JAIST Repository
https://dspace.jaist.ac.jp/

Title	Semantic Enrichment in Ontologies for Analysis and Matching
Author(s)	Nwe, Ni Tun
Citation	
Issue Date	2007-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/3567
Rights	
Description	Supervisor:Satoshi Tojo, 情報科学研究科, 博士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

Semantic Enrichment in Ontologies for Analysis and Matching

by

Nwe Ni Tun

submitted to
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Professor Satoshi Tojo

School of Information Science
Japan Advanced Institute of Science and Technology

March, 2007

Abstract

Ontology—a formal, explicit, shared conceptualization of a domain—is intended to facilitate semantic interoperability among distributed and intelligent information systems where diverse software components, computing devices, knowledge, and data, are involved. Since a single global ontology is no longer sufficient to support a variety of tasks performed on differently conceptualized knowledge, ontologies have proliferated in multiple forms of heterogeneity—terminological heterogeneity, taxonomical heterogeneity, schematic heterogeneity, and instantiation heterogeneity—even for the same domain, and such ontologies are called *heterogeneous ontologies*. For interoperating among information systems through heterogeneous ontologies, mapping mechanisms need to bridge their knowledge gaps. *Ontology matching* (or mapping) is a process of finding correspondences between semantically related entities in heterogeneous ontologies.

The main aim of this research is *to deal with wide-scale semantic heterogeneity in ontology matching*. Although several efforts in ontology mapping have already been contributed, they have different focuses, assumptions, and limitations. A common point among existing methods is that possible correspondences between two ontologies are determined by the similarity of entity names; this is known as *name-based matching*. In order to decide semantic correspondences between concepts, those methods need to analyze the similarities between all related properties and instances; this is known as *content-based matching*. In the case of wide-scale semantic heterogeneity, content-based matching becomes complex, and user's approval or expert-interaction needs to verify mapping results.

In my research, I focus on two issues. The first issue is that the chance of correspondence between two terminologically quite different concepts is very less or not obtainable through name-based matching, because the name of a concept cannot express the precise semantics of the concept. In practice, two concepts with the same name may have different semantics, or two differently-naming concepts may have the same semantics. Thus, what is an alternative approach besides name-based matching, to find the possible correspondences between terminologically heterogeneous ontologies? The second issue is how to reduce complexity, concerning wide-scale semantic heterogeneity in content-based matching.

To accomplish the major aim and focuses, my underlying assumption is *the more explicit semantics is specified in ontologies, the feasibility of matching will be greater*. In order to improve the accuracy and automation of mapping processes, it is necessary that ontologies be well conceptualized with adequate semantics. Hence, an important step in handling semantic heterogeneity should be the attempt to enrich (and clarify) the semantics of concepts in ontologies.

Therefore, I proposed a semantically-enriched model of ontologies (called EnOntoModel) in which every domain concept is treated as a sort—an entity type that carries a criteria for determining the individuation, persistence, and identity of its instances—regarding every individual defined in a universe of discourse is countable and identifiable. In the philosophical literature, ontological concepts can be classified into four disjoint sort categories: type, quasi-type, role, and phase. I set up a logic-based formal system

to classify domain concepts into these sort categories, using three philosophical notions: identity, existential rigidity, and external dependency. In my research, this classification knowledge is intuitively represented as concept-level properties that are different from ordinary properties (called individual-level properties) which are used to specify individuals. Then, I defined EnOntoModel in which the semantics of domain concepts are described by using individual-level properties, as well as concept-level properties; this is my approach of *semantic enrichment*.

The innovation behind EnOntoModel is to supply an identifiable link between two heterogeneous descriptions of a concept, regarding that if two concepts are semantically equivalent, then they must be classified within the same sort category. For the usability of EnOntoModel, I implemented sortal meta-class ontology as an open source interface in enrichment process as well as conceptual analysis of enriched ontologies. By the aim of the thesis, I designed a matching method between enriched ontologies.

A novel idea of EnOntoModel-based Ontology Matching (EOM) method is that direct concept matching is driven between the same categories of sorts instead of exhaustive search among all sorts, because domain concepts are systematically classified into four disjoint sort categories. Moreover, it is examined that EnOntoModel can support not only determining the scope of possible correspondences, but also determining the most relevant properties which can certainly indicate a correspondence between two similar concepts. This means that semantic correspondences between highly heterogeneous concepts can be achieved without taking an exhaustive search in taxonomies and an analysis among all related properties. Consequently, EnOntoModel supports content-based matching in a less complexity.

The method is implemented in Java for matching between OWL ontologies by utilizing Jena OWL API and Protégé OWL API. The efficiency of EOM is evaluated in terms of mathematical complexity and proved that this method could reduce the complexity of the matching process by comparing it with other methods, particularly GLUE's content learners. Moreover, an experiment is done in two real data sets, and the effectiveness of EOM is shown in terms of precision and recall.

Acknowledgments

I would like to thank my supervisor, Professor Satoshi Tojo, for his superintendence throughout the whole study and research. I enjoyed the freedom that was given during the pursuit of my research directions, and I also wish to express my sincere gratitude for that to Professor Satoshi Tojo.

I thank Associate Professor Kiyoaki Shirai for supervision of my sub-theme work, and his valuable comments and prompt technical answers for this research study.

I am happy to express my warm gratitude to Professor Koichiro Ochimizu (JAIST) and Dr. Pyke Tin (UCSY, Myanmar) for their unfalteringly arrangement of a peaceful study in Japan Advanced Institute of Science and Technology (JAIST).

Since I was admitted as a student of JAIST's Graduate Research Program (GRP) which is supported by the special coordination funds from Japanese Ministry of Education, Culture, Sports, Science and Technology (MEXT) in order to promote innovative talent in emerging Science and Technology research areas, I thank Professor Sukekatsu UshiOda (President, JAIST), all GRP staffs principally Professor Takuya Katayama, and MEXT, for their willing financial support not only to study at JAIST, but also to present my research progress in the international conferences. I really appreciate Professor Koichiro Ochimizu and Professor Akira Shimazu, for their kindly effort to achieve an exceptional financial support in order to continue my research merrily and successfully.

I would like to thank Professor Akihiro Tamaki (Tokyo University of Information Science) who sowed seed this PhD spirit ambitiously together with Dr. Pyke Tin at the first batch of UCSY PhD Program (2001-2002). I would have never become a research scientist without being obedient to his cheerful philosophy, "without giving up, you must steadily climb up the mountain where you like to stand on". I also appreciate for his intellectual training of Personal Software Process (PSP) which is really worthy to manage my abilities timely-well in research and authoring works.

I am delighted to thank Professor Nicola Guarino (Laboratory of Applied Ontology, Italy), Natasha F. Noy (Senior Research Scientist, Stanford Medical Informatics), and Protégé developers and community members, who have made many effective suggestions for this research and spent their valuable time in the worthful discussions via emails.

A special thank to all defense program committee members, especially to Professor Koiti Hasida (National Institute of Advanced Industrial Science and Technology) and Associate Professor Kentaro Torisawa (JAIST), for their inspiring and challenging questions.

To Mary Ann Mooradian, from Technical Communication Department, who always assisted in editing English of my publications and this dissertation, I would like to say my happy thanks.

To Yoshita Suzuki, I am grateful for his proofreading especially for logical framework.

A warm thank to all friends and Knowledge Engineering Lab (Tojo & Torisawa-lab) members as well as the 24-hour system of JAIST library for a cheerful and companionable working atmosphere.

To my beloved parents and two brothers, I owe a great deal of thanks for their love and continuous encouragement to finish this research successfully.

Contents

Abstract i

Acknowledgments iii

1 Introduction 1
- 1.1 Motivation . . . 2
  - 1.1.1 The Semantic Web . . . 2
  - 1.1.2 Smart Spaces . . . 5
- 1.2 Problem Definition . . . 7
- 1.3 Objectives . . . 9
- 1.4 Approach and Scope . . . 9
- 1.5 Thesis Outline . . . 12

2 Ontological Engineering 13
- 2.1 Basic Ontology Concepts . . . 13
  - 2.1.1 What is an Ontology? . . . 14
  - 2.1.2 Which are the Main Components of an Ontology? . . . 15
  - 2.1.3 Categorization of Ontologies and their Uses . . . 17
- 2.2 Methodologies for Building Ontologies . . . 19
  - 2.2.1 TOVE Methodology . . . 19
  - 2.2.2 METHONTOLOGY . . . 21
- 2.3 Ontology Languages . . . 22
  - 2.3.1 Evolution . . . 22
  - 2.3.2 RDF-Resource Description Framework . . . 24
  - 2.3.3 OWL-Ontology Web language . . . 25
- 2.4 Ontology Development Tools and Tool Suites . . . 28
  - 2.4.1 SWOOP . . . 28
  - 2.4.2 Protégé . . . 29
  - 2.4.3 Concluding Remarks . . . 33

3 Ontology Matching and Semantic Heterogeneity 34
- 3.1 What is Ontology Matching . . . 34
- 3.2 Semantic Heterogeneity in Ontologies . . . 35
  - 3.2.1 Klein's Mismatches . . . 35
  - 3.2.2 Visser's Mismatches . . . 37
  - 3.2.3 An Alternative Classification of Semantic Heterogeneity . . . 39
- 3.3 Ontology Matching Tools and Techniques . . . 40
  - 3.3.1 MAFRA . . . 41

3.3.2 ONION . . . 42
3.3.3 PROMPT . . . 43
3.3.4 IF-Map . . . 45
3.3.5 COMA++ . . . 46
3.3.6 QOM . . . 46
3.3.7 GLUE . . . 47
3.3.8 Concluding Remarks . . . 48

4 Semantic Enrichment in Ontologies 50
4.1 A First-order Quantified Modal Language $\mathcal{L}^E$ . . . 51
4.1.1 Syntax . . . 52
4.1.2 Semantics . . . 53
4.2 Philosophical Foundations for Ontological Conceptualization . . . 57
4.2.1 Why Fundamental Ontological Classes are Sorts? . . . 57
4.2.2 Sortal Taxonomy and Subsumption Relationship . . . 58
4.2.3 Identity and Identity Condition . . . 60
4.2.4 Existential Rigidity . . . 64
4.2.5 External and Existential Dependency . . . 67
4.3 Modelling Semantically-enriched Ontologies . . . 69
4.3.1 Description of a Sort . . . 69
4.3.2 A Classification of Sorts . . . 70
4.3.3 Concept-level Properties of Sorts . . . 76
4.3.4 A Conceptual Model of Semantically-enriched Ontologies . . . 78
4.4 Implementation of a Sortal Meta-class Ontology . . . 84
4.4.1 Purpose and Scope . . . 85
4.4.2 Design and Implementation . . . 85
4.5 Development of Semantically-enriched Ontologies . . . 88

5 EnOntoModel-based Ontology Matching 95
5.1 Overview . . . 95
5.2 Matching Architecture . . . 97
5.3 Matching Method . . . 99
5.3.1 Why ICs are useful for matching between sorts? . . . 99
5.3.2 IC-based Type Sort Matching . . . 104
5.3.3 Matching of Quasi-type sorts, Role sorts, and Phase sorts . . . 107
5.4 Matching Algorithm . . . 111
5.5 Evaluation . . . 113
5.6 Experimental Results . . . 114
5.7 Related Work . . . 117
5.7.1 Ontology Matching . . . 117
5.7.2 Ontological Analysis . . . 120
5.8 Benefit and Cost of EnOntoModel . . . 121

6 Conclusion 123
6.1 Summary of Contributions . . . 123
6.2 Advantages and Limitations . . . 125
6.3 A Brief History of Research Progress . . . 126
6.4 Future Directions . . . 127

References **128**

Publications **144**

A Sortal Meta-class Ontology: `sort.owl` **145**

List of Figures

1.1 The layer cake of data representation standards for the Semantic Web . . . 3
1.2 An architecture of pervasive computing environment with smart spaces [96] 6

2.1 The ontology classification by Guarino . . . 17
2.2 Examples of top-level ontologies . . . 18
2.3 TOVE Ontologies excerpted from [111] . . . 20
2.4 The major development processes of TOVE Methodology . . . 20
2.5 The ontology development life cycle of METHONTOLOGY Methodology . 21
2.6 Tasks of conceptualization activity according to METHONTOLOGY . . . 22
2.7 A classification of ontology languages . . . 24
2.8 A simple view of RDF triple . . . 24
2.9 A sample RDF graph . . . 25
2.10 A usage of RDFS . . . 25
2.11 An example OWL ontology . . . 27
2.12 The architecture of SWOOP . . . 28
2.13 A screenshot of SWOOP Web Ontology Browser . . . 29
2.14 The architecture of OWL plug-in extended on Protégé core system . . . 30
2.15 A screenshot of the OWL class tab in Protégé . . . 31
2.16 A sample PAL constraint . . . 33

3.1 A view of ontology matching . . . 35
3.2 Different levels of ontology mismatches given in [117] . . . 36
3.3 A classification of ontology mismatches given in [168] . . . 37
3.4 A general illustration of semantic heterogeneity between two classes . . . 39
3.5 The conceptual architecture of MAFRA . . . 41
3.6 The UML representation of MAFRA's semantic-bridge-based ontology mapping [17] . . . 42
3.7 An example of matching by using an articulation rule in ONION . . . 43
3.8 A screenshot of PROMPT . . . 44
3.9 The ontology mapping approach of IF-Map . . . 45
3.10 The architecture of IF-Map . . . 46
3.11 The architecture of COMA++ [69] . . . 47
3.12 Mapping process of QOM . . . 47
3.13 The architecture of GLUE . . . 48

4.1 The outer and inner domains of possible worlds in an $S5$ Kripke model . . 55
4.2 A sortal taxonomy of genealogy domain . . . 59
4.3 An illustration of diachronic IC by Equation 4.2 and 4.3 . . . 61
4.4 OwnIC and CarriedIC through IC inheritance . . . 63

4.5 Rigid sorts and anti-rigid sorts . . . 67
4.6 An example for sort, properties, and individuals . . . 69
4.7 The domain and range of a property . . . 70
4.8 IC is one-to-one functional . . . 71
4.9 A representation of Sort and IC in OWL . . . 71
4.10 A classification of ontological concepts in OntoClean [147] . . . 72
4.11 A classification of universals by Guizzardi [61] . . . 73
4.12 A typical structure of sortal taxonomy . . . 75
4.13 A transition of essential processes in ontology development adopted from Blum (1996) . . . 76
4.14 The taxonomy of `Research-Community.owl` . . . 79
4.15 An example of the similar domain ontology . . . 84
4.16 A structure of sortal meta-classes . . . 85
4.17 A specification of RoleSort meta-class in Protégé . . . 87
4.18 The screenshot of "PAL constraint" editor . . . 88
4.19 Five PAL constraints in the meta-class ontology . . . 89
4.20 Major steps of semantic enrichment process . . . 90
4.21 A view of classes and properties in Proteégé . . . 92
4.22 A screenshot for importing ontology . . . 92
4.23 A screenshot of semantic enrichment in Protégé . . . 93
4.24 A screenshot of conceptual analysis using PAL constraints . . . 93
4.25 An example of enriched concepts in OWL . . . 94

5.1 An overview of enrichment-based matching . . . 95
5.2 A view of ontology matching . . . 96
5.3 Two mapping settings in query answering . . . 97
5.4 A general architecture of EnOntoModel-based ontology matching . . . 98
5.5 EnOntoModel-based matching heuristics . . . 99
5.6 The process flow diagram of matching function f . . . 100
5.7 Taxonomies of O and O' . . . 100
5.8 sort correspondences between O and O' . . . 103
5.9 The IC of sort s, ι_s, is exportable to sort s' . . . 104
5.10 The IC of sort s', $\iota_{s'}$, is importable to sort s . . . 104
5.11 Interchangeability between ι_s and $\iota_{s'}$ . . . 105
5.12 Sameness between ι_s and $\iota_{s'}$ . . . 106
5.13 IC-based type sort matching . . . 106
5.14 The process flow diagram of quasi-type sort matching . . . 108
5.15 The process flow diagram of role sort matching . . . 109
5.16 A view of phase sort matching through divide-and-conquer approach . . . 110
5.17 The average depth and number of leaves in a binary tree . . . 114
5.18 Experimental results in two domains . . . 116
5.19 EnOntoModel-based matching vs GLUE's Content-based matching . . . 120

List of Tables

3.1 Mapping tools (and methods) and their techniques of similarity analysis . . 49

4.1 Two kinds of concept-level properties by sort classification 77
4.2 Type sorts and their ownICs 80
4.3 Quasi-type sorts and their CVAs 80
4.4 Role sorts and their EDRs 81
4.5 Phase sorts and their CCs 81
4.6 Some individual-level properties of `Research-Community.owl` 83
4.7 Meta-classes and their properties 86
4.8 The meanings of some PAL keywords 87

5.1 Statistics of data sets 116
5.2 Precision (P) and recall (R) on data sets 116
5.3 Techniques of similarity analysis applied in each matching method 118

Chapter 1

Introduction

The term *ontology* is borrowed from philosophy, where it is known for *metaphysics*—the science of what is. By the early 1980s, researchers in AI had realized that ontology was relevant to intelligent systems for knowledge representation and reasoning [93]. Philosophical ontology seeks a classification for all types of entities, more precisely, the kinds and structures of objects, properties, events, processes, and relations, in every area of reality [29]. In Information Science, a more pragmatic view of ontologies is taken, where an ontology is considered a kind of consensus on a specific area of knowledge representation.

Ontologies represent the formal semantics of domain terms and their conceptualization in hierarchies. In a formal ontology, the description of each concept is explicitly given with a set of attributive properties together with restrictions. And then relationships among concepts set up a semantic net as ontology.

Today, Ontologies have become a silver bullet not only in the development of the Semantic Web, but also in several collaborative application areas such as Intelligent Environments or Smart Spaces, E-commerce, Multi-Agent Systems, Social Networks, etc., because they are respected as a means of consensus for efficient reasoning and sharing capabilities. Moreover, system interoperability is an important issue, widely recognized in information technology intensive enterprises and in the research community of information systems (IS). Increasing cooperation among organizations have created a need for many organizations to access remote as well as local information sources. Also, the wide adoption of the World Wide Web (WWW) needs interoperability to access and distribute information.

Since a single global ontology is no longer enough to support the variety of tasks pursued in distributed environments, ontologies have proliferated in multiple forms of heterogeneity even for the same domain. Thus, ISs face a trade off between interoperability and heterogeneity. In order to keep a balance between heterogeneity and interoperability, *ontology matching* (or mapping)—a process to find correspondences between semantically related entities among heterogeneous ontologies—has become a plausible solution in various tasks, such as ontology merging, query answering, information retrieval, exchange, and integration of BioInformatics, Medical Informatics, Security Informatics, Social Informatics, Computing Informatics, etc. To access multiple knowledge sources through heterogeneous ontologies, mapping mechanisms need to bridge their knowledge gaps. The main aim of this research is how to deal with wide-scale semantic heterogeneity in ontology matching.

1.1 Motivation

In this section, the development of ontologies with some challenges, which motivated this research are introduced concerning two generous visions: the Semantic Web [200] and Smart Spaces [96].

1.1.1 The Semantic Web

According to Google, the current World Wide Web (WWW) has well over 15.5 billion pages in 2006 [62]. However, the vast majority of them are in human readable format only, more precisely in eXtensible Markup Language (XML). XML provides a set of meta-data[1] tags to represent the semantics of web data, but XML does not define the meaning of the tags. Thus, the information available on the Web can be accessed only by syntactic interoperability. As a consequence, software agents cannot understand and process this information efficiently, and the potential of the Web has so far remained untapped. In 2001, Tim Berners-Lee and his colleagues set up the vision of the Semantic Web [200] as follows:

> *"....The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in co-operation."*
>
> *Tim Berners-Lee, James Hendler, Ora Lassila*
>
> *The Semantic Web, Scientific American, May 2001*

The idea is that ontologies allow users to organize information into taxonomies of concepts, each with their own properties, and to describe relationships between concepts. When data is represented using ontologies, software agents can better understand the content of the message, and therefore more intelligently locate and integrate data for a wide variety of tasks. Tim Berners-Lee illustrated an example of how the Semantic Web might be useful.

> *"...Suppose you want to compare the price and choice of flower bulbs that grow best in your zip code, or you want to search online catalogs from different manufacturers for equivalent replacement parts for Volvo 740. The raw information that may answer these questions, may indeed be on the Web, but it is not in a machine-usable form. You still need a person to discern the meaning of the information and its relevance to your needs....[201]"*

The Semantic Web can address this problem by requesting people to add knowledge to computers, to explain the relationships between different sets of meta-data. For example, one will be able to make a semantic link between web data with XML tag *zip-code* and a data of *zip* column from a database, that they both actually mean the same. Ontologies will allow machines to follow semantic links, and facilitate efficient information retrieval by the integration of data from many different data sources.

Meanwhile, the need has increased for shared semantics. According to the article "The Semantic Web Revisited" which is published by the IEEE Computer Society (2006) [203], there are a number of research areas that also drive the development of ontologies together

[1] Meta-data is data about data.

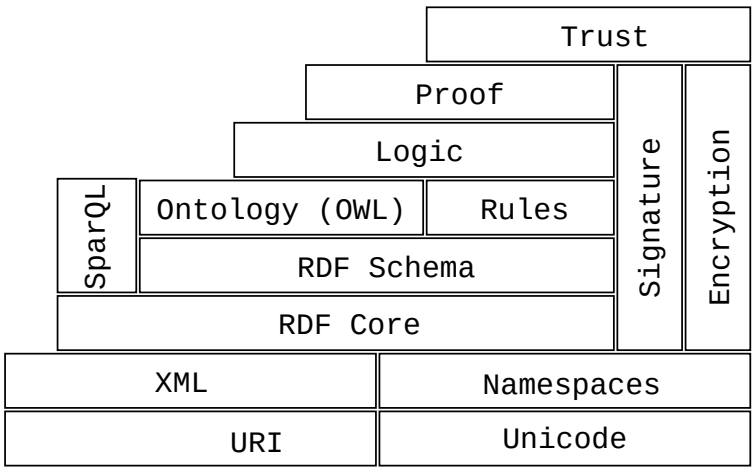

Figure 1.1: The layer cake of data representation standards for the Semantic Web

with the Semantic Web. One major driver is *e-science*—computationally intensive science that is carried out in highly distributed network environments, or science that demands the integration of diverse and heterogeneous data sets originated from distinct communities of scientists in separate subfields. For example, environmental science is looking to integrate data from hydrology, climatology, ecology, and oceanography[2]. Scientists, researchers, and regulatory authorities in genomics, proteomics, clinical drug trials, and epidemiology all need a way to incorporate these components for data and information integration. This is being achieved in large part through the adoption of common conceptualizations referred to as *ontologies*.

A new interpretation of the Semantic Web, described in [203] is as follows:

> *"....The Semantic Web is a Web of actionable information—information derived from data through a semantic theory for interpreting symbols. The semantic theory provides an account of "meaning" in which the logical connection of terms establishes interoperability between systems."*
>
> *Tim Berners-Lee, Massachusetts Institute of Technology*
>
> *Nigel Shadbolt and Wendy Hall, University of Southampton*

To be consistent with the need for the Semantic Web, the Internet Engineering Task Force[3] and the World Wide Web Consortium (W3C)[4], have directed major efforts at specifying, developing, and deploying languages for sharing meaning. These languages provide a foundation for semantic interoperability. The Semantic Web will be built on the standard layers as shown in Figure 1.1.

Uniform Resource Identifiers (URIs) are the most fundamental component of the current Web, which provide the ability to uniquely identify web resources as well as links among the resources. Everything on the Semantic Web must have a URI. Associating a URI with a resource means that anyone can link to it, refer to it, or retrieve it [202].

