfond and R. Watson. *Encyclopedia of Cognitive Science*, chapter Nonmonotonic Logic, pages 375–382. Nature Publishing Group - Macmillan Reference Inc, UK, 2003.

[47] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth Bowen, editors, *Proc. of the Fifth Int. Conf. on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.

[48] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. S-match: an algorithm and an implementation of semantic matching. In *Proceedings of ESWS'04*, number 3053 in LNCS, pages 61–75, Heraklion, Greece, 2004. Springer-Verlag.

[49] Chen Hian Goh. *Representing and reasoning about Semantic Conflicts in Heterogeneous Information Systems*. PhD thesis, Massachusetts Institute of Technology, 1997.

[50] S. Gottwald. Many-valued logics. Entry in the Stanford Encyclopedia of Philosophy, 2004.

[51] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description logic. In *Proc. Intl. Conf. on the World Wide Web (WWW-2003)*, Budapest, Hungary, 2003.

[52] J. A. Hampton. Similarity-based categorization and fuzziness of natural categories. In *Cognition*, volume 65, pages 137–165. 1998.

[53] K. Hawley. Vagueness and existence. In *Proceedings of the Aristotelian Society*, volume CII.2, pages 125–140, 2002.

[54] Bernhard Hollunder. Consistency checking reduced to satisfiability of concepts in terminological systems. *Annals of Mathematics and Artificial Intelligence*, 18(2–4):133 – 157, 1996.

[55] I. Horrocks, U. Sattler, and S. Tobies. Reasoning with individuals for the description logic SHIQ. In David MacAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, number 1831, Germany, 2000. Springer Verlag.

[56] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, Sanibel Island, Florida, 2003.

[57] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A semantic web rule language combining OWL and RuleML. Daml draft v0.5, DAML, 2003. Available from http://www.daml.org/2003/11/swrl/.

[58] H. Hosni. Some foundational issues on nonmonotonic reasoning. Lectures Notes, 2002. http://www.maths.man.ac.uk/ hykel/work/cnuce280602_2up.pdf.

[59] Zisheng Huang, Frank van Harmelen, Annette ten Teije, Perry Groot, and Cees Visser. Reasoning with inconsistent ontologies: a general framework. SEKT Project Deliverable D3.4.1.1., Vrije University Amsterdam, 2004.

[60] Mustafa Jarrar. Ontology modularization and composition. In *Workshop on Ontology Modularization and context, Dec. 14, 2004, Brussel, Belgium*, 2004.

[61] Vipul Kashyap and Amit Sheth. Semantic and schematic similarities between database objects: a context-based approach. *The VLDB Journal*, 5(4):276–304, 1996.

[62] M. Kavouras and E. Tomai. Understanding and reducing ontological vagueness in the geographic realm (position paper). October 2001. SVUG-01 Symposium.

[63] Uwe Keller. Ontology representation language. DIP Project Deliverable D2.7, University of Innsbruck (UIBK), 2004. Due to Month 12.

[64] Uwe Keller, Rubén Lara, and Axel Polleres. Wsmo discovery. Working Draft D5v1.0, WSMO, 2004. Available from http://www.wsmo.org/2004/d5/v1.0/.

[65] Michael Kifer, Geord Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *JACM*, 42(4):741–843, 1995.

[66] Won Kim and Jungyun Seo. Classifying schematic and data heterogeneity in multidatabase systems. *Computer*, 24(12):12–18, 1991.

[67] Atanas Kiriakov. Ordi. DIP Project Deliverable D2.2, Sirma AI Ltd. (Sirma), 2004.

[68] Michel Klein. Combining and relating ontologies: an analysis of problems and solutions. In Asuncion Gomez-Perez, Michael Gruninger, Heiner Stuckenschmidt, and Michael Uschold, editors, *Workshop on Ontologies and Information Sharing, IJCAI'01*, Seattle, USA, August 2001.

[69] Sébastien Konieczny and Ramón Pino Pérez. Some operators for iterated revision. *Lecture Notes in Computer Science*, 2143:498–509, 2001.

[70] Doug Lenat. The dimensions of context space, 1998.

[71] Ioan Alfred Letia, Cristina Feier, and Monica Acalovschi. Achieving competence by argumentation on rules for roles. In *ESAW04*, 2004.

[72] H. J. Levesque. A knowledge-level account of abduction. In *Proceedings of IJCAI'89*, pages 1061–1067, 1989.

[73] Alon Y. Levy and Marie-Christine Rousset. Combining horn rules and description logics in CARIN. *Artificial Intelligence*, 104:165 – 209, 1998.