Resource Description Framework (RDF) provides a simple but powerful triple-based representation language for URIs. An RDF triple consists of a subject, predicate, and

[2] `http://marinemetadata.org/examples/mmihostedwork/ontologieswork`
[3] `http://www.ietf.org/`
[4] `http://www.w3.org/`

object. The subject identifies what object the triple is describing. The predicate defines the relational property from the subject to the object. The object is the actual value. For example, in the statement "Lee initiates the Semantic Web", *Lee* is the subject, *initiates* is the predicate, and *the Semantic Web* is the object. *RDF Schema* is an extension of RDF, in order to provide a modeling language on top of RDF. It has provided a minimal ontology representation language that the research community has adopted fairly widely.

The *ontology* layer provides more meta-information, such as relationships between meta-data, the cardinality of the relationships, the transitivity of the relationships, etc. *Ontology Web Language* (OWL) is the current recommendation of W3C as ontology representation language. The core idea of OWL is to enable efficient representation of ontologies that are also amendable by decision procedures. *Rules* check an ontology to see whether it is logically consistent, or to determine whether a particular concept falls within the ontology. A range of automated reasoners are available[5]. Because it is difficult to specify a formalism that will capture all the knowledge, Rule Interchange Format (RIF)[6] is an attempt to support and interoperate across a variety of rule-based formats. RIF will address the plethora of rule-based formalisms: Horn clause logics, higher-order logics, production systems and so on.

SparQL is a W3C recommended query language for easy access to RDF triples. The *logic* layer enables the writing of reasoning rules. The *proof* layer executes the use of rules and evaluates, together with the logic layer, mechanisms for applications to decide whether to trust the given proof or not.

The Semantic Web has been creating many challenges, especially ontology development and management. Some people perceive ontologies as top-down, some what authoritarian constructs [99]. This perception might be related to the idea of developing a single consistent ontology of everything—like Cyc [39]. However, this is not flexible in the Semantic Web because of its dynamic nature in diverse data, knowledge, and users. Ontologies are attempts to more carefully define parts of the data world, and to allow mappings and interactions between data held in different data formats. The ontologies that will furnish the semantics for the Semantic Web, must be developed, managed, and endorsed by all committed practice communities and users. Thus, the Semantic Web allows the proliferation of ontologies. However, a consistent and seamless data ubiquity is expected through the ontologies. The following substantial research challenges has been considered for such data ubiquity on the Web [203].

- How do we align and map between ontologies that are independently created from different community groups and users?
- How do we effectively query a huge number of decentralized information repositories of varying scales?
- How do we construct a Semantic Web browser that effectively visualizes and navigates the huge connected RDF graph?
- How do we establish trust and provenance of the content?

Regarding the above challenges, ontology matching is still a critical need to accomplish the vision of the Semantic Web.

[5] `http://www.cs.man.ac.uk/ sattler/reasoners.html`
[6] `http://www.w3.org/2005/rules`

1.1.2 Smart Spaces

A smart space is a logical boundary for a rich area of computing and resources in a pervasive computing environment—an environment saturated with numerous devices embedded with heterogeneous computing and communication capabilities [96]. For example, an intelligent meeting room, home, office, university campus, software house, hotel, airport, and every kind of intelligent environment are called *smart spaces*.

A smart space needs to be aware of what users are trying to do, in order to offer appropriate assistance. Intelligent tools and applications need to be built on top of these components. MIT's Oxygen project seeks to construct the prototype of such smart spaces [107]. The project Oxygen proposed Agent-based Intelligent Workspaces (also called Intelligent Environments) [150] and defined them as follows.

> *An Intelligent Environment (IE) is a physical space that is perceptually enabled, that is capable of natural human interactions, and that provides both proactive and reactive services to a community of users.*

Stanford Interactive Workspaces [26] explores new possibilities for people to work in technology-rich meeting spaces that consist of computing and interacting devices on various scales. It has contributed with a great effort in solving problems in switching displays between different sizes of screens. iRoom is the prototype of their interactive workspaces.

Microsoft Easyliving Project [23] explores the architecture and technologies for intelligent environments that contain many different types of devices and support rich interactions with users. Easyliving aggregates diverse devices into a coherent user experience.

A smart space needs to control a wide range of physical devices: lights, audio/video equipment, telephone, mobiles, handheld computers, etc. Additionally, a smart space needs to control a significant number of software components: messaging systems, personal file databases, schedule-keeping agents, and so on. A control system for a smart space needs to provide a standard mechanism that manages components, enables communication between them, facilitates interaction between users and the environment, and protects security and privacy. Mobile components need a mechanism for discovering and effectively using their surroundings, while stable environments need to be able to incorporate those mobile devices. Lastly, these control and coordination mechanisms need to be extensible to work over different smart spaces. It is believed that users and devices are not stationary in one smart space. They move across different spaces.

Satyanarayanan [120] identified three types of challenges in the design and implementation problems raised for the smart spaces of a pervasive computing environment.

- The first type is related to hardware and system designs, such as high-level energy management, client thickness, balancing proactivity and transparency.
- The second type is security issues like privacy and trust.
- The last type is about generating and exchanging information in pervasive computing paradigm, like user's intent (or preferences) and context information (state of environment), to achieve cyber foraging and adaptation strategy.

For the third challenge, ontologies are employed to provide a common representation of information and background knowledge across a variety of computing devices and

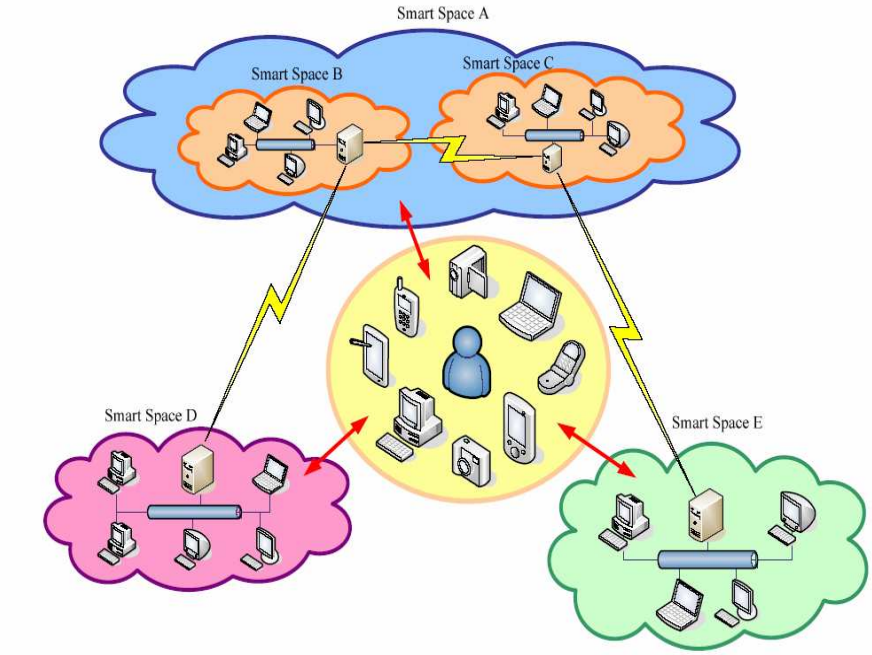

Figure 1.2: An architecture of pervasive computing environment with smart spaces [96]

information systems. Effective internal communications refer to a standard ontology for services and resources lookup. However, effective external communications require bridging knowledge across different ontologies of different smart spaces.

Most smart space research focuses on the design of infrastructure, and explores new technologies, migrating tasks to more powerful devices, and detecting the environment states using sensors. Kong's research work was different from those in that and it was considered to provide interoperability between different smart spaces through online ontology matching [96]. The architecture of a pervasive computing environment with multiple smart spaces proposed by Kong, is described in Figure 1.2.

In the architecture, three types of ontologies have been classified. *Domain ontology* is the ontology about the smart space, for example, the environment context, resources, activities done, and people who are present at the smart space. Smart space monitors are used to store the domain ontology. Assume that there is a single domain ontology in each smart space. *Application ontologies* store the concepts used in applications like device configuration, application parameters, and service descriptions. A smart space can have various application ontologies which all are stored inside application servers and managed by application vendors. At run-time, application ontologies are cached in smart space monitors. *User ontology* consists of the user's knowledge such as user identity, social and mental status, and all sorts of user information and preferences, that all express the characteristics and behaviors of a user. Complete user ontologies may be resident somewhere in the Internet or in their high-storage machine. However, most frequently used concepts (called partial user ontology) will be stored in users' handheld mobile

devices like smart phones[7] or PDA phones[8]. Each user has his/her own user ontology. Kong also presented an interesting daily life scenario in the smart spaces, as follows:

> *"...Alice is traveling to another country on the plane. She does not understand the language spoken in the destination country. As usual, she brings her smart phone that stores her personal details such as her identity and daily schedule. Alice gets off the plane and arrives at the immigration building located at the airport. Once she steps into the building, the sensors immediately inform the smart space monitor about the arrival of Alice and the smart space monitor forwards the information to the immigration department computers. Smart space monitor forwards a travel declaration form to Alice's smart spone when it gets response from immigration department computers. The smart phone askes Alice whether her personal details are allowed to be disclosed when it receives the form. Alice clicks 'OK'. As the terminology used to represent the personal information in Alice's smart phone is different from those required by the immigration department, online ontology matching is performed. When mappings completes, Alice's smart phone automatically fills in the form. Besides, the smart phone retrieves Alice's schedule and fills in the name of the hotel where Alice is going to stay and the departure date. After that, the form is sent back to the immigration department. The immigration department computers collect the form and verify whether Alice can legally enter the country...."*

Kong presented other similar scenarios that need mapping between Alice's user ontology and smart phone application ontology to recognize Alice is hungry, mapping between Alice's user ontology and hotel's domain ontology to locate a fish ball machine, mapping between Alice's user ontology and hotel's application ontologies to access resources and devices in the hotel, and etc. In summary, it is obvious that ontology matching helps in bridging knowledge gaps between smart spaces.

1.2 Problem Definition

In order to tackle the need for sharing knowledge within and across organizational boundaries, the last decade has seen researchers both in academia and industry advocating the development of ontologies and the use of them. Ontolingua [207, 5], Protégé[9] [102, 206, 199, 103, 104, 129, 130, 132, 133, 89, 71, 72, 73, 74], WebODE[10] [86, 159, 87], OntoEdit [222, 223], OILEd[11] [184], and SWOOP [12, 13, 14, 15, 16], are some examples of ontology development tools. TOVE [111] and METHONTOLOGY [109, 9, 10, 110] are the

[7] A smart phone is any electronic handheld device that integrates the functionality of a mobile phone, personal digital assistant (PDA) or other information appliance. A key feature of a smartphone is that additional applications can be installed on the device.

[8] A PDA Phone is a combination of mobile phone (cellular phone) and personal digital assistant functionality in one device. It differs from a smartphone in that it has a touch screen and a stylus. Compared with a smartphone it usually has a larger screen, a more powerful microprocessor, more memory, etc. In short, it functions more like a computer in it's input/output of information.

[9] `http://protege.stanford.edu/`

[10] `http://webode,dia.fi.upm.es/`

[11] `http://oiled.man.ac.uk`

methodologies for building ontologies. Cycl [38, 39], XOL [163], RDF [161], OIL [79, 80], DAML+OIL [81], and OWL [106], are gradually invented as ontology languages.

Because of independent conceptualization of domain knowledge, and different application requirements in each domain, ontologies have proliferated, so that a trade off between interoperability and heterogeneity is faced. Heterogeneity is both a welcome and unwelcome feature because it improves the efficiency of applications on one hand, but it degrades interoperability on the other hand. In order to keep a balance between heterogeneity and interoperability, ontology matching has become a plausible solution in various tasks. Heterogeneity is generally distinguished in terms of syntactic heterogeneity and semantic heterogeneity. *Syntactic heterogeneity* is caused by using different ontology modeling paradigms (e.g., RDF-based model or Frame-based model) and different ontology languages (e.g. DAML or OWL), while *semantic heterogeneity* is created by conceptualization divergence in describing the semantics of ontological classes.

Research on resolving syntactic heterogeneity has been undertaken by many researchers so far [185, 66, 67]. For instance, Gruber [210] describes a mechanism for defining ontologies that are portable over representation systems. Descriptions written in a standard format of predicate calculus are translated by a system called Ontolingua into specialized representations, including frame-based languages as well as relational languages. Today, most ontology editors allow the utilities of 'import' and 'export' for syntax translation. The deep and unsolved problems are thus with the semantic issue. Therefore, I focus on semantic heterogeneity between ontologies.

Recent mapping methods, methodologies and tools such as IF-Map [220], ONIONS [164, 6, 63], FCA-Merge [64], PROMPT [128, 131, 134, 135, 136, 137], MAFRA [17, 18], GLUE [4], NOM [108], QOM [113], OMEN [165], COMA [68], and COMA++ [69], have attempted various semi-automatic and automatic methods for the discovery of correspondences, using a common reference ontology, instance analysis, schema analysis, referring to a shared thesaurus like WordNet[12] and corpuses, analyzing structural information, applying statistic and probabilistic models, and other machine learning techniques. Although many efforts in ontology mapping have already been contributed, they have different focuses, assumptions, and limitations. A common point among existing methods is that possible correspondences between two ontologies are determined by the similarity of entity names; this is known as *name-based matching*. In order to decide semantic correspondences between concepts, the methods need to analyze the similarities between all related properties and instances; this is known as *content-based matching*. In the case of several forms of heterogeneities, content-based matching becomes complex, and user's approval or expert-interaction needs to verify mapping results.

In my research, I focus on two issues. The first issue is that using name-based matching to determine possible correspondences is risky. In practice, two concepts with the same name may have different semantics, or two differently-naming concepts may have the same semantics. Because the name of a concept cannot express the precise semantics of the concept, the chance of correspondence between two terminologically quite different concepts is very less or not obtainable through name-based matching. The second issue is how to reduce complexity, concerning wide-scale semantic heterogeneity in content-based matching.

As discussed in the motivation section, a matching method without (or with very little) expert-interaction is helpful especially in the applications and services of the Semantic

[12] `http://wordnet.princeton.edu/`

Web and Smart Spaces. When matching is done between two large ontologies, efficiency becomes critical. The accuracy of matching results accelerates interoperability. Therefore, this research is considered as a theoretical framework that can improve the efficiency and accuracy of matching between semantically heterogeneous, large ontologies, in a balanced manner particularly concerning the above two issues.

1.3 Objectives

The aim of this thesis is *how to deal with wide-scale semantic heterogeneity in matching between large ontologies*. This aim is further divided into the following intermediate goals.

1. To propose an intuitive idea of bridging over semantic heterogeneities between ontologies based on some philosophical foundations. I call this semantic enrichment in ontologies.

2. To present a formal modeling framework of semantically-enriched ontologies.

3. To provide an implementation framework of semantically-enriched ontologies.

4. To supply a conceptual analysis system that gruntees for well-structured and consistent ontologies.

5. To present a method of enrichment-based ontology matching for the effective discovery of semantic correspondences between heterogeneous ontologies.

6. To provide an evaluation of enrichment-based matching method.

7. To analyze the applicability of enrichment framework to existing ontologies and real applications.

1.4 Approach and Scope

As noted by McGuinness [42], an explicit description of the semantics of domain terms would be helpful in determining whether two concepts are similar or not. For ontology matching, my underlying assumption is *the more explicit semantics is specified in ontologies, the feasibility of matching will be greater*. Hence, an important step in handling semantic heterogeneity should be the attempt to enrich the semantics of concepts in ontologies, as it is well understood that the richer knowledge the ontologies possess, the higher probability of accurate and efficient mappings will be derived.

The semantic enrichment techniques are based on different theories and a variety of knowledge sources, linguistic knowledge, fuzzy terminology, and intensional or extensional knowledge [187]. An ontology mostly specifies the semantics of concepts using intensional knowledge which consists of the properties of concepts and relationships between them. Extensional knowledge is used to populate ontologies by interpreting each concept with a set of individuals from a universe of discourse[13]. The linguistic knowledge, especially shared thesauri, is used to assist in determining correspondences between domain terms.

[13]The term 'universe of discourse' generally refers to the entire set of terms used in a specific discourse. In model-theoretical semantics, it refers to the set of individual entities that a model is based on.

However, Mitra and Wiederhold claim that full automation for a mapping using such linguistic knowledge, is not feasible due to the inadequency of today's natural language processing technology [164]. There are still different opinions on whether it is intension or extension that best decides the exact context of a concept [218]. It is obvious that the semantics of similar concepts described by either intensional knowledge or extensional knowledge, in two different ontologies, is still heterogeneous especially in open and distributed systems.

According to METHONTOLOGY [110], a process of ontology development involves five activities as follows:

- The *specification* activity states purpose, scope of domain knowledge, and intended user, for an ontology.
- The *conceptualization* activity converts an informally perceived view of a domain into a conceptual model represented in the form of graphs and tables.
- The *formalization* activity transforms a conceptual model into a formal computable model using logic languages.
- The *implementation* activity codes computable models in the syntax of ontology languages, via ontology editors.
- The *maintenance* activity corrects and updates ontologies and their models, if needed.

Among these activities, semantic heterogeneity between independently designed ontologies is progressively appeared by the conceptualization step. Thus, we may also call it conceptual heterogeneity of ontologies.

In order to improve the accuracy and automation of mapping processes, it is necessary that ontologies be well conceptualized with adequate semantics. For this purpose, I propose an enrichment approach that is based on the classification of concepts/classes using some philosophical notions. In my approach, every fundamental domain concept is treated as a sort—an entity type that carries a criteria for determining the individuation, persistence, and identity of its instances—regarding every individual (or instance) defined in a universe of discourse is countable and identifiable. This means there is no ontology without sorts which are the most fundamental classes to answer what an individual is. A detailed discussion can be seen in Section 4.2.1. In the philosophical literature [147, 61], ontological concepts can be classified into four disjoint sort categories: *type*, *quasi-type*, *role*, and *phase*. I redefine these sort categories using three philosophical notions: *identity*, *existential rigidity*, and *external dependency*. Also, the notions are reformalized in a precise semantics by providing a First-order Quantified Modal Language. Then, a model of semantically-enriched ontologies (called EnOntoModel) is created by using this classification scheme, and a formal way of embedding concept-level properties into ontologies is developed. This is my approach of *semantic enrichment*. Concept-level properties are the properties which describe the classification knowledge for concepts only and not for individuals, that is individuals cannot employ these properties. Individual-level properties (or intensional knowledge) of a concept describes the semantics of the concept, however they are used as the specification to define data of individuals instantiated to the concept. Thus, concept-level properties are different from individual-level properties and they are

also called the *meta-knowledge* of concepts. Though the description of a concept can be slightly different according to domain experts, the meta-knowledge—particularly sort category—of the concept is not distinctive for the same semantics. The innovation behind EnOntoModel is to supply an identifiable link between two heterogeneous descriptions of a concept, regarding that if two concepts are semantically equivalent, then they must be classified within the same sort category. For the usability of EnOntoModel, a sortal meta-class ontology is implemented as an open source interface not only for enrichment process but also for conceptual analysis of enriched ontologies. By the aim of the thesis, a matching method between enriched ontologies is designed.

A novel idea of EnOntoModel-based Ontology Matching (EOM) method is that direct concept matching is driven between the same categories of sorts instead of exhaustive search among all sorts, because domain concepts are systematically classified into disjoint sort categories. Moreover, it is examined that EnOntoModel can support not only determining the scope of possible correspondences, but also providing the most relevant properties which can certainly indicate a correspondence between two similar concepts. In this thesis, an assumption is made for ontological concepts and individuals following by a postulate of Guizzardi [61], that is, *if a concept is interpreted as an abstract description of individuals that are countable and identifiable, then every individual should be the instance of at least one sort which supplies an Identity Condition (IC)*—a property of a sort that provides a unique IC value for each individual of the sort such as *supplying fingerprint* is the IC for sort `Person`. By this assumption, semantic correspondences between two concepts are possibly to be detected mainly by finding similarity between their ICs instead of all available properties and relationships of the concepts. This idea is applied in EOM in order to support a content-based matching with less complexity.

This research brings together techniques in philosophy, conceptualization, formal ontologies, mathematical logic, and knowledge representation. It is constituted with two major phases: (1) semantic enrichment phase and (2) enrichment-based matching phase. In phase (1), a philosophy-based semantic enrichment approach is proposed. It is composed of three major contributions:

1. modeling semantically enriched ontologies in a formal way by providing a First-order Quantified Modal Language $\mathcal{L}^E$,

2. implementing a sortal meta-class ontology as an open source interface for the usability of EnOntoModel, and

3. a practical framework of enrichment and conceptual analysis.

In phase (2), the design and evaluation of enrichment-based matching method are presented. It consists of four parts:

1. a content-based matching method between enriched-ontologies for the purpose of intelligent information retrieval or exchange,

2. an evaluation of matching method in terms of mathematical complexity,

3. the implementation of EOM in Java using Jena OWL API and Protégé OWL API, and an experiment with two real data sets, and

4. a comparison with some related mapping methods as well as with OntoClean.

1.5 Thesis Outline

In this chapter, the motivation of the work, the main problem tackled, the objectives, and the approach and scope of solving the problem, are described. The remainder of the thesis is organized as follows.

Chapter 2 presents the fundamental concepts of ontological engineering in order to introduce the current technical foundation of ontologies.

Chapter 3 focuses on heterogeneities in ontologies and presents my classification of semantic heterogeneities. Moreover, a survey work on existing ontology mapping tools and methods is well provided.

Chapter 4 is about semantic enrichment in ontologies. This is one of the main chapters of the thesis. It consists of five sections. In Section 1, a First-order Quantified Modal Language $\mathcal{L}^E$ is constructed in order to express the precise meaning of each referred philosophical notion based on a Kripke model that concerns the issue of actual existence and varying domains among possible worlds. In Section 2, I present some philosophical foundations, in particular identity, existential rigidity, and external dependency, for ontological conceptualization. Section 3 is a modeling framework of semantically-enriched ontologies, where ontological concepts are classified into four sort categories based on the philosophical notions. Then, a model for semantically-enriched ontologies (called EnOntoModel) is proposed. In the model, the semantics of concepts are well defined with not only a set of individual-level properties, but also concept-level properties. Thus, it is called enriched-model. Section 4 is an implementation framework of a sortal meta-class ontology that supplies as an open source interface for the enrichment process. Five subsumption constraints are authored in Protégé Axiom Language (PAL) and embedded in the meta-class ontology for conceptual analysis of enriched ontologies. In the final section, Section 5, the development of semantically-enriched ontologies is demonstrated in Protégé OWL API together with an analysis of conceptual consistency in enriched ontologies via "PAL Constraints" plug-in.

Chapter 5 presents EnOntoModel-based ontology matching method. It consists of matching architecture, method, algorithm, implementation, and its evaluation in terms of mathematical complexity for efficiency, as well as in terms of precision and recall for effectiveness. As for related work, I discuss my work concerning ontology matching and conceptual analysis.

Chapter 6 concludes the thesis with a summary of my contributions, advantages and limitations, a brief history of my research progress, and a list of issues and perspectives sketched for the future research.

Chapter 2

Ontological Engineering

Today, ontologies are widely used in Computer Science, in applications related to knowledge engineering and management, intelligent information retrieval and integration, and in new emerging fields like the Semantic Web and Smart Spaces.

> **Ontological Engineering** *refers to the set of activities that concern the ontology development process (ontology life cycle), and the methodologies, tools, and languages for building ontologies [11].*

This chapter introduces the theoretical and technical foundation of ontological engineering.

2.1 Basic Ontology Concepts

The word *ontology* comes from the Greek *ontos* for being and *logos* for treatise [11]. In philosophy [169], ontology is *a study of being or existence*. The concept of ontology is generally thought to have originated in early Greece and occupied Plato and Aristotle. The oldest extant record of the word itself is the Latin form *ontologia* which appeared in 1606. The term *Ontology* was coined in 1613 by Rudolf Göckel—a German scholastic philosopher who is also known as Rudolf Goclenius. The first occurrence in English of "ontology" as recorded by the OED appears in Baileys dictionary of 1721, which defines ontology as *an Account of being in the Abstract*. Ontology has some basic questions [169]:

- What is existence?
- Is existence a property?
- What constitutes the identity of an object?
- What features are the essential attributes of a given object?
- Can one give an account of what it means to say that a physical object exists?
- When does an object go out of existence, as opposed to merely changing?

These questions have being debated by a number of philosophers and logicians. Also, philosophers have struggled with deep problems of existence, such as God, life and death, or whether a statue and the marble from which it is made are the same entity.

2.1.1 What is an Ontology?

Guarino and Giaretta [141] proposed to use the words 'Ontology' (with capital 'o') and 'ontology' (with small 'o') to refer to the philosophical and knowledge engineering senses respectively. There are many definitions about what an ontology, is [169]. In this section, a summary of those definitions changed and evolved over the years is provided, and I conclude with my own.

A definition given by Neches and Colleagues [181] is as follows:

> *An ontology defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary. (Neches and colleagues, 1991)*

Note that, according to Neches's definition, an ontology includes not only the terms but also the rules to infer new knowledge from them. A few years later, Gruber [208] defined as:

> *An ontology is an explicit specification of a conceptualization. (Gruber, 1993)*

This definition became the most quoted one in the ontology community. Borst [215] modified Gruber's definition slightly as:

> *Ontologies are defined as a formal specification of a shared conceptualization. (Borst, 1997).*

Gruber's and Borst's definitions have been merged and explained by Studer and colleagues [182] as follows:

> *"An ontology is a formal, explicit specification of a shared conceptualization". Conceptualization* refers to an abstract model of some phenomenom in the world by having identified the relevant concepts of that phenomenom. *Explicit* means that the types of concepts used, and the constraints in their use are explicitly defined. *Formal* refers to the fact that the ontology should be machine-readable. *Shared* reflects the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by a group (Studer and colleagues, 1998).

Regarding an ontology to be used for building several knowledge bases, Swartout and colleagues [30] defined an ontology as:

> *An ontology is a hierarchically structured set of terms for describing a domain that can be used as a skeletal foundation for a knowledge base (Swartout and colleagues, 1997).*

In 1998, Guarino and Giaretta [142] proposed to consider an ontology as:

> *A logical theory which gives an explicit, partial account of a conceptualization (Guarino and Giaretta, 1998).*

Since ontologies are widely used for different purposes in different communities, Uschold and Jasper [125] defined ontology in a different way:

An ontology may take a variety of forms, but it will necessarily include a vocabulary of terms and some specification of their meaning. This includes definitions and an indication of how concepts are inter-related which collectively impose a structure on the domain and constrain the possible interpretations of terms (Uschold and Jasper, 1999).

Even though the definitions are slightly different from each other, according to different development experience in different domains and communities, "*a conceptualization of consensual knowledge in a domain*" is commom among these definitions. By the demands of semantic heterogeneity, my own definition is given below.

An ontology is an explicit and semantically-enriched specification of a shared conceptualization in a domain, that is formalized in terms of a logic system and is coded in a machine readable format, so that efficient interoperability with seamless reasoning and data ubiquity, is performed in an intelligent manner among information systems.

2.1.2 Which are the Main Components of an Ontology?

The answer depends on the kind of knowledge modeling paradigm used for an ontology. Moreover, the degree of formality and granularity of the knowledge that is conceptualized for an ontology, distinguishes the components of one ontology from another ontology.

Ontologies are generally classified as *lightweight* and *heavyweight* ontologies. Lightweight ontologies include concepts, concept taxonomies, relationships between concepts, and properties that describe concepts. On the other hand, heavyweight ontologies add axioms and constraints to lightweight ontologies in order to clarify the intended meaning of each concept.

Moreover, ontologies can be *highly informal* if they are expressed in natural language; *semi-informal* if expressed in a restricted and structured form of natural language; *semi-formal* if expressed in an artifical and fomally defined language (i.e., RDF, OWL); and *rigorously formal* if they provide precisely defined terms with formal semantics, theorems and proofs of soundness and completeness[1] [124, 162]. According to the definition of Studer and colleagues, a highly informal ontology would not be an ontology because it is not machine readable.

Gruber [208] proposed a model of ontologies using frames[2] and first order logic[3]. He identified five kinds of components: classes, relations, functions, formal axioms, and instances, as follows:

- *Classes* represent both abstract concepts (`Belief`, `Feeling`, `Action`, etc.) and concrete concepts (`Person`, `Book`, `Computer`, `Car`, `Wine`, `Apple`, etc.).

[1] Inspired by classical first order logic terminology [70, 95], we say that an ontology is sound if and only if it does not allow deducing invalid conclusions. We say that an ontology is complete if and only if it allows deducing all the possible valid conclusions starting from the ontology vocabulary and applying deduction rules permitted.

[2] A frame is a collection of certain slots which are the relations between constant values. Alternatively, a frame can be considered just a convenient way to represent a set of predicates applied to constant symbols.

[3] First order logic is a system of mathematical logic which extended propositional logic using quantifiers: $\forall$ and $\exists$. It is also known as First Order Predicate calculus and it allows variables ranging over different sorts of individuals given in a universe of discourse.

- *Relations* represent a type of n-ary association between concepts. However, ontologies usually contain binary relations between two arguments called domain and range. Subsumption or sub-class relationships in class taxonomies are examples of binary relations. For examples, `SubclassOf(Student, Person)`, `SubclassOf(RedWine, Wine)`. Binary relations are also used to express concept attributes (aka slots). Attributes are distinguished from relations in that their range is a datatype such as `String`, `Integer`, `Date`, `Year`. As an example, `VintageOf(Wine, Year)`, `NameOf(Person, String)`.

- *Functions* are a special case of relations in which the $n+1$-th element of the relation is retrieved for the n preceding elements. For example, if Z is the set of integers, N is the set of natural numbers (except for zero), and Q is the set of rational numbers, then division is a binary function from Z and N to Q.

- *Formal axioms* serve to model sentences that are always true, and they are used to verify the consistency of the ontology itself or to infer new knowledge. Every class is a sub-class of itself, or sub-class relation does not allow symmetric relationship between two classes, or there is no instance which belongs to two disjoint classes, are some examples of formal axioms.

- *Instances* are used to represent individuals of a class. For example, "Louis Latour, France, Burgundy, Chambolle Musigny, red wine, Pinot Noir, 750ml, 1993" is an instance of RedWine.

An alternative candidate for modeling ontologies is using description logic based systems. Description Logic (DL) [52] is a logical formalism whose early implemented languages and systems are: KL-ONE [175], Krypton [174], Classic [1], LOOM [178], and Kris [51]. The theory of DL is divided into two parts: the T-Box and the A-Box. T-Box contains terminological knowledge together with the properties of concepts. The A-Box contains assertional knowledge which is specific to the individuals of a domain. DL system allows the representation of ontologies with three kinds of components: concepts, roles, and individuals. Basically,

- *Concepts* have the same usage like the classes of a frame paradigm.

- *Roles* describes the binary relations between concepts, hence they allow the description of the properties of concepts.

- *Individuals* represent instances of concepts.

The T-Box contains the definitions of concepts together with roles, while the A-Box contains the definitions of individuals according to the T-Box. Both concepts and roles allow hierarchies with subConcept and subRole relationships respectively. In DL, the term *terminological axioms* are used to make statements about how concepts or roles are related to each other, such as subsumption or equality relationships between concepts.

In summary, the major components of an ontology consist of classes (or concepts), properties of both relations and attributes (or roles), individuals (or instances), and axioms (formal axioms and terminological axioms).

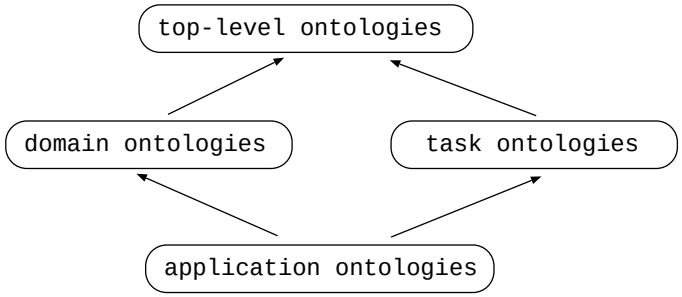

Figure 2.1: The ontology classification by Guarino

2.1.3 Categorization of Ontologies and their Uses

Ontologies are generally categorized according to abstraction level and subject of conceptualization.

The classification of ontologies by Guarino [142] is presented in Figure 2.1. **Top-level ontologies** or **upper-level ontologies** describe very general concepts and provide general notions under which the top-most concepts of other ontologies should be linked. Even though a top-level ontology can serve as a root of other ontologies, the existing several heterogeneous top-level ontologies which are shown in Figure 2.2 (which is also described in [11]) prove that a single global ontology cannot fulfill all requirements of enormous ontology-based applications.

Domain ontologies provide a set of concepts, their relationships, and principles governing their usage, within a single domain. For example, *Universal Standard Products and Services Classification* (UNSPSC)[4], *Gene Ontology*[5], *Education Ontology*[6], *BreastCancerOntology*[7], and *Wine Ontology*[8].

Task ontologies describe the vocabulary related to a generic task or activity like diagnosing, scheduling, selling, communication, etc., by specializing the concepts of top-level ontologies. Task ontologies may include more than one domain. Some examples of task ontologies are *OKAR*[9]-Ontology, *COBRA-ONT Action Ontology*[10], and *FIPA Agent Communication Ontology*[11].

Domain-task ontologies are task ontologies in a given domain, but not across domains. Moreover, they are application independent. Some examples of domain-task ontologies are (a) the *Enterprise Ontology*[12] developed in the enterprise project by the Artificial Intelligence Applications Institute at the University of Edinburgh with its partner: IBM, Lloyd's Register, Logica UK Limitted, and Univeler, (b) *Biological Processes Ontology*[13] developed for the Gene Ontology project, and (c) *ATO-98 Message Set On-*

[4] http://www.cs.vu.nl/ mcaklein/unspsc/
[5] http://protege.stanford.edu/ontologies/go/gopage.html
[6] http://education.state.mn.us/mde/index.html
[7] http://acl.icnet.uk/ mw/
[8] http://ontolingua.stanford.edu/doc/chimaera/ontologies/wines.daml
[9] http://jp.fujitsu.com/group/labs/en/
[10] http://daml.umbc.edu/ontologies/cobra/0.4/action.owl
[11] http://taga.umbc.edu/ontologies/fipaowl
[12] http://www.aiai.ed.ac.uk/project/enterprise/ontology.html
[13] http://mis.hevra.haifa.ac.il/ morpeleg/NewProcessModel/Malaria_PN_Example_Files.html

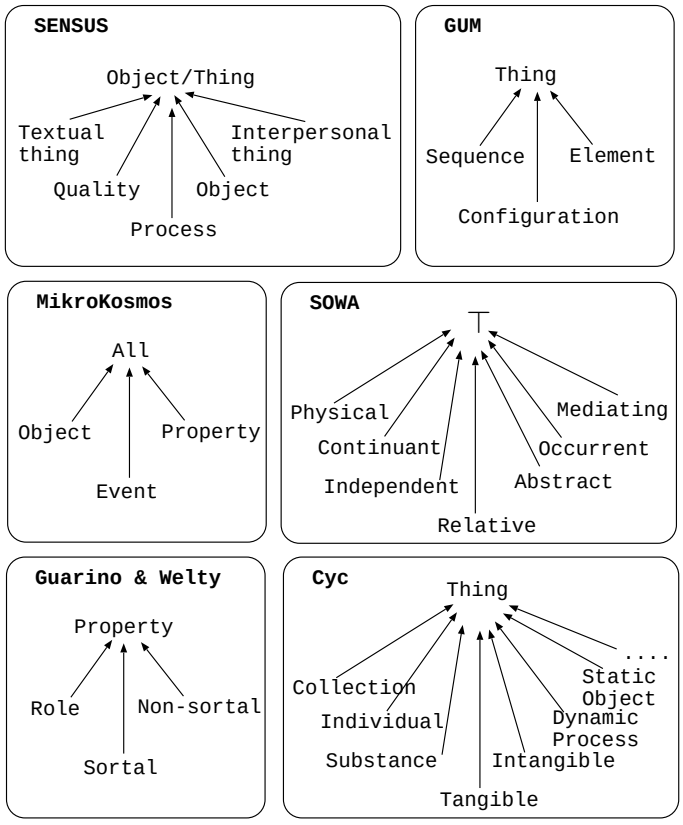

Figure 2.2: Examples of top-level ontologies

tology[14] developed for the Joint Battlespace Infosphere (JBI) project of the Air Force Research Laboratory.

Application ontologies are application-dependent domain and task ontologies, for instance, *Soccer Match Ontology*[15], *United States Postal Service Zone Improvement Program (ZIP) Code Ontology*[16], and *Travel Itinerary Ontology*[17].

Meta-ontologies are ontologies which capture the representation primitives used to formalize knowledge under a given knowledge representation paradigm such as OKBC, RDF(S), OIL, OWL, Frames, UML, etc.

Mizoguchi and colleagues [180] proposed the following four kinds of ontologies.

- *Content* ontologies for reusing knowledge. These ontologies include three subcategories: *task* ontologies, *domain* ontologies, and *general* ontologies.
- *Communication* (or Tell & Ask) ontologies are the ones for sharing knowledge.
- *Indexing* ontologies are the ones for knowledge retrieval.
- *Meta-ontologies* are the ones for other ontologies to refer as a knowledge representation ontologies.

[14] http://reliant.teknowledge.com/DAML/ATO98MessageSet_Ontology.owl
[15] http://www.lgi2p.ema.fr/ ranwezs/ontologies/soccerV2.0.daml
[16] http://www.daml.org/2001/10/html/zipcode-ont
[17] http://www.daml.org/2001/06/itinerary/itinerary-ont

The basic notions of Mizoguchi's general ontologies is similar to Guarino's top-level ontologies. Mizoguchi's communication ontologies are included in task ontologies of Guarino.

Additional examples can be seen in ontology libraries like OWL ontology library[18], DAML ontology library[19], and SchemaWeb ontology library[20]. This research concerns the semantic enrichment of domain ontologies under Guarino & Welty's top-level ontology expressed in Figure 2.2.

2.2 Methodologies for Building Ontologies

The goal of this section is to present some typical methodologies[21] used to build ontologies.

Gruber [209] identified five general principles in designing ontologies, as follows:

- *Clarity*: Concept definitions in ontologies should be objective, clearly stated in terms of formal axioms, and well-documented in natural language.
- *Coherence*: This concerns consistent reasoning. If a sentence that can be inferred from the axioms contradicts a given definition or example, then it is incoherent.
- *Extendibility*: The existing concept definitions should be able to define new concepts, in a way that does not require revision.
- *Minimal encoding bias*: The conceptualization should be specified at the knowledge level without depending on a particular symbol-level encoding.
- *Minimal ontological commitment*: Since ontological commitments[22] are defined as agreements to use shared vocabulary in a coherent and consistent manner.

The above principles should be employed in the ontology development process. There are a number of well-known methodologies to develop ontologies. Two of them are Grüninger and Fox's TOVE methodology and METHONTOLOGY methodology.

2.2.1 TOVE Methodology

Based on the experience of the TOVE project[23] in the enterprise domain, which was developed at the University of Toronto, Grüninger and Fox [111] set out to design a methodology for creating ontologies in several categories as shown in Figure 2.3. The methodology is composed of six steps as depicted in Figure 2.4. First, a set of motivating scenarios are defined in order to identify intuitively possible applications and solutions. Second, a set of informal competency questions that the ontology must answer in order to support the motivating scenarios, are defined. Third, the formal terminology of the ontology—objects, attributes, and relations—are defined in terms of FOL functions and

[18] `http://protege.stanford.edu/plugins/owl/owl-library/`

[19] `http://www.daml.org/ontologies/`

[20] `http://schemaweb.info`

[21] A methodology is a comprehensive, integrated series of techniques or methods creating a general system theory of how a class of though intensive work ought be performed.

[22] According to Guarino [139], ontological commitments offer connections between ontology vocabulary and the meaning of the terms of vocabulary.

[23] `http://www.eil.utoronto.ca/enterprise-modelling/entmethod/index.html`

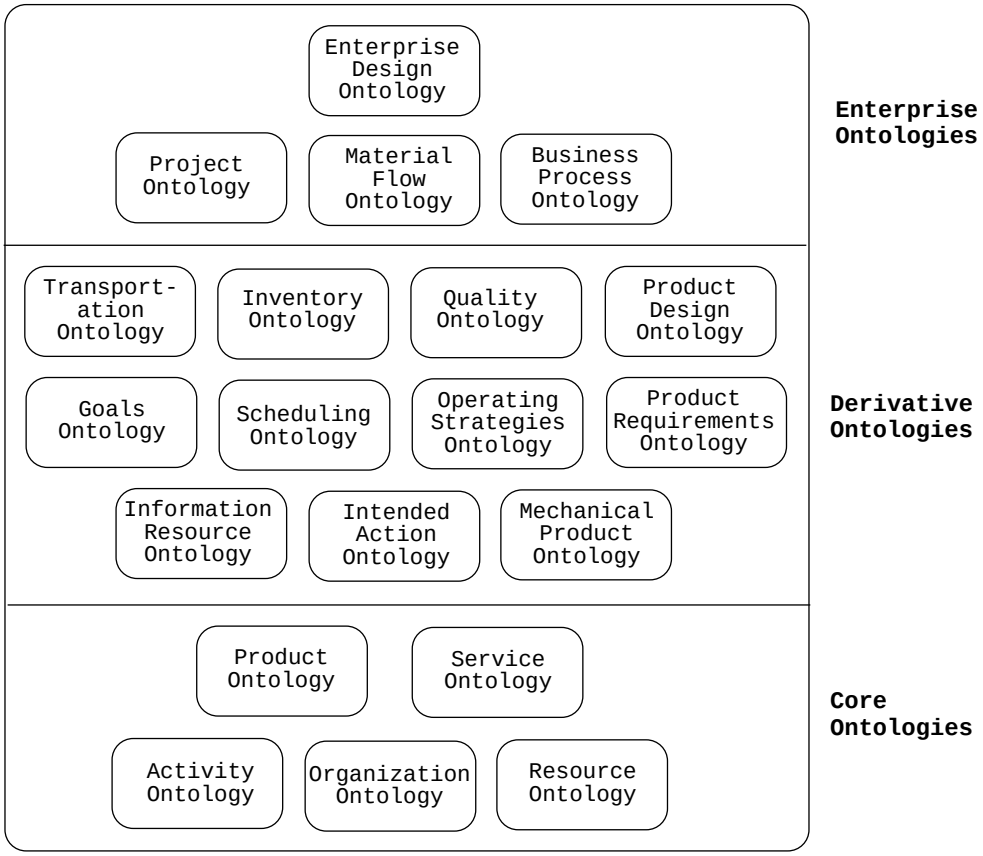

Figure 2.3: TOVE Ontologies excerpted from [111]

predicates. Fourth, the competency questions are formally redefined as an entailment of consistency problems with respect to the axioms in the ontology. Fifth, the formal axioms are defined using FOL. Finally, completeness theorems—conditions (or constraints) under which the solutions to the questions are complete–are constructed.

Grüninger and Fox's methodology is inspired by the development of knowledge-based systems using FOL. This is a very formal FOL-based methodology that takes advantage of the robustness of classical logic, and can be used as a guide to transform scenarios in computable models. The unique contribution of this work is the introduction of compentency questions as a basis of defining the scope of an ontology.

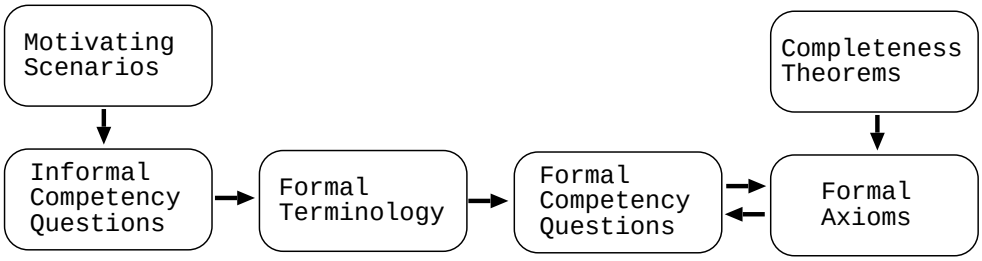

Figure 2.4: The major development processes of TOVE Methodology

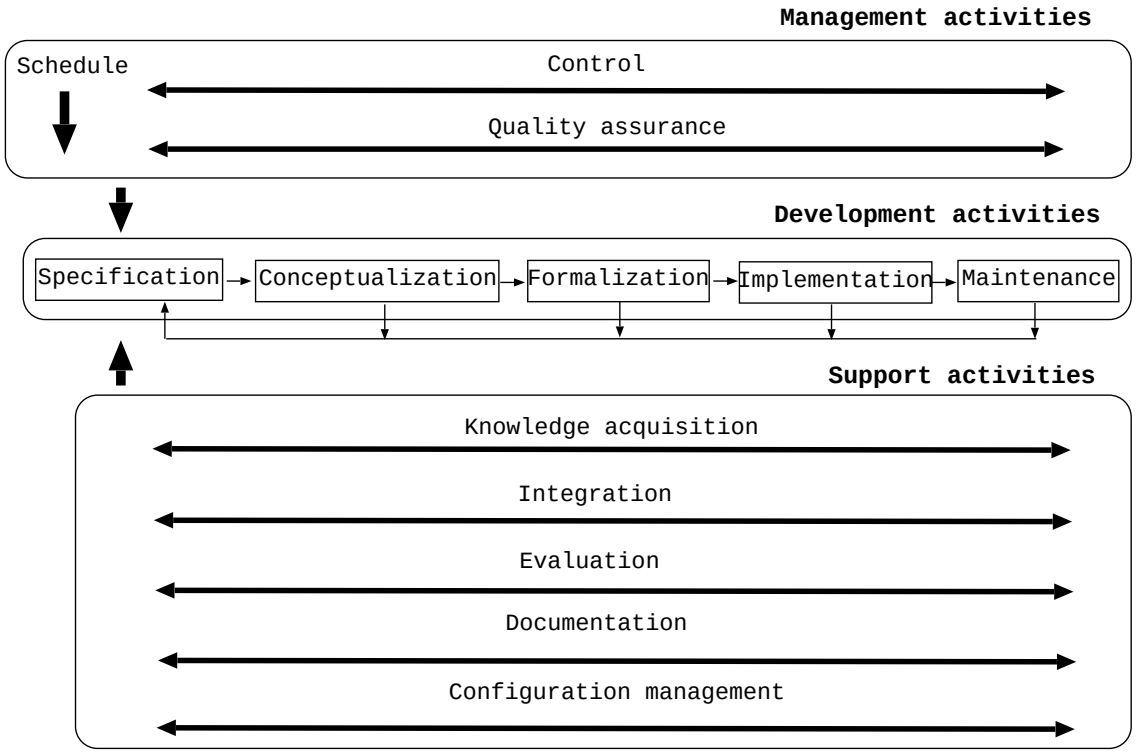

Figure 2.5: The ontology development life cycle of METHONTOLOGY Methodology

2.2.2 METHONTOLOGY

METHONTOLOGY [109, 9, 10, 110] is a methodology created by the ontology group of Universidad Politécnica de Madrid. It includes some main activities identified in the software development process [78] and knowledge engineering activities [36]. The activities are divided into three layers: 1) management, 2) development-oriented, and 3) support, as shown in Figure 2.5.

Each major activity of the development process is explained as follows:

- The *specification* activity states purpose, scope of domain knowledge, and intended user, for an ontology.
- The *conceptualization* activity converts an informally perceived view of a domain into a conceptual model represented in the form of graphs and tables.
- The *formalization* activity transforms a conceptual model into a formal computable model using logic languages.
- The *implementation* activity codes computable models in the syntax of ontology languages, via ontology editors.
- The *maintenance* activity corrects and updates ontologies and their models, if needed.