[74] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *Proc. 27th Int. Conf. on Very Large Data Bases (VLDB)*, 2001.

[75] P. Marquis and N. Porquet. Resource-bounded paraconsistent inference. *Annals of Mathematics and Artificial Intelligence*, 39:349–384, 2003.

[76] J. McCarthy. Circumscription: a form of non-monotonic reasoning. In *Artificial Intelligence*, volume 13 of *27-39*, pages 171–172. 1980.

[77] D. McDermott and J. Doyle. Non-monotonic logic i. In *Artificial Intelligence*, volume 13, pages 41–72. 1980.

[78] Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Developing metadata-intensive applications with rondo. *Journal of Web Semantics*, 1(1), December 2003.

[79] Prasenjit Mitra, Gio Wiederhold, and Martin L. Kersten. A graph-oriented model for articulation of ontology interdependencies. In *Proceedings of Conference on Extending Database Technology (EDBT 2000)*, Konstanz, Germany, March 2000.

[80] R. Moore. Semantical considerations on nonmonotonic logic. In *Artificial Intelligence*, volume 25, pages 75–94. 1985.

[81] Boris Motik, Ulrike Sattler, and Rudi Studer. Adding DL-safe rules to OWL DL. 2004. to be submitted.

[82] B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235 – 249, 1990.

[83] Natalya F. Noy and Mark A. Musen. Anchor-prompt: Using non-local context for semantic matching. In *Proceedings of the Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, Seattle, WA, USA, 2000.

[84] Jeff Z. Pan and Ian Horrocks. OWL-E: Extending OWL with expressive datatype expressions. IMG Technical Report IMG/2004/KR-SW-01/v1.0, Victoria University of Manchester, 2004. Available from http://dl-web.man.ac.uk/Doc/IMGTR-OWL-E.pdf.

[85] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL web ontology language semantics and abstract syntax. Recommendation 10 February 2004, W3C, 2004.

[86] D. Perlis. Sources of, and exploiting, inconsistency: preliminary report. *Journal of Applied Non-Classical Logics*, 7(1-2):25–75, 1997.

[87] G. Priest and K. Tanaka. Paraconsistency logic. Entry in the Stanford Encyclopedia of Philosophy, 2004. http://plato.stanford.edu/entries/logic-paraconsistent/.

[88] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, 2001.

[89] R. Reiter. On closed world databases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 119–140. Plenum, New York, 1978.

[90] Ray Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81 – 132, 1980.

[91] Dumitru Roman, Holger Lausen, and Uwe Keller. Web service modeling ontology standard (WSMO-standard). Working Draft D2v1.0, WSMO, 2004. Available from http://www.wsmo.org/2004/d2/v1.0/.

[92] B. Russell. Vagueness. *The Australasian Journal of Psychology and Philosophy*, 1:84–92, June 1923. http://www.santafe.edu/ shalizi/Russell/vagueness/.

[93] Marco Schaerf and Marco Cadoli. Tractable reasoning via approximation. *Artificial Intelligence*, 74(2):249–310, 1995.

[94] S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proceedings of the eighteenth International Joint Conference on Artificial Intelligence, IJCAI'03*. Morgan Kaufmann, 2003.

[95] R. A. Sorensen. Vagueness. Entry in the Stanford Encyclopedia of Philosophy, 1997. http://plato.stanford.edu/entries/vagueness/.

[96] Heiner Stuckenschmidt and Michel Klein. Modularization of ontologies. In *WonderWeb: Ontology Infrastructure for the semantic Web, Del 21, V.0.6, May 14,2003*, 2003.

[97] D. Touretzky. *The Mathematics of Inheritance Systems*. Los Altos. Morgan Kaufmann, 1986.

[98] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume II*. Computer Science Press, 1989.

[99] Pepijn R. S. Visser, Dean M. Jones, T. J. M. Bench-Capon, and M. J. R. Shave. An analysis of ontological mismatches: Heterogeneity versus interoperability. In *AAI 1997 Spring Symposium on Ontological Engineering*, 1997.

[100] Raphael Volz. *Web Ontology Reasoning with Logic Databases*. PhD thesis, AIFB, Karlsruhe, 2004.

[101] Holger Wache. *Semantische Mediation für heterogene Informationsquellen*. PhD thesis, University of Bremen, 2003.

[102] Gio Wiederhold. An algebra for ontology composition. In *Proc. of 1994 Monterey Workshop on Formal Methods*, 1994.