Figure 2.6 illustrates a set of step-by-step tasks performed in conceptualization activity. Each step emphasizes specific ontology components—concepts, attributes, relations, constants, formal axioms, rules, and instances. ODE [105] and WebODE [159] were build

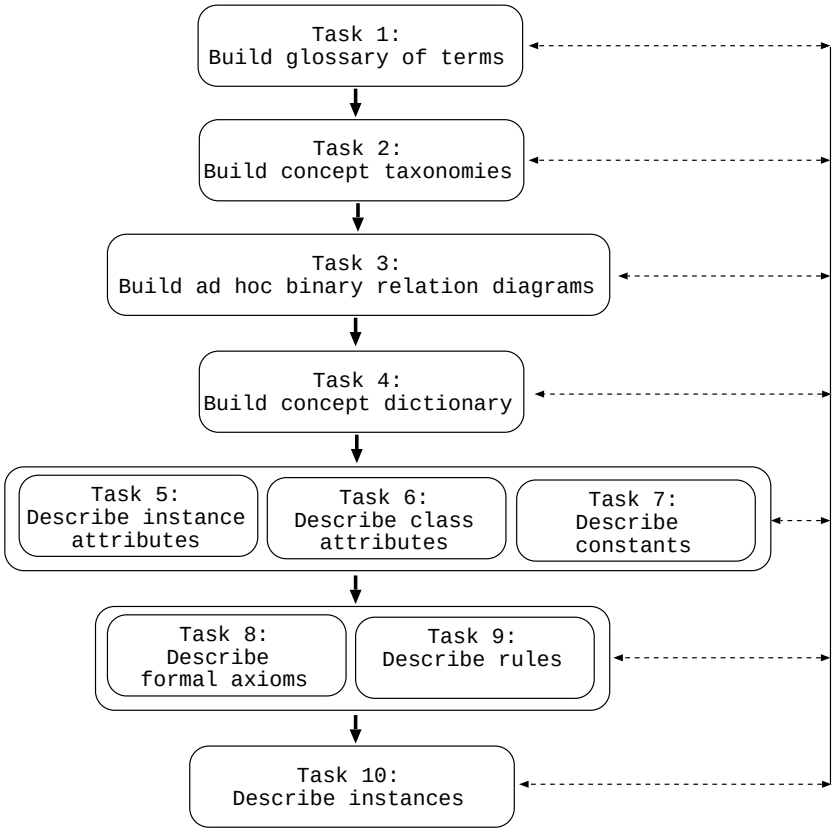

Figure 2.6: Tasks of conceptualization activity according to METHONTOLOGY

to provide technological support for METHONTOLOGY. When tools like WebODE ontology editor are used, the conceptualization model can be automatically implemented into several ontology languages using appropriate translators. Consequently, formalization is not a mandatory activity in METHONTOLOGY. METHONTOLOGY has been applied in the building of many ontologies [110] such as a chemical ontology, an ontology of Monatomic Ions, Environmental Pollutant ontologies, Silicate ontology, Reference ontology, and FIPA-Foundation for Intelligent Physical Agents[24] ontology.

2.3 Ontology Languages

This section briefly present the step-by-step evolution of ontology representation languages. Thereafter, RDF (Resource Description Framework) and OWL (Ontology Web Language) are introduced with their key characteristics. For a comparative survey of ontology languages, refer to [217, 112, 219, 11].

2.3.1 Evolution

At the begining of 1990s, a set of AI-based ontology languages was created. Basically, the Knowledge Representation (KR) paradigm underlying such ontology languages were based on First Order Logic (FOL), F-Logic, and Description Logic (DL).

[24] `http://www.fipa.org/specs/fipa00086/`

Cycl is a formal language whose syntax is derived from first-order predicate calculus. It was first developed in the Cyc project [38, 39] in the 1980s, which aimed at providing a general ontology (called Cyc ontology) for commonsense knowledge. **KIF** [119, 127] was created in 1992, and was designed as a knowledge interchange format based on FOL. Since ontologies are difficult to create directly in KIF, **Ontolingua** [207, 208, 5]—the ontology language supported by ontolingua server—was developed on top of it. It was released in 1992 by the Knowledge Systems Laboratory of Stanford University. At the same time, **LOOM** [178, 179] was built initially for general knowledge bases using DL and production rules. LOOM provides automatic concept classificaation features. **OCML** (Options Configuration Modeling Language) [50] was invented in 1993, as a kind of operational Ontolingua for developing executable ontologies and models in problem solving methods (PSMs). In 1995, FLogic [116] was appeared as a language that combined frames and FOL.

The boom of the Internet led to the creation of ontology languages for exploiting the characteristics of the web. Such languages are usually called *web-based ontology languages* or *ontology markup languages*. Their syntax is based on existing markup languages such as HTML (Hyper Text markup Language) [43] and XML (eXtensible markup Language) [205], whose purpose was not for ontology development, but data representation and exchange on the web. The first ontology markup language to appear was **SHOE** [196]. SHOE is a language that combines frames and rules. It was built an extension of HTML, in 1996. Later, its syntax was adpted to XML. **XOL** (XML-Based Ontology Exchange Language) [163] is a language for exchange of ontologies. The language is intended to be used as an intermediate language for transferring ontologies among different database systems, ontology-development tools or application programs. **RDF** (Resource Description Framework) [161] was developed by the W3C (World Wide Web Consortium) as a semantic-network-based language to describe web resources. Its development started in 1997, and RDF was proposed as a W3C Recommendation in 1999. The **RDF Schema** [37] language was also built by the W3C as an extension to RDF with frame-based primitives. The combination of RDF and RDF Schema is known as **RDF(S)**.

Three additional languages have been developed as extensions to RDF(S): OIL, DAML +OIL, and OWL. **OIL** (Ontology Inference Layer) [79, 80] was developed at the begining of the year 2000 in the framework of the European IST project On-To-Knowledge[25]. It adds frame-based KR primitives to RDF(S) and its formal semantics is based on description logics. **DAML+OIL** (DAML-DARPA Agent markup Language) [81] was created between the years 2000 and 2001 by a joint committee from the US and the EU in the context of the DARPA project DAML[26]. It was based on the previous DAML ontology language specification and OIL. DAML+OIL adds DL-based KR primitives to RDF(S). In 2001, the W3C formed a working group called Web-Ontology (WebOnt) Working Group[27]. The aim of this group was to make a new ontology markup language for the Semantic Web. The result of their work is **OWL** (Web Ontology Language) [106]. OWL covers most of the features of DAML+OIL, but renames them.

Figure 2.7 depicts a classification of ontology languages, by adopting from [160]. The background logic of all traditional ontology languages is FOL. Later, F-logic and DL are integrated into the languages. Web standard languages are developed by combining the

[25] `http://www.ontoknowledge.org`
[26] `http://www.daml.org`
[27] `http://www.w3.org/2001/sw/WebOnt/`

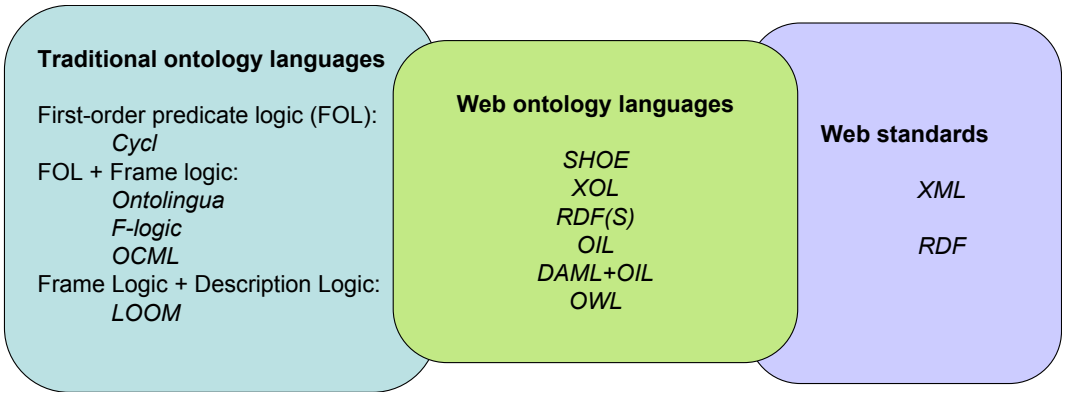

Figure 2.7: A classification of ontology languages

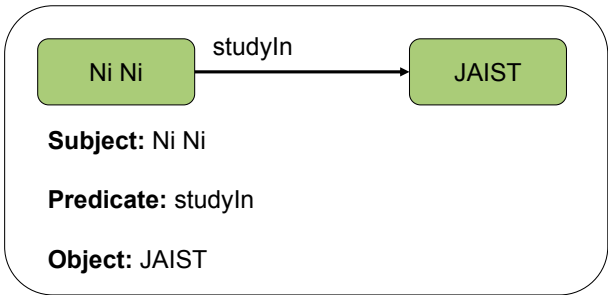

Figure 2.8: A simple view of RDF triple

web standards XML and RDF onto the traditional languages.

2.3.2 RDF-Resource Description Framework

Resource Description Framework (RDF) is a graphical language used for representing information about resources on the web. It is a basic of all web ontology languages. RDF statements are expressed in the form of RDF triples: subject, predicate, and object. Figure 2.8 shows a simple view of *RDF triple*.

RDF uses URIRefs to identify resources. A URIRef consists of a URI and an optional Fragment Identifier separated from the URI by the hash symbol ♯. For example, http://www.jaist.ac.jp/is/student♯nini. A set of URIRefs is known as a *vocabulary*. A set of linked RDF statements (triples) forms an *RDF Graph*. The subject of one statement can be the object of another statement. A sample RDF graph depicted for a personal ontology (denoted by 'po') is shown in Figure 2.9. For ontology 'po', its vocabulary may consist of `po:hasName`, `po:hasDOB`, `po:hasAddress`, `po:hasHomepage`, `po:studyIn`, `po:hasFriend`, `po:Person`, and so on.

The *RDF Vocabulary* is the set of URIRefs used in descibing the constructs of RDF language, for example, `rdf:Property`, `rdf:Resource`, `rdf:type`. The *RDFS Vocabulary* is the set of URIRefs used in describing the RDFS language. Some of RDFS built-in constructs are `rdfs:Class`, `rdfs:subClassOf`, `rdfs:domain`, `rdfs:range`, etc. Figure 2.10 illustrates a simple usage of RDFS to define an ontology. The `rdf:type` property is used to state any instance of a class. RDF and RDFS provide basic capabilities for defining the vocabularies of ontologies.

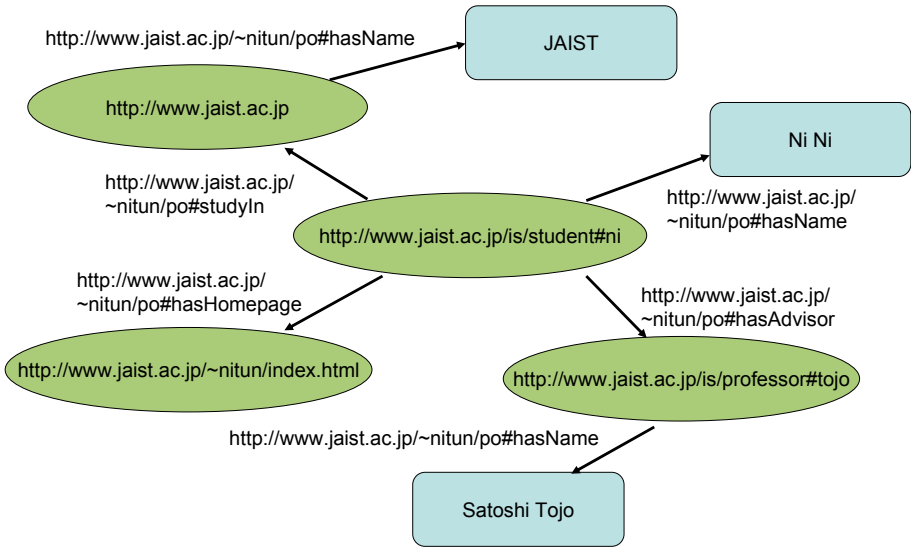

Figure 2.9: A sample RDF graph

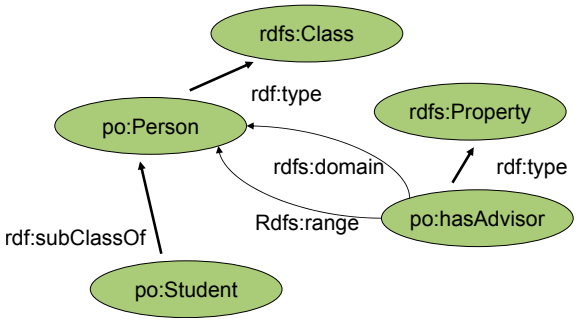

Figure 2.10: A usage of RDFS

2.3.3 OWL-Ontology Web language

Ontology Web language (OWL)[28] is the latest W3C proposed recommendation for that purpose. OWL has three increasingly-expressive sub-languages, namely, OWL Lite, OWL-DL, and OWL Full. OWL Lite was designed for easy implementation of concept hierarchy and simple constraints, and to provide users with a functional subset that will get them started in the use of OWL. OWL-DL was designed to support the existing Description Logic and to provide a language subset with desirable computational properties for reasoning systems. The complete OWL language (called OWL Full) relaxes some of the constraints on OWL-DL so as to make available features of using many database and knowledge representation systems, but which violates the constraints of current DL reasoners. Ontology developers should consider which sub-languages best suits their needs.

There are three components of OWL: classes, properties, and individuals. A class is interpreted as a set of individuals, and a property generally describes a relationship between two individuals. OWL is primarily designed to describe and define classes. Classes are therefore the basic building blocks of an OWL ontology. OWL supports six main ways of describing classes, as follows:

- *Named class* is the simplest one defined by using `owl:Class`. Every individual

[28] `http://www.w3.org/2004/OWL/`

in the OWL world is a member of the class `owl:Thing`. Thus each user-defined class is implicitly a subclass of `owl:Thing`. Domain-specific classes are defined by simply declaring a named class. OWL named classes can be organized in a taxonomy with subsumption (or set inclusion) relationship, using a subclass property `rdfs:subClassOf`.

- *Intersection classes* are formed by combining two or more classes with property `owl:intersectionOf`.
- *Union classes* are formed by integrating two or more classes with property `owl:unionOf`.
- *Complement classes* are specified by negating another class. `owl:complementOf` is an OWL construct for that purpose.
- *Restrictions* describe a class of individuals based on the type and possibly number of relationships that they participate in. Restrictions can be grouped into three main categories: two quantifier restrictions (`owl:someValuesFrom` for Existential $\exists$, `owl:allValuesFrom` for Universal $\forall$), two cardinality restrictions (`owl:maxCardinality`, `owl:minCardinality`), and hasValue restriction (`owl:hasValue`). The most common type of restriction is the existential restriction which means some values from, or 'at least one'. An existential restriction describes the class of individuals that have at least an individual that is a member of a specified class. Universal means 'all values from', or 'only'. For a given property, all the individuals must be members of a specified class. Cardinality restrictions allow us to talk about the number of relationships that a class of individuals participate in. `owl:hasValue` restriction allows us to specify a class of individuals that participate in a specified relationship with a specific value.
- *Enumerated classes* are specified by explicitly and exhaustively listing the individuals that are members of the enumerated class.

Individuals are the instances of the universe of discourse. An individual is minimally introduced by declaring it to be a member of a class.

There are two main categories of OWL properties: object properties and datatype properties. Object properties (`owl:ObjectProperty`) link individuals to individuals. Datatype properties (`owl:DatatypeProperty`) link individuals to XML Schema datatype values, for example `xsd:integer`, `xsd:float`, `xsd:string`, etc., or other user defined datatype values. OWL properties have a specified domain and range by using `rdfs:domain` and `rdfs:range`. The certain characteristics of OWL properties are described in the following.

- *Functional (*`owl:FunctionalProperty`*)*: For a given individual, the property takes only one value, and not more than one.
- *Inverse functional (*`owl:InverseFunctionalproperty`*)*: The inverse of the property is functional.
- *Symmetric (*`owl:SymmetricProperty`*)*: If a property links A to B then it can be inferred that it links B to A.

```
<owl:Class rdf:ID="Person"/>
        <owl:DatatypeProperty rdf:ID="hasName">
                <rdfs:domain rdf:resource="#Person"/><rdfs:range rdf:resource="&xsd;string"/>
        </owl:DatatypeProperty>
        <owl:DatatypeProperty rdf:ID="hasDOB">
                <rdfs:domain rdf:resource="#Person"/><rdfs:range rdf:resource="&xsd;date"/>
        </owl:DatatypeProperty>
</owl:Class>
        <owl:Class rdf:ID="UnivStudent">
        <rdfs:subClassOf rdf:resource="#Person"/>
        <owl:Restriction>
                <owl:onProperty rdf:resource="#StudyIn"/>
                <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
        </owl:Restriction>
</owl:Class>
<owl:Class rdf:ID="PhDStudent">
        <rdfs:subClassOf rdf:resource="#UnivStudent"/>
        <owl:Restriction>
                <owl:onProperty rdf:resource="#EnrollForDegree"/><owl:someValuesFrom rdf:resource="#PhD"/>
        </owl:Restriction>
</owl:Class>
<owl:Class rdf:ID="Professor">
        <rdfs:subClassOf rdf:resource="#Person"/>
        <owl:Restriction>
                <owl:onProperty rdf:resource="#WorkIn"/>
                 <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
         </owl:Restriction>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasAdvisor">
        <rdfs:domain rdf:resource="#UnivStudent"/><rdfs:range rdf:resoruce="#Professor"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="EnrollForDegree">
        <rdfs:domain rdf:resource="#UnivStudent"/><rdfs:range rdf:resoruce="#Degree"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="Degree">
        <owl:oneOf rdf:parseType="Collection">
                <owl:Thing rdf:about="#PhD"/><owl:Thing rdf:about="#MS"/><owl:Thing rdf:about="#BE"/>
        </owl:oneOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="StudyIn">
       <rdfs:domain rdf:resource="#UnivStudent"/> <rdfs:range rdf:resoruce="#University"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="WorkIn">
       <rdfs:domain rdf:resource="#Professor"/> <rdfs:range rdf:resoruce="#University"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="University">
        <owl:oneOf rdf:parseType="Collection">
              <owl:Thing rdf:about="#MIT"/><owl:Thing rdf:about="#Stanford"/><owl:Thing rdf:about="#JAIST/>
        </owl:oneOf>
</owl:Class>
<Professor rdf:ID="tojo">
        <hasName rdf:datatype="&xsd:string">Satoshi Tojo</hasName>
        <hasDOB rdf:datatype="&xsd;date">11/09/54</hasDOB>
        <WorkIn rdf:resource="#JAIST</WorkIn>
</Professor>
<PhDStudent rdf:ID="Ni">
        <hasName rdf:datatype="&xsd:string">Ni Ni</hasName>
        <hasDOB rdf:datatype="&xsd;date">24/07/70</hasDOB>
        <StudyIn rdf:resource="#JAIST</StudyIn>
        <EnrollForDegree rdf:resource="#PhD"/>
        <hasAdvisor rdf:resource="#tojo"/>
</PhDStudent>
```

Figure 2.11: An example OWL ontology

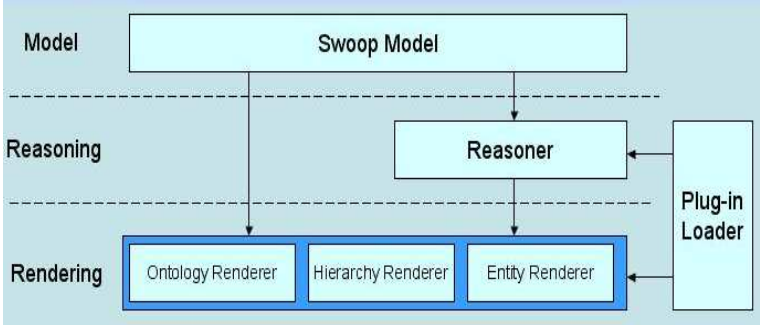

Figure 2.12: The architecture of SWOOP

- *Transitive (*`owl:TransitiveProperty`*)*: If a property links A to B and B to C then it can be inferred that it links A to C.

OWL properties can have a subproperty relationship. `rdfs:subpropertyOf` is a construct for that. A detailed expression of OWL is available at [54, 55, 56, 57]. Ipresent an example OWL ontology in Figure 2.11.

2.4 Ontology Development Tools and Tool Suites

Ontology development tools have improved enormously since the creation of the first environments in the mid-1990s. The tools can be divided into two groups as follows:

- *Language dependent* tools are those whose knowledge model maps directly to an ontology language. They are developed as ontology editors for a specific language. Some examples are the Ontolingua server [5] with Ontolingua and KIF, Ontosaurus [30] with LOOM, WebOnto [88] with OCML, OilEd [184] with OIL at first, and later with DAML+OIL, and SWOOP[29].

- *Language independent* tool suites are ontology development-environments whose main characteristic is that they have an extensible architecture, and whose knowledge model is usually independent of an ontology language. These tool suites provide a core set of ontology-related services and are easily extended wth other tools to provide more flexible functions. Protégé[30], WebODE[31], OntoEdit[32], and KAON[33], are included in this group.

Among the tool suites, a brief introduction about SWOOP and protégé are presented below.

2.4.1 SWOOP

SWOOP is a hypermedia-inspired OWL-based web ontology browser and editor, which is implemented by the Maryland Information and Network Dynamics Lab Semantic Web

[29] `http://www.mindswap.org/2004/SWOOP/`
[30] `http://protege.stanford.edu`
[31] `http://webode.dia.fi.upm.es/webode/jsp/webode/frames.jsp`
[32] `http://ontoserver.aifb.uni-karlsruhe.de/ontoedit/`
[33] `http://kaon.semanticweb.org/`

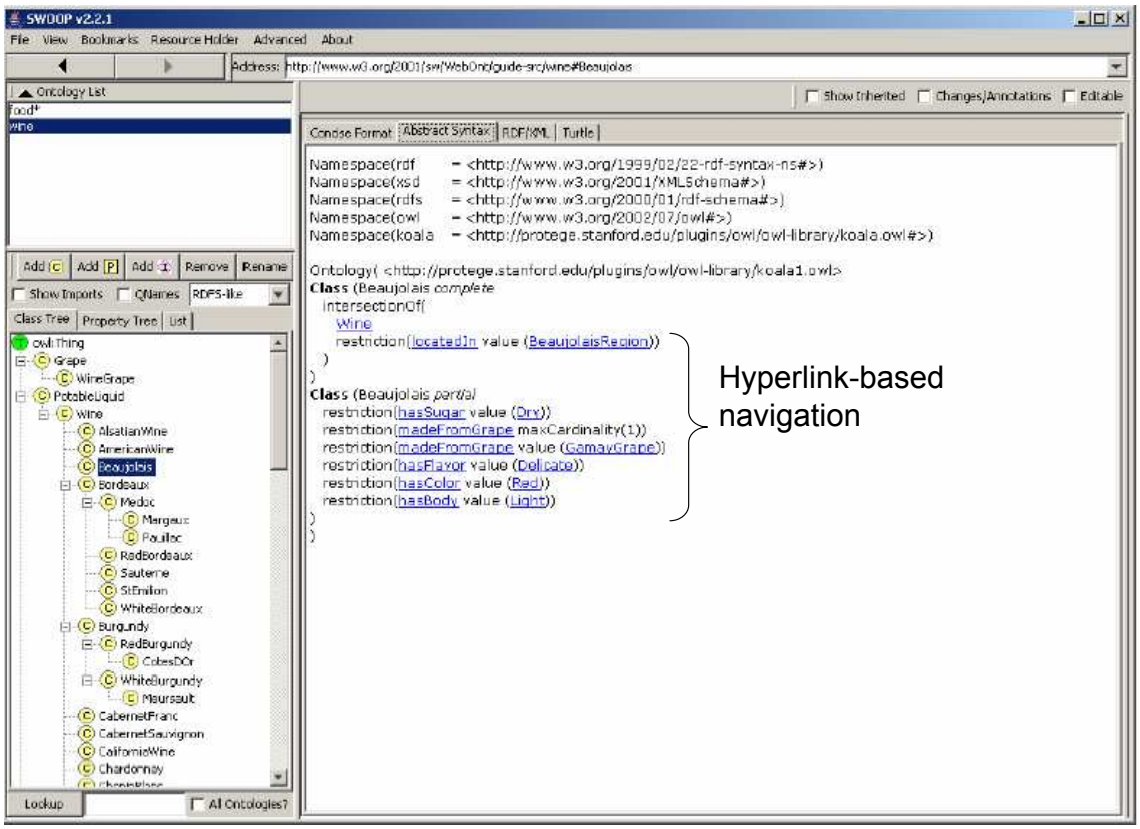

Figure 2.13: A screenshot of SWOOP Web Ontology Browser

Agents Project [12, 13, 14, 15, 16]. The current version is v2.2.1, that is downloadable at MINDSWAP project page[34]. The main purpose of SWOOP is to develop OWL ontologies concerning the Semantic Web.

Swoop is based on the Model-View-Controller (MVC) paradigm [176]. In the architecture described in Figure 2.12, the *SwoopModel* component stores all ontology-centric information. Control is handled through a plugin based system, which loads new Renderers and Reasoners dynamically. Swoop is implemented by using Java, as an editor for OWL ontologies. Thus, all OWL reasoners can be integrated for consistency checking in terms of open-world semantics. Since Swoop intends to provide a web browser for the Semantic Web, its user interface supplies the feature of hyperlink-based navigation, as shown in Figure 2.13. This is the main distinctive feature of SWOOP from other OWL editors.

2.4.2 Protégé

Protégé 3.2 beta[35] is the latest version of the Protégé tools, created by the Stanford Medical Informatics (SMI) group at Stanford University. The first Protégé tool was created in 1987 [118]; its main aim was to simplify the knowledge acquisition process for expert systems. To achieve this objective, it used the knowledge acquired in the previous stages of the process to generate customised forms of acquiring more knowledge. Since

[34] `http://www.mindswap.org/2004/SWOOP`
[35] `http://protege.stanford.edu/download/registered.html`

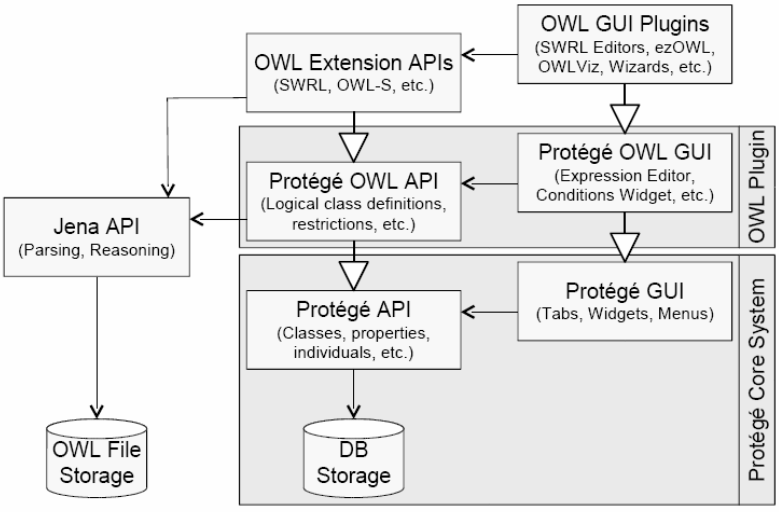

Figure 2.14: The architecture of OWL plug-in extended on Protégé core system

then, Protégé has gone through several releases, and has focused on different aspects of knowledge acquisition (knowledge bases, problem solving methods, ontologies, etc.,), the result of which is Protégé 3.2 beta. The evolution of Protégé was described by Gennai and colleagues [89]. Protégé has a community of 17,000 members and 52,000 registered users, at the moment. Although the development of Protégé has historically been mainly driven by biomedical applications [4], the system is domain-independent, and has been successfully used for many other application areas as well.

Architecture

Like most other modeling tools, the architecture of Protégé is cleanly separated into a "model" part and a "view" part. Protégé's model is the internal representation mechanism for ontologies and knowledge bases. Protégé's view components provide a user interface to display and manipulate the underlying model. Protégé metamodel (also known as knowledge model) is based on frames and FOL. The main modeling components of Protégé are classes, slots, facets, and instances. Protégé has an extensible architecture for creating and integrating easily new extensions (aka plug-ins). Most of these plug-ins are available from the Protégé Plug-in Library[36], where contributions from many different research groups can be found.

Among those plug-ins, the OWL plug-in is a complex Protégé extension that can be used to edit OWL files and databases. The OWL plug-in includes a collection of custom-tailored tabs and widgets for OWL, and provides access to OWL-related services such as consistency checking, taxonomy classification, inferencing, ontology testing, graphical view of OWL classes, and OWL query with SparQL. As illustrated in Figure 2.14, the OWL plug-in extends the Protégé model and its API with classes to represent OWL specification. The OWL plug-in supports RDF(s), OWL Lite, OWL DL (except for anonymous global class axioms, which need to be given a name by the user) and significant parts of OWL Full (including metaclasses). The OWL plug-in provides a comprehensive

[36] `http://protege.stanford.edu/download/plugins.html`

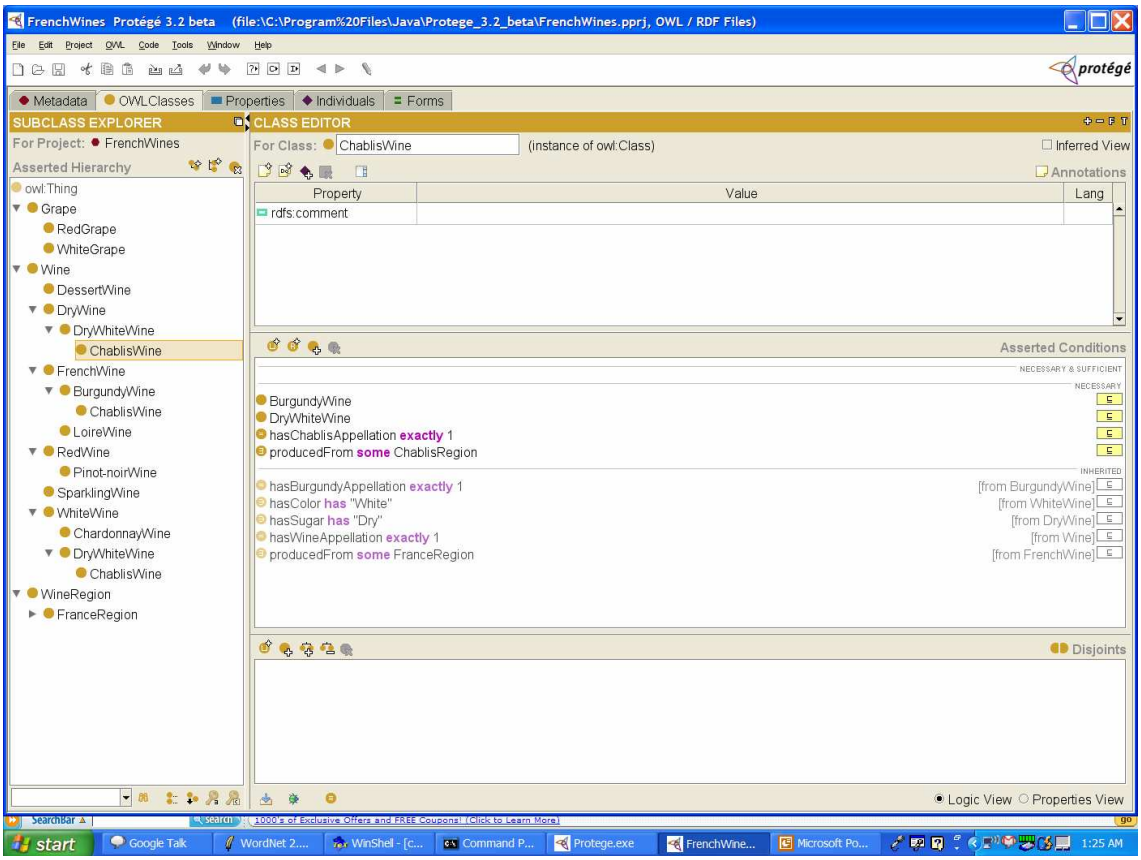

Figure 2.15: A screenshot of the OWL class tab in Protégé

mapping between its extended API and the standard OWL parsing library Jena[37]. There are many ways to access Protégé ontologies from ontology-based applications. All the ontology components can be accessed with the Protégé Java API. Hence it is easy for ontology-based applications to access ontologies, as well as use other functions provided by different plug-ins. This API is also available through a CORBA-based[38] server so that remote clients can access Protégé ontologies. Moreover, Protégé ontologes can be exported and imported with some backends: RDF(S), XML, UML, XMI, and OWL.

OWL Plugin User Interface

The Protégé OWL Plugin provides several custom-tailored graphical user interface components for OWL. When started, the system displays five tabs: Metadata, OWL classes, Properties, Individuals, and Forms, as depicted in Figure 2.15. Most ontology designers will focus on the OWL classes and Properties tabs. The Forms and Individuals tabs are mostly geared for the acquisition of Semantic Web contents (instance data), while the Metadata tab allows users to specify global ontology settings such as imports and namespaces.

The generic architecture of Protégé and the OWL-specific extensions make it relatively easy to add custom-tailored components. For example, optimized editors for Semantic

[37] `http://jena.sourceforge.net`
[38] Common Object Request Broker Architecture

Web Rule language (SWRL) [85] or OWL-based Web Service Ontology (OWL-S)[39] could be added to the system. Likewise, description-logic reasoners such as Racer[40], could be directly implemented on top of the Protégé OWL API or Jena. Since all of these components are available as open-source, it is possible to extend and customize them.

The OWL Plugin provides direct access to DL reasoners such as Racer, Pellet. The current user interface supports two types of DL reasoning: consistency checking and classification.

Protégé Axiom Language

The Protégé Axiom Language (PAL)[41] extends the Protégé knowledge modeling environment with support for writing and storing logical constraints and queries about frames in a knowledge base. More than just a language, PAL is a plugin toolset that comprises engines for checking constraints and running queries on knowledge bases, as well as a set of useful user interface components. More specifically, the features of PAL toolset are:

- a language to express logical constraints and queries in a knowledge base;
- a set of special-purpose frames to model constraints and queries;
- a structured editor that provides context-sensitive help to write PAL sentences;
- a constraint-checking engine, which can be invoked either with the PAL Constraints Tab or programmatically; and
- a querying engine which can be invoked either with the PAL Queries Tab or programmatically.

The primary purpose of the PAL is exactly to support the definition of such arbitrary logical constraints on the frames of a knowledge base. PAL constraints are modeled with special-purpose frames (called meta-classes) and thus can be saved as part of the knowledge base. The PAL constraint-checking engine can be run against the knowledge base to detect frames that violate those constraints. Note that the underlying philosophy of PAL is model-checking rather than theorem-proving. In other words, PAL makes strong *closed world assumptions* and is used for writing restrictions on existing knowledge, not for asserting new knowledge. The primary goals of PAL are therefore to detect incomplete entry of information and to check entered information for inconsistencies.

A constraint (or a query) is a statement that holds on a certain number of variables, which range over a particular set of values. Therefore, a constraint or query in PAL consists of a set of variable range definitions and a logical statement that must hold on those variables. The language of PAL is a limited predicate logic extension of Protégé that supports the definition of such ranges and statements. A PAL constraint statement consists of a sequence of sentences linked by logical connectives. A sample PAL constraint written for "the salary of an editor should be greater than the salary of any employee whom the editor is responsible for" is as described in Figure 2.16.

To write those sentences, PAL supports a number of predefined predicates and functions, that can be used in constraint statements to test or compute properties of variables.

[39] `http://www.daml.org/services/owl-s/1.1B/`
[40] `http://www.sts.tu-harburg.de/ r.f.moeller/racer/`
[41] `http://protege.stanford.edu/plugins/paltabs/pal-quickguide/`

```
(forall ?editor (forall ?employee
(=> (and
                (responsible_for ?editor ?employee)
                (own-slot-not-null salary ?editor)
                (own-slot-not-null salary ?employee))
 (> (salary ?editor) (salary ?employee)))))
```

Figure 2.16: A sample PAL constraint

In addition, any slot in a knowledge base can be used directly as a function or a unary predicate in PAL statements. Finally, all variables that appear in a constraint statement must be quantified, either with the universal quantifier (forall) or with the existential quantifier (exists). Documentation of the PAL Structure Editor can be found in the comprehensive documentation of the PAL plugin[42].

2.4.3 Concluding Remarks

As I have discussed in above, SWOOP is a URI-based Web ontology editor while Protégé is a frame-based ontology editor. Although both editors provide the development of OWL ontologies, I choose Protégé OWL API to use in the later enrichment process with respect to the following utilities.

- Protégé knowledge model supports the creation of customized *meta-classes*[43].
- Protégé Axiom Language (PAL) is available to create internal constraints. And "PAL constraints" plugin is provided for verification and query on the defined knowledge. Moreover, Protégé supports to write PAL constraints in OWL ontologies.
- Protégé ontologies can be exported into a variety of formats including RDF(S), OWL, and XML Schema.

[42]http://protege.stanford.edu/plugins/paltabs/pal-documentation/
[43]A meta-class is a frame template that is used to define new classes in an ontology.

Chapter 3

Ontology Matching and Semantic Heterogeneity

The aim of this chapter is to provide a survey of tools and techniques for automatic and semi-automatic ontology matching. I will start by exploring the terminology related to ontology matching, and then will provide a clear definition and prototype for that. To lay the foundation of ontology matching, I first examine the classifications of semantic heterogeneity which are characterized by mismatches between ontological entities, and then propose my classification. A number of matching mechanisms has been proposed in the literature. An objective style review of the tools and methods will be presented in this chapter.

3.1 What is Ontology Matching

In general, ontology matching is a process that takes two ontologies as input and retrieves similarity relationships between their entities as output.

In order to set the context and scope for ontology matching precisely, first, I list some definitions of the term "ontology matching (or mapping)" in the literature.

- In [92], it is defined that a mapping will be a set of formulae that provide the semantic relationships between the concepts in the models.
- In [131], it is explained that matching is to establish correspondences among the source ontologies, and to determine the set of overlapping concepts, that are similar in meaning but have different names or structure, and concepts that are unique to each of the sources.
- In [40], it is stated that the aim of mapping is to map concepts in the various ontologies to each other, so that a concept in one ontology can be queried to other ontologies to find corresponding concepts.

To sum up, ontology matching is a process to find semantically similar concepts between ontologies. Due to the wide range of expressions used in this area, I want to describe my understanding of the term *ontology matching* as:

> *Given two ontologies O_1 and O_2, ontology matching means for each concept (node) in ontology O_1, a corresponding concept (node) which has the same or similar semantics is discovered in ontology O_2, and vice verse.*

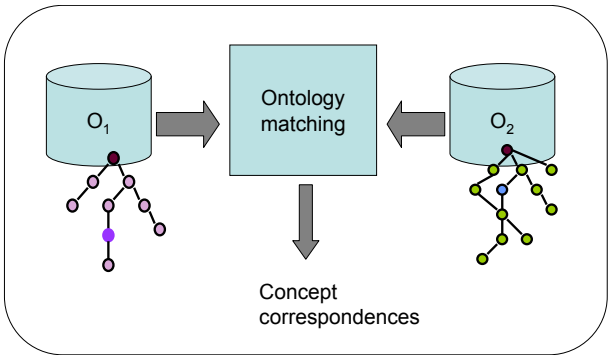

Figure 3.1: A view of ontology matching

The idea is depicted in Figure 5.2. A well defined matching process can be considered as a component which provides a mapping service. This service can be plugged into various applications. Through this work, I will use the following terms consistently according to their given specific meaning:

- **Merging or Integration**: Building a new ontology from two or more existing ontologies as an approach of knowledge reuse.
- **Aligning**: Bring two or more ontologies into mutual agreement, making them consistent and coherent with one another.
- **Correspondences**: The specification of matched relations between concepts.
- **Articulation**: The linkage between two aligned ontologies, that is, the specification of the alignment.
- **Translation**: Changing the representation formalism (or syntax) of an ontology while preserving its semantics.

3.2 Semantic Heterogeneity in Ontologies

First of all, I present a number of classification frameworks in heterogeneity, that have been discussed in the literature [167, 168, 65, 66]. Then, I will summarize them and provide my own classification of semantic heterogeneity in ontologies.

3.2.1 Klein's Mismatches

Heterogeneities between ontologies are called mismatches in [117], where a framework of issues related to the integration of ontologies, is depicted in Figure 3.2. Among the three issues discussed, practical problems, mismatches between ontologies, and versioning, the main concern in this thesis is related to the number of mismatches between ontologies. These mismatches can be described generally as occurring at the language level, and/or the ontology level. The former conforms to the syntactic layer, and the latter to the semantic layer. I describe a summary of ontology level mismatches presented by Klein, below.

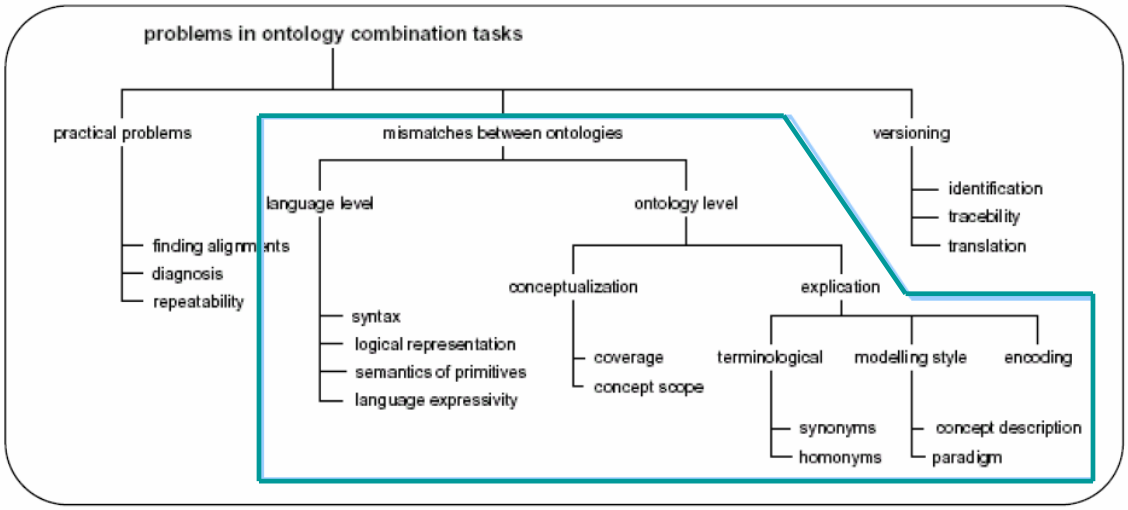

Figure 3.2: Different levels of ontology mismatches given in [117]

Ontology Level Mismatches are the mismatches happened at the ontology level when two or more ontologies—which have overlapping domains—are combined. These mismatches occur when the ontologies are written in the same language, as well as when they use different languages. At the ontology level a distinction is made between conceptualization and explication, as described in [168]. A conceptualization mismatch is a difference in the way a domain is interpreted, whereas an explication mismatch is a difference in the way the conceptualization is specified.

- *Conceptualization mismatches* are further divided into model coverage and concept scope (granularity).
  - *Model coverage and granularity.* This is a mismatch in the part of the domain that is covered by the ontology, or the level of detail to which that domain is modeled. Chalupsky [66, 67] gives the example of an ontology about cars: one ontology might model cars but not trucks. Another one might represent trucks but only classify them into a few categories, while a third ontology might make very finegrained distinctions between types of trucks based on their physical structure, weight, purpose, etc.
  - *Concept scope.* Two classes seem to represent the same concept, but do not have the same instances, although they may intersect. The classical example is the class "employee", where several administrations use slightly different concepts of employee, as mentioned by Wiederhold [65].
- *Explication mismatches* are divided into terminological, modeling style, and encoding.
  - *Terminological mismatches:* Two types of differences can be classified as terminological mismatches.
    - *Synonym terms:* Concepts are represented by different names. One example is the use of the term "car" in one ontology and the term "automobile" in another ontology.

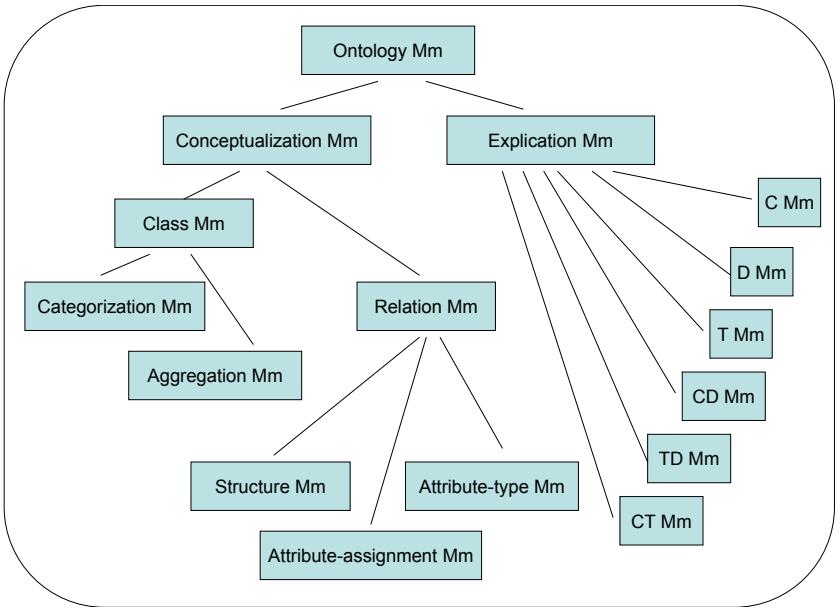

Figure 3.3: A classification of ontology mismatches given in [168]

  - ∗ *Homonym terms:* The meaning of the same term is different in different contexts. For example, the term conductor has a different meaning in a music domain than it has in an electric engineering domain.
- *Modeling style:* This is related to the paradigm and conventions taken by the developers.
  - ∗ *Paradigm:* Different paradigms can be used to represent concepts such as time, action, plans, causality, propositional attitudes, etc. For example, one model might use temporal representations based on interval logic, while another might use a representation based on point logic [66].
  - ∗ *Concept description:* This type of difference is called modeling convention in [66]. Several choices can be made for the modeling of concepts in the ontologies. For example, a distinction between two classes can be modeled using a qualifying attribute or by introducing separate class.
- *Encoding:* One last mismatch in the explication category is encoding. Encoding mismatches are differences in value formats, like measuring distance in miles or in kilometers.

3.2.2 Visser's Mismatches

In the work of Visser et al [167, 168], a detailed assessment of semantic heterogeneity is presented by classifying a number of ontology mismatches. They classified ontology mismatches into two levels: conceptualization mismatches and explication mismatches, similarly to Klein. A detailed classification of both levels described in [168] is shown in Figure 3.3. We briefly summarize the mismatches below.

- *Conceptualisation mismatches* are mismatches between two (or more) conceptualisations of a domain. The conceptualisations differ in the ontological concepts distinguished or in the way these concepts are related.

  - A *Class mismatch* is a conceptualisation mismatch relating to the classes distinguished in the conceptualisation. In particular, this type of mismatch concerns classes and their subclasses.
  - A *categorisation mismatch* occurs when two conceptualisations distinguish the same class but divide this class into different subclasses. As an example, Visser gave two categorizations about animals: mammals and birds, or carnivores and herbivores.
  - An *aggregation-level mismatch* occurs if two conceptualisations both recognise the existence of a class, but define classes at different levels of abstraction. For instance, one conceptualisation may distinguish person as male and female, but it does not have person as their superclass.

- A *relation mismatch* is a conceptualisation mismatch relating to the relations distinguished in the conceptualisation. Relation mismatches concern, for instance, the hierarchical relations between two classes, or the assignment of attributes to classes.
  - A *structure mismatch* occurs when two conceptualisations distinguish the same sets of classes, but differ in depicting their relations.
  - An *attribute-assignment* mismatch occurs when two conceptualisations differ in the way they assign an attribute to classes.
  - An *attribute-type mismatch* occurs when two conceptualisations distinguish the same attribute in two different datatypes or units, such as 'Distance' in meters and in kilometers.

- *Explication mismatches* are not defined on the conceptualisation of the domain but on the way the conceptualisation is specified.
  - A *CT mismatch* (or Concept and Term mismatch) occurs when two ontologies use the same definiens D, but differ in both the concept C they define and the term T linked to the definiens. For instance, both *Vessel* and *Whale* are defined as something which is large and sea going.
  - A *CD mismatch* (or Concept and Definiens mismatch) occurs when two ontologies use the same term T, but differ in the concept C they define and the definiens D used for the definition. Consider the term *Mitre*. One ontology may define the concept of the headgear of a bishop, whereas a second ontology may define the concept of a straight angled joint of wood.
  - A *C mismatch* (or Concept mismatch) occurs when both ontologies have the same term T and definiens D, but differ in the concepts they define.
  - A *TD mismatch* (or Term and Definiens mismatch) occurs when two ontologies define the same concept C but differ in the way they define it; both with respect to the term T and the definiens D. For instance, *Vessel* and *Ship* are defined as things which are large and floating.
  - A *T mismatch* (or Term mismatch) occurs when two ontologies define the same concept C using the same definiens D, but refer to it with different terms, such as *Vessel* and *Ship*.

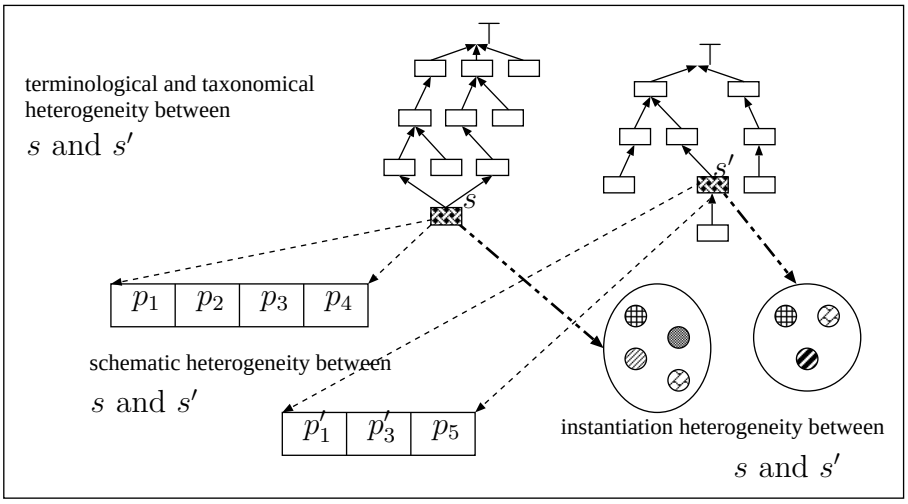

Figure 3.4: A general illustration of semantic heterogeneity between two classes

– A *D mismatch* (or Definiens mismatch) occurs when two ontologies define the same concept C and use the same term T to refer to the concept, but use different definiens. For example, *Mitre* in two different definitions.

3.2.3 An Alternative Classification of Semantic Heterogeneity

In summary, the above mismatches can be distinguished in terms of syntactic heterogeneity and semantic heterogeneity. *Syntactic heterogeneity* is caused by using different ontology modeling paradigms (e.g., XML-based model or Frame-based model) and different ontology languages (e.g. DAML or OWL), while *semantic heterogeneity* is created by conceptualization divergence in describing the semantics of ontological classes.

Dealing with semantic heterogeneity in ontologies is strongly related to information integration of heterogeneous databases and systems [31, 198]. Since ontologies have close relations to both knowledge-based systems and database systems, this classification of semantic heterogeneity in ontologies will concern both.

Ceri and Widom [186] list four categories of semantic conflicts in databases: naming conflicts, domain conflicts, meta-data conflicts, and structural conflicts. In knowledge acquisition, Shaw and Gaines [122] have proposed a method to compare and resolve conflicting conceptualizations by domain experts in terms of four different dimensions: (a) *consensus* if the same name is used for the same semantics, (b) *correspondence* if different names are used for similar or equivalent semantics, (c) *conflict* if the same name is used for different semantics, and (d) *contrast* if different names are used for different semantics.

Since ontology mapping is a process to find *correspondences* between semantically related classes among heterogeneous ontologies, I classified semantic heterogeneity between ontology classes into four categories. For two semantically similar or equivalent classes, there is

1. *terminological* heterogeneity if they have different names or labels;

2. *taxonomical* heterogeneity if they have different taxonomic (or subsumption) structures;

3. *schematic* heterogeneity if they have different sets of properties and constraints;

4. *instantiation* heterogeneity if they are defined using different sets of individuals.

Figure 3.4 illustrates these heterogeneities between two semantically similar classes: s and s'.

> *Ontologies which are conceptualized for the same or overlapped domain in the form of the above kinds of semantic heterogeneities are called semantically-heterogeneous ontologies (in short heterogeneous ontologies).*

For two semantically-same concept names, terminological heterogeneity may basically involve. Moreover, the number of properties defined for those concept names can not be the same by different subsumption levels in ontologies. In addition, instantiation heterogeneity can exist between them. This condition is called *wide-scale semantic heterogeneity*. In this research, I propose a semantically-enriched model of ontologies in order to deal with matching between wide-scale heterogeneous ontologies.

3.3 Ontology Matching Tools and Techniques

To discover the correspondences between ontologies, several different approaches have been proposed in the literature [131, 137, 17, 4, 64, 42, 220, 197]. Semantic correspondences are found in roughly two ways: (1) applying a set of matching rules or heuristics, and (2) evaluating similarity measures that compare a set of possible correspondences and help to choose valid correspondences from them.

The creation of correspondences will rarely be completely automated. However, automated tools can significantly speed up the process and provide efficient matching. In large domains, while many correspondences might be fairly obvious, some parts need expert-interaction. There are two general approaches for building ontology mapping tools.

- *The first is to use a wide range of heuristics to generate correspondences.* The heuristics are often based on names or structures of concepts. In some cases, domain independent heuristics may be augmented by more specific heuristics for the particular representation language or application domain.

- *A second approach is to learn for matching.* In particular, manually provided correspondences or experts' suggestions are considered as examples or training data for a learning algorithm that can generate subsequent correspondences.

Developing ontology mapping tools and methodologies has focused on a variety of works originating from diverse communities over a number of years. There are many works regarding ontology mapping, merging, alignment, and integration. Kalfoglou and Schorlemmer conducted a comprehensive survey of a total of 35 mapping-related works by classifying them into nine categories: frameworks, methods and tools, translators, mediators, techniques, experience reports, theoretical frameworks, surveys, and examples [221]. Noy and Musen also provided an evaluation-oriented analysis of some mapping tools, comparing them with their experience in PROMPT for ontology merging [137]. There is also a survey of schema-based matching approaches by Shvaiko and Euzenat [166].

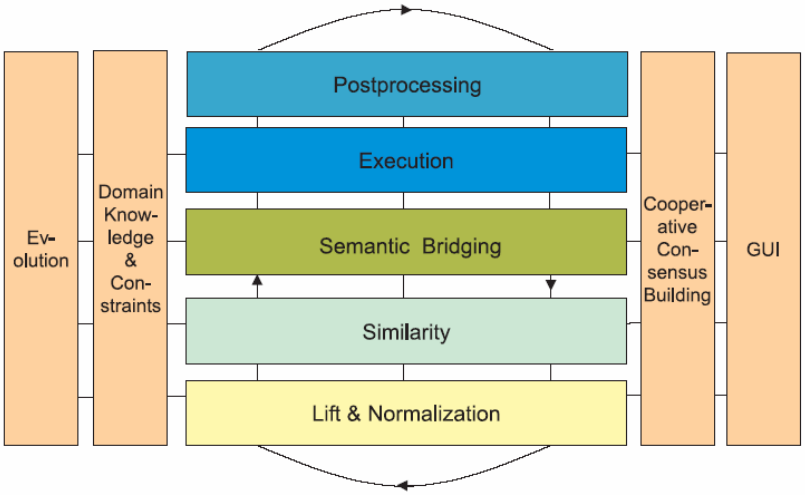

Figure 3.5: The conceptual architecture of MAFRA

Not only the major tasks but also the assumptions employed in each work are more or less different. Here, I provide an objective style review concerning how each tool deals with semantic heterogeneity, and to what extent. For that purpose, I present these methods and tools in brief, with respect to their background theory, mapping approach, and the level of expert-interaction.

3.3.1 MAFRA

MAFRA is an ontology MApping FRAmework using Semantic Bridge Ontology (SBO)—a skeleton ontology of semantic bridges (or similarity relations) between ontology components, such as concept bridges, property bridges, etc., in order to transform instances of a source ontology into instances of a target ontology. The framework consists of five horizontal modules and four vertical modules as shown in Figure 3.5 [17, 18].

Horizontal modules describe the phases of a typical mapping process, and vertical modules accompany them to assist in constructing semantic bridges and in archiving the bridges for future mapping. Among horizontal modules, the first module, *Lift & Normalization*, is a process of normalization between source and target ontologies by transforming them into a common ontology representation (RDFs), using LIFT. The second module, *similarity*, calculates similarity between ontology components employing a multi-strategy approach, mainly using lexical analysis via WordNet, domain glossaries, bi-lingual dictionaries, and corpuses. The third module, *semantic bridging*, constructs semantic bridges between concepts and between properties, based on the output of the similarity module. Figure 3.6 illustrates MAFRA's semantic bridge-based ontology mapping.

- First, concept bridging chooses a bridge between two concepts according to the similarities found in the previous phase, pairs of entities to be bridged. The same source entity may be part of different bridges.

- Second, the property bridging step is responsible for specifying the matching properties for each concept bridge. As for concepts, a property may be part of several matchings.

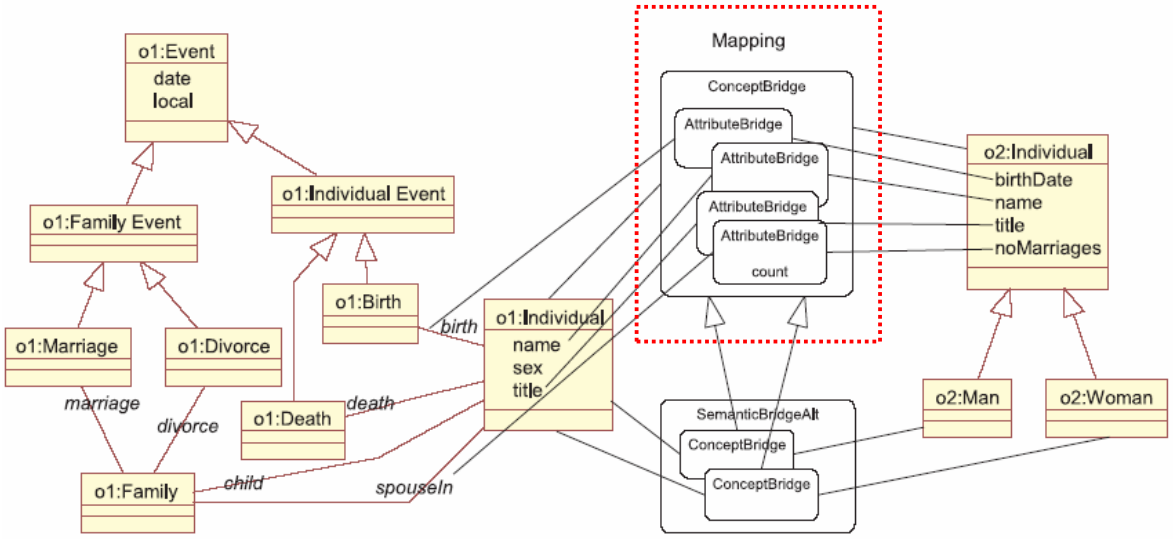

Figure 3.6: The UML representation of MAFRA's semantic-bridge-based ontology mapping [17]

- Third, the inferencing step focus in endowing the mapping with bridges for concepts that do not have a specific counterpart target concept.

The fourth module, *execution*, executes on semantic bridges in order to transform instances of source ontology into target ontology. The final module is *post-processing* that is an expert-interactive process to improve the quality of executed results.

Regarding the examples given in the work, there is no explicit deterministic heuristics other than lexical heuristics (or synonyms), in the semantic bridge construction. In the case of terminological heterogeneity with complex labels between concepts and between properties, MAFRA may not have a solution for how to determine semantic bridges.

3.3.2 ONION

ONION [164] is an heuristic-based ONtology compositION system to resolve terminological heterogeneity using two matching approaches: linguistic matching via WordNet[1] and instance-based matching via databases. ONION has a semi-automatic Articulation Generator (ArtGen) that suggests matches between source ontology and target ontology, regards the similarity score returned by any approach, and verifies the matches by domain expert. Figure 3.7 illustrates a matching of two airline ontologies using an articulation rule.

Mitra and Wiederhold argue that semantic heterogeneity can be resolved by using *articulation rules* which express the relationship between two (or more) concepts belonging to the ontologies.

Establishing such rules manually, the authors continue, is a very expensive and laborious task; on the other hand, they also claim that full automation is not feasible due to the inadequacy of today's natural language processing technology. So, they take into account relationships in defining their articulation rules, but these are limited to *subclass*

[1] `http://wordnet.princeton.edu/`

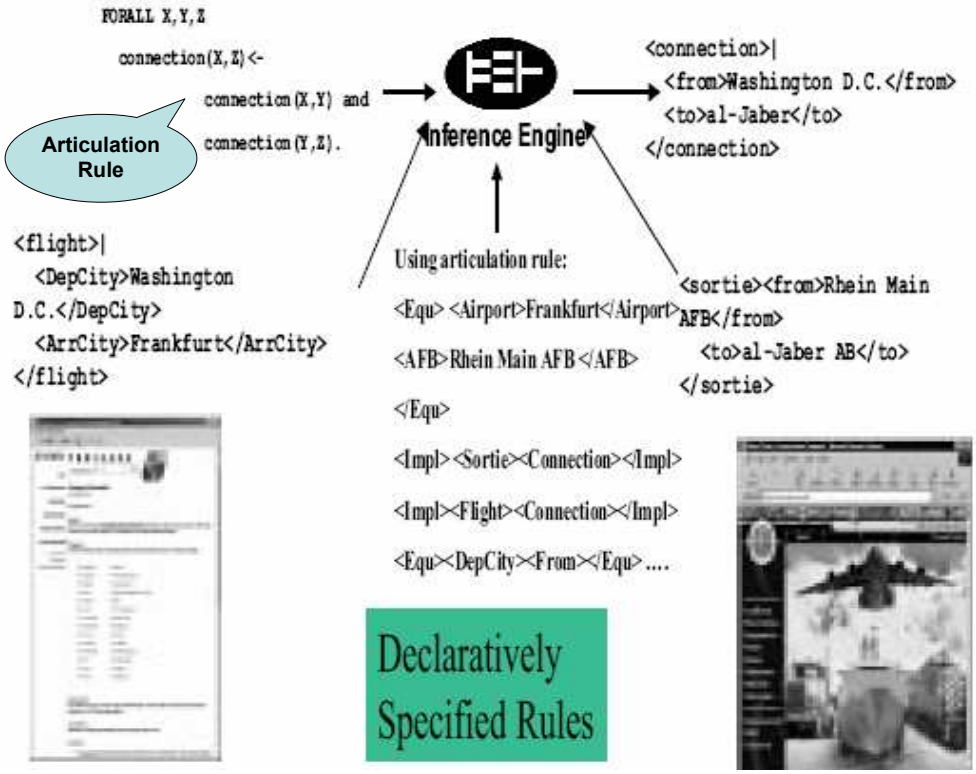

Figure 3.7: An example of matching by using an articulation rule in ONION

of, *part of*, *attribute of*, *instance of*, and *value of*. In their experiments the ontologies used were constructed manually, and represent two websites of commercial airlines. The articulation rules were also established manually. However, the authors used a library of heuristic matchers to construct them. Then, a human expert, knowledgeable about the semantics of concepts in both ontologies, validated the suggested matches. Finally, they include a learning component in the system which takes advantage of users feedback to generate better articulation in the future, when articulating similar ontologies. The algorithms used for the actual mapping of concepts are based on linguistic features.

ONION provided algorithms for how to calculate a similarity score between some complex names of ontological components such as 'Department of Defense' and 'Defense Ministry'. In my opinion, analysis of lexical semantics alone cannot fully decide the matches between domain concepts precisely.

3.3.3 PROMPT

PROMPT [137] is a semi-automatic and interactive tool suit for performing ontology mapping, alignment, versioning, and merging, based on the Frame paradigm. Noy and Musen have developed ANCHORPROMPT [134] for ontology mapping and PROMPT-DIFF [136] for ontology merging. PROMPT is available as a plugin for the open source ontology editor Protégé. For the phase of matching, AnchorPROMPT first detects linguistic similarity matches (called anchors) between ontology components. Then, Anchor-PROMPT analyzes the paths of the input ontologies delimited by the anchors in order to determine terms frequently appearing in similar positions on similar paths. Finally, based on frequencies and user feedback, AnchorPROMPT determines matching candidates (or

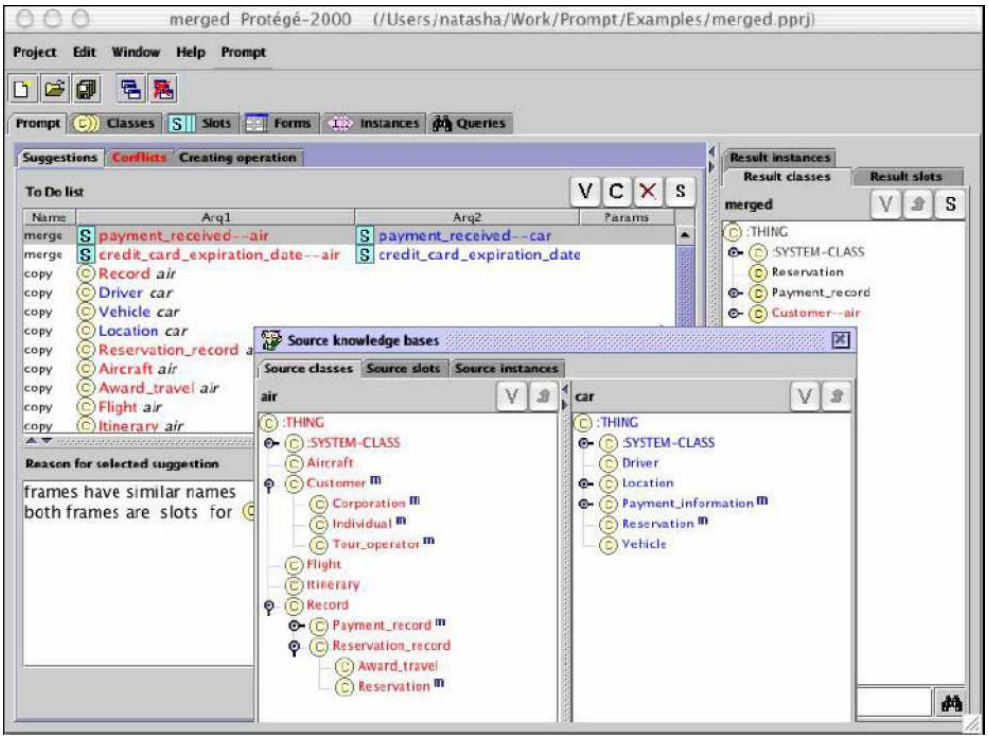

Figure 3.8: A screenshot of PROMPT

correspondences). The limitation of PROMPT is that the two ontologies in the mapping (and merging) process should be different versions of the same ontology.

A user makes many of the decisions, and PROMPT either performs additional actions automatically based on the user's choices,or creates a new set of suggestions and identifies additional conflicts among the input ontologies. The tool takes into account different features in the source ontologies to make suggestions for merging two classes such as similar names between the classes, attachement by similar slots, similar restrictions on their facets, and similarity between their super-classes. In addition to providing suggestions to the user, PROMPT identifies conflicts. Some of the conflicts that PROMPT identifies are:

- **name conflicts** (more than one frame with the same name),
- **dangling references** (a frame refers to another frame that does not exist),
- **redundancy** in the class hierarchy (more than one path from a class to a parent other than root),
- **slot-value restrictions** that violate class inheritance.

Figure 3.8 shows a screenshot of PROMPT. The main window (in the background) shows a list of current suggestions in the top left pane and the explanation for the selected suggestion at the bottom. The righthand side of the window shows the evolving merged ontology. The internal screen presents the two source ontologies side-by-side (the superscript m marks the classes that have been merged or moved into the evolving merged ontology).

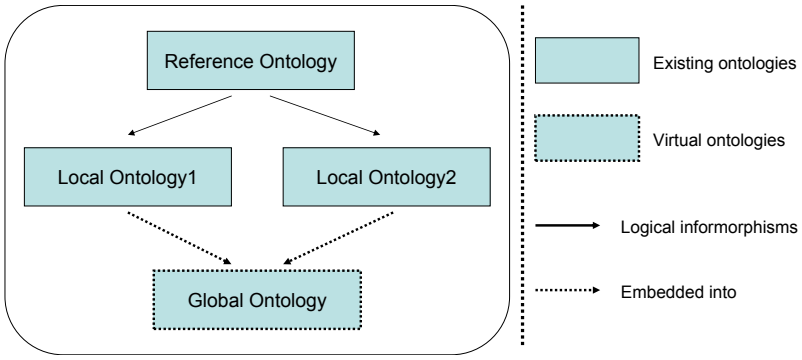

Figure 3.9: The ontology mapping approach of IF-Map

Summarizing, PROMPT gives iterative suggestions for concept merges and changes, based on linguistic and structural knowledge, and it points out to the user the possible effects of these changes. User approval is considered for merging between all kinds of correspondences. A big limitation of PROMPT tools is that the two ontologies in the mapping (and merging) process, should be different versions of the same ontology.

3.3.4 IF-Map

IF-Map (Information Flow-based Mapping) [220] is a channel theory based on ontology mapping technique. Kalfoglou and Schorlemmer developed an automatic method for ontology mapping based on the Barwise- Seligman theory of information flow.

Figure 3.9 illustrates the underpinning framework of IF-Map for establishing mappings between ontologies. The solid rectangular lines surrounding Reference ontology, Local ontology 1 and Local ontology 2 denotes existing ontologies. It is assumed that Local ontologies are used by different communities and populated with instances, while Reference ontology is an agreed understanding that favours the sharing of knowledge, and is not supposed to be populated. The dashed rectangular line surrounding Global ontology denotes an ontology that does not exist yet, but will be constructed by merging. This is similar to Kents virtual ontology of community connections [177].

The architecture of IF-Map is shown in Figure 3.10. The authors built a step-wise process that consists of four major steps: (a) ontology harvesting, (b) translation, (c) infomorphism generation, and (d) display of results. In the ontology harvesting step, ontology acquisition is performed. IF-Map applies a variety of methods such as using existing ontologies from ontology libraries, editing them in ontology editors, and harvesting them from the Web. Then, the format of ontologies is translated into Prolog clauses. The next step in their process is the main mapping mechanism—the IF-Map method. This step finds logic infomorphisms, if any, between the two ontologies under examination and displays them in RDF format.

As a summary, there are two assumptions in IF-Map, which are (1) a common reference ontology for all local ontologies, and (2) considering an equal set of instances for the decision of concept mapping. Kalfoglou and Schorlemmer claim that IF-Map could provide fully automation for a matching process. However, the second assumption is a big restriction for the applicability of IF-Map concerning instantiation heterogeneity.

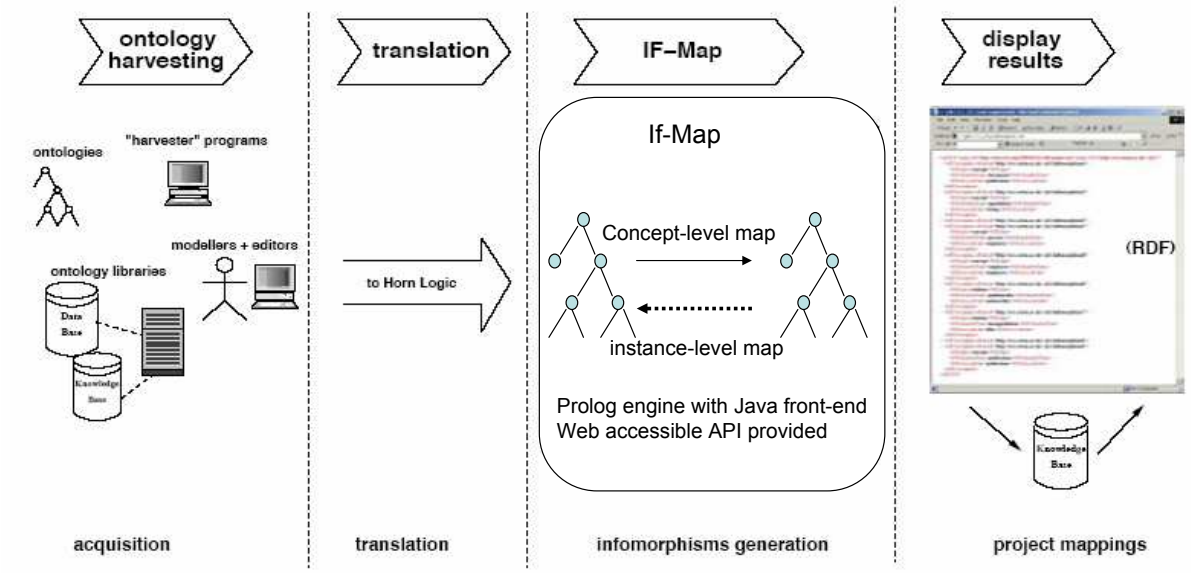

Figure 3.10: The architecture of IF-Map

3.3.5 COMA++

COMA++ (COmbination of MAtching algorithms) [69] extends COMA [68] using a comprehensive infrastructure and a graphical user interface. COMA is a composite schema matching tool that combines different matching algorithms. COMA++ supports higher-level strategies to address complex match problems, in particular fragment-based matching and the reuse of previous match results. Following the divide-and-conquer idea, it decomposes a large match problem into smaller subproblems by matching at the level of schema fragments. The architecture of COMA++ is shown in Figure 3.11. COMA++ encompasses two matching phases: (a) identifying similar fragments, and (b) matching fragments. In the automatic mode, both phases are executed in a single pass using pre-specified strategies. COMA++ also supports step-by-step fragment matching, allowing the user to verify and make changes to the outcome of each phase.

3.3.6 QOM

QOM (Quick Ontology Mapping) [113] is a successor of NOM (Naive Ontology Mapping) [108] which adopts the idea of composite matching from COMA. Some other innovations with respect to COMA, are in the set of elementary matchers based on rules, exploiting explicitly codified knowledge in ontologies, such as information about super- and sub-concepts, super- and sub-properties, etc. At present the system supports 17 rules. For example, rule (R5) states that if superconcepts are the same, the actual concepts are similar to each other. NOM also exploits a set of instance-based techniques. QOM focuses on less run-time complexity for mapping efficiency of large-size, light-weight ontologies. For the purpose of efficiency, the use of some rules has been restricted. The principle idea of QOM's mapping process is as shown in Figure 3.12. QOM avoids the complete pair-wise comparison of trees in favor of a top-down strategy. The approach is based on the idea that the loss of quality in matching algorithms is marginal (compared to a standard baseline), however improvement in efficiency can be tremendous. However,

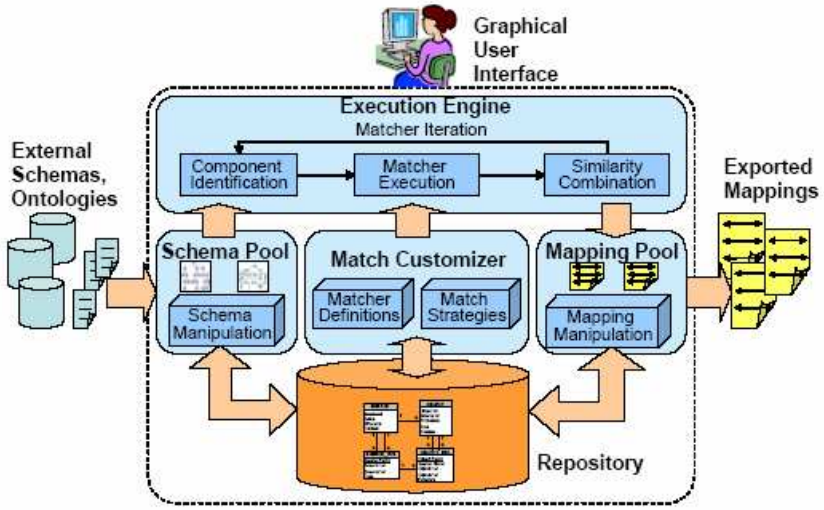

Figure 3.11: The architecture of COMA++ [69]

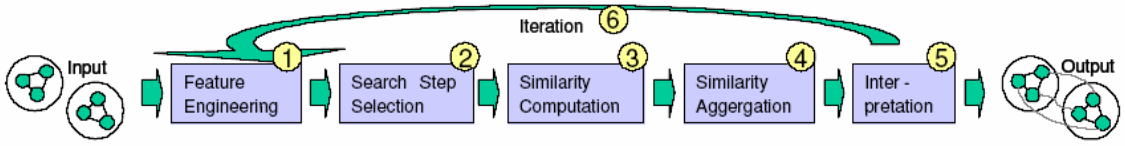

Figure 3.12: Mapping process of QOM

QOM mostly constitutes a straightforward name-based similarity computation on RDFS syntax in order to determine correspondences between entities defined in two ontologies.

3.3.7 GLUE

GLUE [3, 4] is a system that employs a multi-strategy machine learning technique with joint probability distribution to identify similarities of instances, and then compares between concepts and between relations.

The basic architecture of GLUE is shown in Figure 3.13 [4]. It consists of three main modules: Distribution Estimator, Similarity Estimator, and Relaxation Labeler.

- First, the *Distribution Estimator* takes as input two taxonomies O1 and O2, together with their data instances. Then machine learning techniques are applied to compute joint probability distributions for every pair of concepts. GLUE contains two kinds of base learner: *name learner* and *content learner*. Name learner uses linguistic knowledge to calculate similarity between the names of two entities, by exploiting the frequency of words. Each content learner focuses on a certain type of information belonging to instances. Meta-learner is used to linearly combine the predictions of all base learners. The distribution estimator learns on a sample mapping set using a set of base learners and a meta-learner.

- Second, the *similarity estimator* determines the similarity between instances using multiple *base learners* and a *meta learner*. The output from this module is a similarity matrix between the concepts in the two taxonomies.

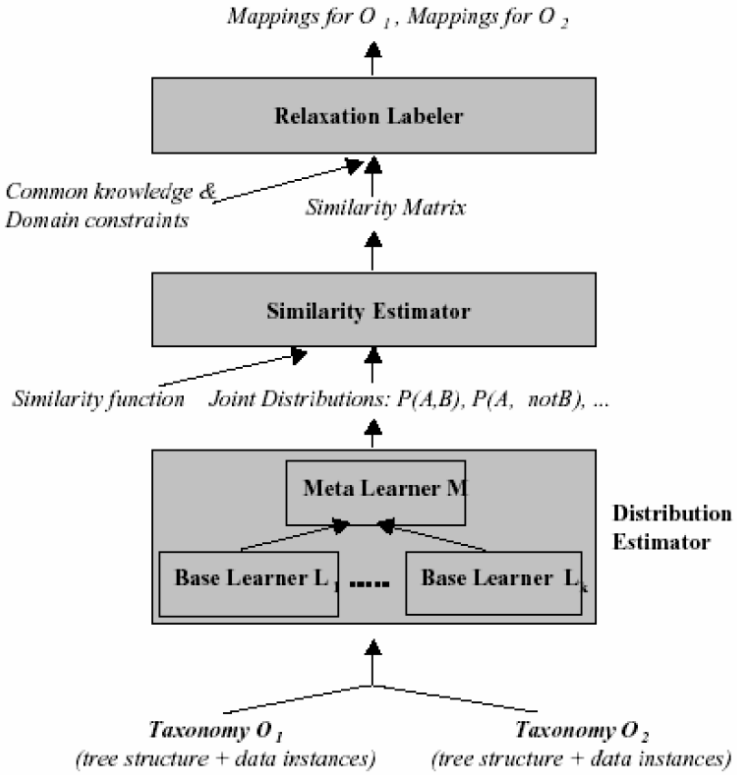

Figure 3.13: The architecture of GLUE

- Finally, *relaxation labeler* determines the best mapping—which best satisfies the given domain constraints and heuristic knowledge—for each entity by analyzing the similarity results of all neighborhood entities. Machine learning supports more automation on the one hand, but the accuracy of matches relies much on training data sets.

3.3.8 Concluding Remarks

According to the above tools, the techniques of similarity analysis used in existing mapping methods can be generally classified into four groups as follows.

- Instance-based similarity analysis (IBSA): The similarity between two concepts is determined by having common instances.
- Lexical-based similarity analysis (LBSA): The similarity between two concepts is decided by analyzing the linguistic meaning of their names.
- Schema-based similarity analysis (SBSA): The similarity between two concepts is found by analyzing similarity between their properties.
- Taxonomy-based similarity analysis (TBSA): The similarity between two concepts is found by analyzing structural relationships between them such as subsumption (generalization/specialization), siblings, etc.

Matching Tools and Methods	IBSA	LBSA	SBSA	TBSA
MAFRA	*	*	*	*
ONION	*	*		
PROMPT		*	*	*
IF-Map	*			
COMA++		*	*	*
QOM		*	*	*
GLUE	*	*	*	*

Table 3.1: Mapping tools (and methods) and their techniques of similarity analysis

I present a general review of each of above mapping tools in Table 3.1 regarding the techniques of similarity analysis. More than one technique may involve in each tool. Each technique is helpful to determine correspondences between two ontologies.

I learned that most matching tools rely much on name-based matching between ontological entities, rather than semantics (or content) defined for each entity, in order to predict possible correspondences. For very complex names, the tools need expert's verification or user's approval. Except in the cases of GLUE and QOM, expert-interaction is highly involved in other mapping techniques. According to the name-based matching methods, the accuracy of matches or unmatches is rather risky. Two concepts might have different names. However, they can have semantic correspondence because the meaning of concept names cannot completely express the semantics of concepts. Moreover, terminological heterogeneity as well as other kinds of heterogeneity may be involved between two concepts. In that case, content-based matching of all available properties and instances will become complex.

Regarding the major aim of my thesis, I have two focuses as described in the following.

- Besides name-based matching, what is an alternative approach to determine the possible semantic correspondences between heterogeneous ontologies.
- Concerning wide-scale semantic heterogeneity, how to reduce complexity in content-based matching.

I will present an enrichment-based matching method regarding these two focuses.

Chapter 4

Semantic Enrichment in Ontologies

A good step in handling semantic heterogeneity is the attempt to enrich the semantics of concepts in ontologies, as it is well understood that the richer knowledge the ontologies possess, the higher probability of accurate and efficient semantically-correspondences will be derived. Thus, I define the meaning of seamntic enrichment as follows:

> **Semantic enrichment** *is a process that renders adequate and precise semantics for domain concepts in the form of structured and consistent taxonomies.*

In general, semantic enrichment for ontology matching involves the usage of alternative knowledge sources together with original ontology. The semantic enrichment techniques are currently based on different theories and a variety of knowledge sources such as linguistic knowledge, fuzzy terminology, intensional or extensional knowledge [187]. Ontology mostly specifies the semantics of concepts using the intensional knowledge—the properties and relations of concepts. The extensional knowledge is used to populate ontologies by defining each concept with a set of individuals from the universe of discourse. The linguistic knowledge especially shared thesauri like WordNet, is used to assist in determining correspondences between domain terms.

However, Mitra and Wiederhold [164] claim that full automation for a mapping by such linguistic knowledge, is not feasible due to inadequency of today's natural language processing technology. There are still different opinions on whether it is intension or extension that best decides the exact context of a concept [218]. It is obvious that the semantics of similar concepts described by either intensional knowledge or extensional knowledge, in two different ontologies, can be heterogeneous because of different specification and conceptualization on diverse knowledge. Therefore, my enrichment approach is based on defining concept-level properties according to a classification of ontological concepts through some philosophical foundations.

In this chapter, I first provide a First-order Quantified Modal Language $\mathcal{L}^E$ in order to express the precise semantics of philosophical notions, as well as my enriched-model of ontologies. Second, I discuss some philosophical foundations for ontological conceptualization. Third, I present my approach of modeling semantically-enriched ontologies. Fourth, for the development of such enriched ontologies by users, I provide a sortal meta-class ontology (named `sort.owl`) as an open source interface. Thus, a design and implementation of the sortal meta-class ontology is presented. Finally, the development of semantically-enriched ontologies using `sort.owl` is set out and demonstrated through Protégé OWL API.

4.1 A First-order Quantified Modal Language $\mathcal{L}^E$

In order to deal with semantic heterogeneity, I adopt three philosophical notions of OntoClean[1] [147]. These notions are identity, rigidity, and dependency. *Identity* is the logical relation of sameness, in which an individual identifies only to itself globally. *Rigidity* supports the essentiality of a concept to its individuals. By Lowe [48], "a concept is essential for its individuals" means every individual of a concept is always an individual of the concept. *Dependency* states an externally dependent relation between concepts. I apply these notions as foundational knowledge to cope with heterogeneity in ontological conceptualization. In this section, I provide a First-order Quantified Modal Language $\mathcal{L}^E$ as a formal logic language to represent my theory of semantic enrichment.

Quantified Modal Logic (QML) is a philosophical logic that develops formal systems and structures for the analysis of ontological concepts using philosophical notions such as essence, existence, actualism, individualism, possibilism, identity, part-whole, dependency, etc [101]. Modal Logic is the logic of *necessity* (or 'must be') and *possibility* (or 'may be'). QML comprises modal logic and first-order predicate logic grammatically, axiomatically, and semantically. In QML, there are some philosophical issues such as actualism/possibilism[2], realism about possible worlds[3], and trans-world identity of individuals[4]. These subject matters relate to the fields of ontology, epistemology, philosophy of science, ethics, etc. QML has first appeared in papers by Barcan Marcus [170, 171, 172], Hintikka [90, 91], Prior [20, 21, 22], and Kripke [189, 190, 191, 192].

There are three circumstances for this research to employ QML.

- The basic influence is Guarino & Welty mentioned that the notions of OntoClean were formalized in $S5$[5] QML with the Barcan formula (BF)[6], which gives us a *constant domain* (every object exists in every possible world) and *universal accessibility* (every world is accessible from every other world) [35].

- The second condition is a claim of varying domains among possible worlds, because in practice, we cannot expect the same set of individuals actually exists in each arbitrary accessible world. Consequently, there is a need of actual existence, as opposed to logical existence, that indicates some objects actually exist in the possible worlds [27, 114]. For example, "God exists" is a kind of logical existence, however "Mars exists" is an actual existence because we can prove that a planet called Mars actually exists in our universe.

- The third condition is that QML amounts to *trans-world identity*—the identity of two incomplete descriptions of an individual can be detected across multiple possible worlds.

[1] OntoClean is a domain-independent methodology for ontological analysis—a framework for cleaning taxonomic structure of ontologies.

[2] The fundamental thesis of actualism is "Everything that exists is actual". Possibilism is the denial of this thesis with a claim of possible but non-actual individuals [33].

[3] Modal realism is the view, notably propounded by David Lewis, that possible worlds are as real as the actual world [34].

[4] A trans-world identity is an identity that holds across possible worlds [115].

[5] $S5$ is a system where accessibility relation R is reflexive: $\Box\phi \rightarrow \phi$, symmetric: $\phi \rightarrow \Box\Diamond\phi$, and transitive: $\Box\phi \rightarrow \Box\Box\phi$.

[6] $\forall x \Box\phi \rightarrow \Box\forall x\phi$ [Barcan formula]

4.1.1 Syntax

A first-order QML is a group of logical axioms and rules of inference that systematizes the logically true sentences of a standard first-order modal language with identity. Thus, it is the sum of classical propositional logic, classical first-order quantification theory, the logic of identity, and modal theory [114, 53, 59]. Here, I provide a formal language of QML, that concerns possible worlds which have different (or varying) actual domains in a given universe of both individuals and datatype values. The language amounts necessity and possibility with a special concern of actual existence.

Let $\mathcal{L}^E$ be a first-order modal language which contains a special predicate symbol for the actual existence of individuals.

Definition 1 (Alphabet) *The alphabet of language* $\mathcal{L}^E$ *is* $\mathcal{A}^E = \{\mathcal{X}, \mathcal{C}, \mathcal{P}_n, \mathcal{F}_n, E, =\}$ *where*

- $\mathcal{X}$ *is a countable infinite set of individual variables* $x, y, z, \cdots$;
- $\mathcal{C}$ *is a countable infinite set of individual constants* $c_1, c_2, \cdots$;
- $\mathcal{P}_n$ *is a countable infinite set of n-ary predicate symbols* $p_1, p_2, \cdots$;
- $\mathcal{F}_n$ *is a countable infinite set of n-ary function symbols* $f_1, f_2, \cdots$;
- E *is a predicate symbol to describe the actual existence of individuals, and*
- $=$ *is identity symbol.*

The following connectives, quantifiers, and modal operators, will also be used in $\mathcal{L}^E$ according to the usual way of first-order predicate logic and modal logic [59].

- propositional connectives: not ($\neg$), and ($\wedge$), or ($\vee$), implies ($\rightarrow$), and equivalence ($\leftrightarrow$)
- quantifiers: universal quantifier ($\forall$) and existential quantifier ($\exists$)
- modal operators: necessity ($\Box$) and possibility ($\Diamond$)
- falsehood: $\bot$

Terms of $\mathcal{L}^E$, $t_1, t_2, \cdots$, are either constants, variables, or n-ary functions $f_n(t_1, ..., t_n) \in \mathcal{F}_n$.

Definition 2 (Atomic Formula) *If* p_n *is an n-ary predicate symbol and* $\langle t_1, ..., t_n \rangle$ *is an n-tuple of terms, then* $p_n(t_1, ..., t_n)$ *is an atomic formula.*

Definition 3 (Atomic Identity Formula) *If* t_1 *and* t_2 *are any terms, then* $t_1 = t_2$ *is an atomic identity formula, that is,* t_1 *is identical to* t_2.

Definition 4 (Atomic Actual Existence Formula) *If* t *is any term, then* $E(t)$ *is an atomic actual existence formula, that is, an individual belongs to term* t *actually exists in a possible world.*

Definition 5 (Formulas of $\mathcal{L}^E$) *A set of formulas $\phi, \psi, \cdots$ in alphabet $\mathcal{A}^E$ (called $\Delta_{\mathcal{A}^E}$) is defined as follows:*

- *All atomic formulas are formulas of $\Delta_{\mathcal{A}^E}$.*
- *If ϕ is a formula, then so is $\neg\phi$.*
- *If ϕ,ψ are formulas, then so is $\phi \rightarrow \psi$.*
- *If ϕ is a formula and x is a variable, then $\forall x\phi$ is a formula of $\Delta_{\mathcal{A}^E}$.*
- *If ϕ is a formula, then so is $\Box\phi$.*

Complex formulas are constructed using connectives. The definitions of some complex formulas are given below.

- $\phi \wedge \psi =_{def} \neg(\phi \rightarrow \neg\psi)$
- $\phi \vee \psi =_{def} \neg\phi \rightarrow \psi$
- $\phi \leftrightarrow \psi =_{def} (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$
- $\exists x\phi =_{def} \neg\forall x\neg\phi$
- $\Diamond\phi =_{def} \neg\Box\neg\phi$

4.1.2 Semantics

I give the semantics of languages $\mathcal{L}^E$ in terms of Kripke semantics. Kripke semantics is a formal semantics for non-classical logic systems, created in late 1950 and early 1960 by Saul Kripke. It was originally developed for modal logics, but it was subsequently adapted to intuitionistic logic and some other non-classical systems. The discovery of Kripke semantics was a major breakthrough in the development of non-classical logics, as the model theory of such logics was virtually nonexistent before Kripke.

Definition 6 (Kripke frame) *A kripke frame in QML is $F = \langle W, R\rangle$ where W is a non-empty set, and R is a binary relation on W. Set W is intuitively interpreted as the domain of possible worlds, whereas R is the accessibility relation between worlds.*

Since this logic system is intended to formalize ontological components due to the specification of OWL properties: object properties (`owl:ObjectProperty`) and datatype properties (`owl:DatatypeProperty`)[7], I consider both individuals and datatype values, in a universe.

Definition 7 (Universe) *Universe U includes a set of individuals U_{ind} and a set of datatype values U_{dtp}, such that $U = U_{ind} \cup U_{dtp}$.*

U_{dtp} is similar to the concrete domain of Description Logic language $SHIQ(D)$[8]. The **interpretation** function of language $\mathcal{L}^E$ is defined as follows.

[7] owl:ObjectProperty relates two individuals, but owl:DatatypeProperty relates an individual and a datatype value (see `http://www.w3.org/TR/owl-guide/`).

[8] $SHIQ(D)$ [84] is an extension of $SHIQ$ with a concrete domain with datatype values. $SHIQ$ is an extension of the well known DL ALC [52] to include transitively closed primitive roles [82].

Definition 8 (Interpretation) *A tuple $\mathcal{I} = \langle U, \cdot^I \rangle$ is the interpretation function of language $\mathcal{L}^E$ in Kripke frame F, where U is a non-empty universe and $.^I$ is a mapping function from a symbol of alphabet $\mathcal{A}^E$ to the members of U.*

The key insight in Kripke's QML is the replacement of the single domain of individuals in the interpretation of a first-order modal language with a function that assigns to each world its own distinct domain of individuals. Thus, instead of a single domain common to all worlds, domains are permitted to vary from world to world. Interpretations like this for first-order modal languages in which each world has its own domain are known as Kripke models. Regarding an $S5$ Kripke frame, domains of possible worlds become constant. However, if we regard actual existence of individuals and no individuals actually exists forever in a world, for example, a person can die in someday, then each world has two nested domains: a possible domain and an actual domain. Therefore, I define such nested domains in each possible world, that is constant outer domain and varying inner domain, by distinguishing the actual domain of a possible world from its possible domain by using existential predicate E.

Definition 9 (Kripke Model) *A Kripke model given in universe U, is a quintuple $M = \langle F, \Vdash, D, d, \mathcal{I} \rangle$ where*

- *F is a Kripke frame,*
- *$\Vdash$ is a satisfaction relation between members of $w \in W$ and formulas of $\Delta_{\mathcal{A}^E}$,*
- *D is a function that assigns a non-empty outer domain $D(w) = U$ to every $w \in W$,*
- *d is a function that assigns a non-empty inner domain to every $w \in W$ such that $d(w) \subseteq D(w)$, and*
- *$\mathcal{I} = \langle U, \cdot^I \rangle$ is the interpretation in frame F such that*
  - *$\mathcal{I}(p_n^I, w) \subseteq D(w)^n$ for any n-ary predicate $p_n \in \mathcal{P}_n$,*
  - *$\mathcal{I}(f_n^I, w) : D(w)^n \to D(w)$ for any n-ary function $f_n \in \mathcal{F}_n$,*
  - *$\mathcal{I}(c^I, w) \in D(w)$ for any individual constant $c \in \mathcal{C}$, and*
  - *$\mathcal{I}(E^I, w) = d(w)$ for existential predicate E.*

Each *outer domain* $D(w)$ contains the objects which it makes sense to talk about the possible domain of w, on the other hand in each *inner domain* $d(w)$ there appear individuals actually existing in w.

I assume that model M satisfies the *inclusion requirement* [114], that is, if wRw' then $D(w) \subseteq D(w')$. As frame F employs $S5$ there is a constant outer domain between possible worlds such that $D(w) = D(w')$. In practice, we cannot expect that the same individuals actually exist in each arbitrary accessible world. Therefore, I regard that the inner domain of each world varies, depending on the actual existence of individuals in the world.

Definition 10 (w-assignment) *To define truth conditions for atomic and quantified formulas with variables $x \in X$ given in $\mathcal{L}^E$, w-**assignment function** ∂ into interpretation $\mathcal{I}$ in world w is defined as $\mathcal{I}^\partial(x^I, w) = \partial(x, w)$. There is also a variant of w-assignment, $\partial^{x,a}$, which assigns individual element $a \in D(w)$ to x.*

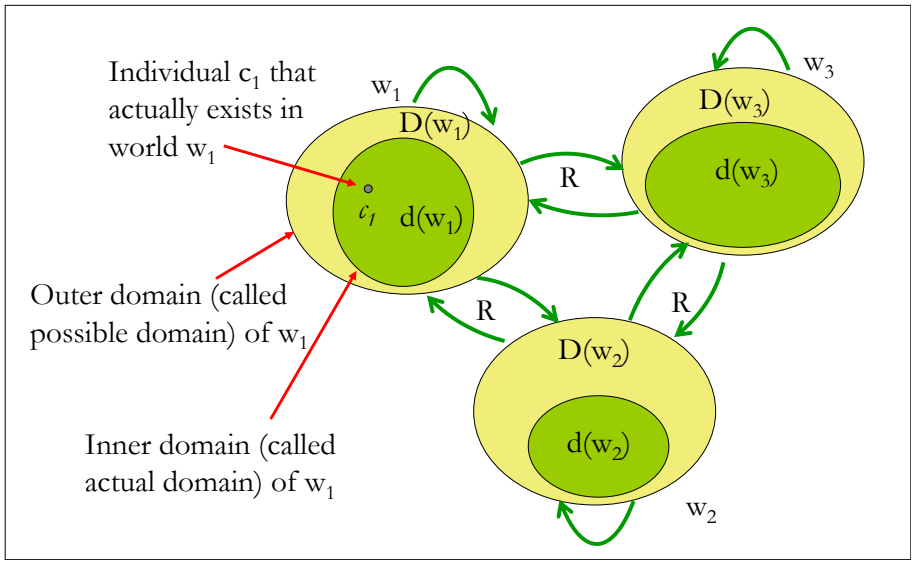

Figure 4.1: The outer and inner domains of possible worlds in an $S5$ Kripke model

Definition 11 (Satisfaction) *For any world $w \in W$ given in Kripke model M, the satisfaction relation of the formulas of $\Delta_{\mathcal{A}^E}$ with respect to $\mathcal{I}^\partial$ is as follows:*

- $(\mathcal{I}^\partial, w) \Vdash p_n(t_1, ..., t_n)$ *iff* $\langle \mathcal{I}^\partial(t_1^I, w), ..., \mathcal{I}^\partial(t_n^I, w)\rangle \in \mathcal{I}^\partial(p_n^I, w)$
- $(\mathcal{I}^\partial, w) \Vdash t_1 = t_2$ *iff* $\mathcal{I}^\partial(t_1^I, w) = \mathcal{I}^\partial(t_2^I, w)$
- $(\mathcal{I}^\partial, w) \Vdash \neg\phi$ *iff* $(\mathcal{I}^\partial, w) \nVdash \phi$
- $(\mathcal{I}^\partial, w) \Vdash \phi \rightarrow \psi$ *iff* $(\mathcal{I}^\partial, w) \Vdash \neg\phi$ *or* $(\mathcal{I}^\partial, w) \Vdash \psi$
- $(\mathcal{I}^\partial, w) \Vdash \Box\phi$ *iff for every* $w' \in W$ *such that* wRw', $(\mathcal{I}^\partial, w') \Vdash \phi$
- $(\mathcal{I}^\partial, w) \Vdash \forall x\phi$ *iff for every individual* $a \in D(w)$, $(\mathcal{I}^{\partial^{x,a}}, w) \Vdash \phi$
- $(\mathcal{I}^\partial, w) \Vdash E(t)$ *iff* $\mathcal{I}^\partial(t^I, w) \subseteq d(w)$

In particular $(\mathcal{I}^\partial, w) \Vdash \bot$ never holds. A formula is true in Kripke model M if and only if it is true in every possible world $w \in W$ of M. Similarly, a formula is *valid* (denoted by $\vdash$) in Kripke frame F if and only if it is true in every Kripke model M given on F. For obtaining an $S5$ frame F, the following axioms must be satisfied in any Kripke model M given in the frame.

- K: $\Box(\phi \rightarrow \psi) \rightarrow (\Box\phi \rightarrow \Box\psi)$
- T: $\Box\phi \rightarrow \phi$ [reflexive]
- 4: $\Box\phi \rightarrow \Box\Box\phi$ [transitive]
- 5: $\phi \rightarrow \Box\Diamond\phi$ [symmetric]

Figure 4.1 illustrates the state of outer and inner domains of possible worlds given in an $S5$ Kripke model M. In an $S5$ Kripke model, every world is accessible not only to itself, but also to any other possible worlds. This is called universal accessibility, and by that every possible world has a constant outer domain whilst the inner domains differ from each other according to the actual existence of individuals in the worlds.

Following to Menzel [33], a proof of Barcan Formula (BF) in $S5$ can be driven through the following steps:

- **Proof(I):** $\phi \rightarrow \Box\Diamond\phi$
  (1) $\Box\neg\phi \rightarrow \neg\phi$ [instance of axiom T]
  (2) $\phi \rightarrow \neg\Box\neg\phi$ [from (1) by contraposition]
  (3) $\phi \rightarrow \Diamond\phi$ [from (2) by definition of $\Diamond$]
  (4) $\Diamond\phi \rightarrow \Box\Diamond\phi$ [instance of axiom 5]
  (5) $\phi \rightarrow \Box\Diamond\phi$ [from (3) and (4) by propositional logic]

- **Proof(II):** $\Diamond\Box\phi \rightarrow \phi$
  (1) $\neg\phi \rightarrow \Box\Diamond\neg\phi$ [instance of Proof(I)]
  (2) $\Box\Diamond\neg\phi \leftrightarrow \neg\Diamond\Box\phi$ [instance of modal negation principle]
  (3) $\neg\phi \rightarrow \neg\Diamond\Box\phi$ [from (1) and (2) by propositional logic]
  (4) $\Diamond\Box\phi \rightarrow \phi$ [from (3) by contraposition]

- **Proof of Rule1: if** $\vdash \phi \rightarrow \psi$**, then** $\Box\phi \rightarrow \Box\psi$
  (1) $\phi \rightarrow \psi$ [assume as theorem]
  (2) $\Box(\phi \rightarrow \psi)$ [from (1) by rule of necessitation: $\phi \rightarrow \Box\phi$]
  (3) $\Box(\phi \rightarrow \psi) \rightarrow (\Box\phi \rightarrow \Box\psi)$ [instance of axiom K]
  (4) $\Box\phi \rightarrow \Box\psi$ [from (2) and (3) by Modus Ponen (MP): $(\phi \wedge \phi \rightarrow \psi) \rightarrow \psi$]

- **Proof of Rule2: if** $\vdash \Diamond\phi \rightarrow \psi$**, then** $\phi \rightarrow \Box\psi$
  (1) $\Diamond\phi \rightarrow \psi$ [assume as theorem]
  (2) $\Box\Diamond\phi \rightarrow \Box\psi$ [by Rule1]
  (3) $\phi \rightarrow \Box\Diamond\phi$ [instance of Proof(II)]
  (4) $\phi \rightarrow \Box\psi$ [from (2) and (3) by propositional logic]

- **Proof of Barcan Formula (BF):** $\forall x\Box\phi \rightarrow \Box\forall x\phi$
  (1) $\forall x\Box\phi \rightarrow \Box\phi$ [By quantifier axiom]
  (2) $\Box[\forall x\Box\phi \rightarrow \Box\phi]$ [from (1) by rule of necessitation $\phi \rightarrow \Box\phi$]
  (3) $\Box(\phi \rightarrow \psi) \rightarrow (\Box\phi \rightarrow \Box\psi)$ [by K]
  (4) $\Diamond\forall x\Box\phi \rightarrow \Diamond\Box\phi$ [from (2) given (3)]
  (5) $\Diamond\Box\phi \rightarrow \phi$ [instance of $\Diamond\Box\phi \rightarrow \phi$]
  (6) $\Box\forall x\Box\phi \rightarrow \phi$ [from (4) and (5) by propositional logic]
  (7) $\forall x[\Diamond\forall x\Box\phi \rightarrow \phi]$ [from (6) by generalization]
  (8) $\forall x\Box\phi \rightarrow \forall x\phi$ [from (7) and quantifier axiom by MP]
  (9) $\forall x\Box\phi \rightarrow \Box\forall x\phi$ [from (8) by Rule2]

Since language $\mathcal{L}^E$ contains existence predicate E, the following two axioms are applied [114, 53].

- $\forall x[\phi \rightarrow (E(y) \rightarrow \phi[x/y])]$ [E-exemplification][9]

[9]For a proof for $(\mathcal{I}^\partial, w) \Vdash \phi[x/y]$ iff $(\mathcal{I}^{\partial^{x,\partial(y)}}, w) \models \phi$, I refer to [53].

- $\frac{\phi \rightarrow (E(x) \rightarrow \psi)}{\phi \rightarrow \forall x \psi}$ where x is not free in ϕ [Universal E-instantiation]

A logic formulated in a given language is said to be *sound* with respect to the given semantics, if and only if every theorem of the language is valid relative to that semantics (i.e., is true in every interpretation or model of the semantics). Also, the logic is said to be *complete* if and only if every valid formula of the language is a theorem of the logic.

The soundness and completeness of QML for $S5+BF$, have been proved by Corsi and Belardinelli. Thus, I refer to [58, 53] for the proofs of the soundness and completeness.

4.2 Philosophical Foundations for Ontological Conceptualization

Conceptual Modeling is a fundamental discipline in computer science, playing an essential role in areas such as database and information systems design, software and domain engineering, design of knowledge-based and intelligent systems, requirements engineering, information integration, semantic interoperability, natural language processing, and enterprise modeling.

In the philosophical sense, ontology is the study of existence and models of existence. In computer science, ontology is the study of what exists in a given domain. *Domain ontologies* are used to refer to specific theories about material domains such as law, medicine, archeology, molecular biology, etc. Abstractions of a given portion of reality are constructed in terms of concepts. I name modeling a set of concepts used to articulate abstractions of the state of affairs in a given domain as *ontological conceptualization*. Therefore, a domain ontology is a kind of conceptual specification and, hence, ontology modeling is a specific type of conceptual modeling.

Having a precise representation of a given conceptualization becomes even more critical when we want to integrate different independently-developed models (or systems based on those models). In order for these systems to function properly together, we must guarantee that they ascribe compatible meanings to real world entities of their shared subject domain. The ability of systems to interoperate, with compatible real-world semantics is known as *semantic interoperatibility* [126]. In order to support semantic interoperability through heterogeneous ontologies, I apply some philosophical notions to cope the semantic heterogeneity of ontological conceptualization.

In this section, I present a philosophical theory for defining ontological distinctions on the category of conceptual modeling, as well as constraints on the construction of taxonomic structures using these distinctions. By using a number of formally defined philosophical notions, we can apply ontological classification to enrich the semantics of ontological concepts.

4.2.1 Why Fundamental Ontological Classes are Sorts?

In the philosophical literature, ontological concepts are generally divided into two categories: sortal concepts (called sorts) and non-sortal concepts.

"**Sort** is an entity type[10] that carries a criteria for determining the individu-

[10]Entity type has an extension (instances) and an intension which includes an applicability criteria for determining whether an entity is an instance of it.

ation[11], persistence, and identity[12] of its instances (Corazzon, 1729)[13]".

"No entity without identity" (Quine, 1969) [216].

"A class is called a **sort** if it supplies or carries an Identity Condition (IC); otherwise non-sortals" (Guarino & Welty, 2001) [147].

According to the above-quoted statements, it is significant that the principles of *identity* and *individuation* supplied by sorts are essential in conceptual modeling, together with a universe of discourse. Therefore, Guizzardi and Wagner [60] made the following postulate.

"Every object in a conceptual model (CM) of a domain must be an instance of a CM-class representing a sortal" (Guizzardi et al., 2004) [60].

In this research, I follow the above postulate and treat fundamental ontological classes as sorts, and non-sortals as the attributes values or properties of sorts. Some examples of sorts are `Person`, `Planet`, `Dog`, `House`, `Student`, `Wine`, `Book`, and `Car`, where individuals (or instances) are countable and identifiable. Unlike sorts, `Red`, `Happy`, and `Beautiful`, are non-sortals, which do not supply identity for their individuals. The ontological difference between sort and non-sort, for example `Wine` and `Red`, is that `Wine` corresponds to a *natural kind* (or type[14]) whereas `Red` corresponds to an *attribution*. Whilst the former applies necessarily to its individuals (a wine cannot cease to be a wine without ceasing to exist), the latter only applies contingently. Moreover, whilst the former supplies a principle of identity for its individuals such as wine name including appellation, winery, and vintage year, the latter cannot supply such characteristics.

Again, let us consider `Person` and `Happy`. Every individual person has a specific fingerprint, by which characteristic we can identify each individual person from others. While *Person* is a sort, *Happy* is a non-sortal in that all happy people are not identifiable via the characteristic of being happy. When *HappyPerson* is a sort, there is a property *hasEmotion* which is restricted by attribute value "happy". Thus, in this work, the sorts are considered as classes and non-sortals are treated as attributive properties of the sorts.

However, whether a concept is a sort or not, should rely on the possession of identity criteria rather than the common sense of a concept's name. Also, I summarize that *individuals* of a sort are countable as well as identifiable. And, any abstract class of such individuals can be represented as a sort.

4.2.2 Sortal Taxonomy and Subsumption Relationship

Taxonomies are a central part of ontologies. Subsumption relationship is mainly used to organize sorts in taxonomies. This is also known as 'is-a' or 'class inclusion'. To represent sorts, I use unary predicates of language $\mathcal{L}^E$, by adding predicated names corresponding to sort names, such as $p_s \in \mathcal{P}_n$. Then, the fundamental semantics of a subsumption relationship $\sqsubseteq$ between two sorts s_1 and s_2, can be interpreted as an implication relation [32].

[11] An individual means an entity which is countable as a whole [146].

[12] An identity criteria (also called identity Condition) supports the judgment of whether two particulars describe the same entity or not.

[13] `http://www.formalontology.it/index.htm`

[14] A type is a category of thing, e.g., a person is a type of living being.

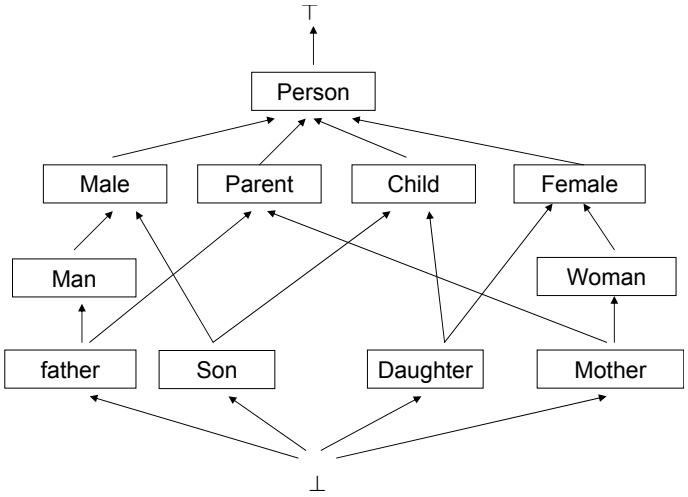

Figure 4.2: A sortal taxonomy of genealogy domain

Definition 12 (Subsumption relationship) *For two sorts s_1 and s_2,*

$$s_2 \sqsubseteq s_1 \quad \textit{iff} \quad \forall x[p_{s_2}(x) \rightarrow p_{s_1}(x)].$$

This is read as "if sort s_2 is subsumed by sort s_1 then every individual of s_2 is an individual of s_1, and vice versa". Then, s_1 is called *super-sort* and s_2 is called *sub-sort*.

Example 1 *Student $\sqsubseteq$ Person means every student is a person. In this case,* `Person` *is a super-sort and* `Student` *is a sub-sort.*

Definition 13 (Sortal Taxonomy) *A hierarchy of sorts with subsumption relationships is called a sortal taxonomy, denoted by $\langle S, \sqsubseteq\!\!\!\sqsubseteq \rangle$ where S is a set of sorts and $\sqsubseteq\!\!\!\sqsubseteq$ is a collection of subsumption relationships between any two sorts of S.*

I consider sorts in a lattice[15] with the topmost sort $\top$ and a bottommost sort $\bot$. Generally speaking, because some sorts may not have a proper meet, the whole set S of sorts is not a lattice; however, if I assume that every leaf is connected to the lowest sort $\bot$ then it becomes a lattice. $\sqsubseteq\!\!\!\sqsubseteq$ is reflexive[16], transitive[17] and anti-symmetric[18] For any two sorts s_1 and s_2, if $s_1 \sqcap s_2 = \bot$ then I say that s_1 and s_2 are *disjoint* and I abbreviate this as $s_1 \asymp s_2$. Figure 4.2 illustrates a sortal taxonomy of genealogy domain, where every male is a person, every man is a male, and every father is a man as well as a parent. Similarly, every female is a person and every woman is a female. `Child` is a join of *Son* and *Daughter* ($Son \sqcup daughter$), and `Mother` is a meet of $Woman$ and $Parent$ ($Woman \sqcap Parent$).

[15]A lattice is a partially ordered set in which any two sorts have join and meet. Join denoted by $\sqcup$ is the least upper bound of two sorts and meet denoted by $\sqcap$ is the greatest lower bound of two sorts in a lattice.

[16]For any s, $s \sqsubseteq s$.

[17]For any s_1, s_2 and $s_3 \in S$, if $s_1 \sqsubseteq s_2$ and $s_2 \sqsubseteq s_3$, then $s_1 \sqsubseteq s_3$.

[18]For any s_1 and s_2, if $s_1 \sqsubseteq s_2$ then $s_2 \not\sqsubseteq s_1$.

4.2.3 Identity and Identity Condition

Identity is one of the most fundamental notions in ontology. Identity states a logical relation of sameness, in which an individual identifies only to itself. Precisely speaking, identity is related to the problem of distinguishing a specific instance of a certain class from other instances by means of a characteristic property, which is unique to it. A typical example used in OntoClean, to explain identity is "is that my dog?"; see [46] for an account of identity problems of ordinary objects, and [75] for a collection of philosophical papers in this area. Logically speaking, identity is a primitive equivalence relation, with the peculiar property of allowing the substitution of terms within logical formulas (Leibniz's rule[19]). Based on the idea that every individual is what it is and not anything else, identity is useful in distinguishing a specific individual from other individuals by means of an *identity condition* (IC)—a property that provides a unique IC value to each individual.

Before discussing the formal semantics of IC, some clarifications about its intuitive meaning may be useful. If I say "Two persons are the same if they have the same fingerprint", I seem to create a puzzle: how can they be two if they are the same person? The puzzle can be solved by recognizing that two incomplete descriptions of a person can be different, even while referring to the same person. The statement "two individuals are the same" can be therefore rephrased as "two descriptions refer to the same individual". A description can be seen as a set of properties that apply to a certain individual. My intuition is that two incomplete descriptions refer to the same individual if they have a common IC.

In the philosophical literature, an identity condition for an arbitrary concept p_ϕ[20] is usually defined as a suitable relation ρ satisfying the following formula [147]:

$$p_\phi(x) \wedge p_\phi(y) \rightarrow (\rho(x,y) \leftrightarrow x = y). \tag{4.1}$$

The nature of ρ in Equation 4.1 is based on the characteristic relation of a certain concept, which is unique for a specific individual. That characteristic relation must be definable for each individual of the concept and must satisfy Equation 4.1 [147]. In point of fact, identity criteria are conditions used to determine equality and that are entailed by equality.

Guarino & Welty think of IC as *diachronic IC*—a stable IC at different time points [147]. Moreover, they consider the actual existence of individuals in the possible world semantics. Thus, they revised Equation 4.1 as follows:

$$\Box(E(x,t) \wedge p_\phi(x,t) \wedge E(y,t') \wedge p_\phi(y,t') \wedge x = y \rightarrow \Sigma(x,y,t,t')\ ) \tag{4.2}$$

$$\Box(E(x,t) \wedge p_\phi(x,t) \wedge E(y,t') \wedge p_\phi(y,t') \wedge \Sigma(x,y,t,t') \rightarrow x = y\ ) \tag{4.3}$$

where the predicate E is for actual existance, t and t' are time parameters, and Σ is the sameness formula (for example, fingerprint(x,t)=fingerprint(y,t') for a diachronic identity between two individual persons). An IC is *necessary* if it satisfies 4.2 and *sufficient* if it satisfies 4.3. In OntoClean, Guarino & Welty used a non-modal time parameter 't' to mention a time line (called branching time) in each possible world in order to distinguish diachronic IC (we may call it global IC) from synchronic IC (IC at a single point in time

[19] $(x = y) \rightarrow (p_\phi \rightarrow p_\phi[x/y])$ where p_ϕ denotes a first-order unary predicate for ontological property ϕ.

[20] In OntoClean, Guarino & Welty consider an ontological concept as a *property* similar to the meanings (or intensions) of expressions like *being an apple* or *being a table*, which correspond to unary predicates in First-order Logic. Here, I use a predicate symbol p_ϕ instead of ϕ from the original equations.

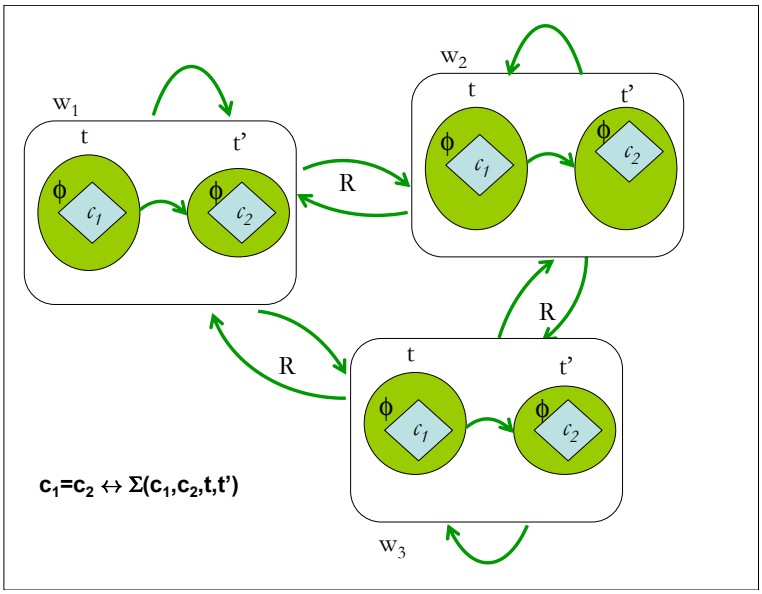

Figure 4.3: An illustration of diachronic IC by Equation 4.2 and 4.3

or called local IC). I illustrate the idea of Equation 4.2 and 4.3 in a $S5$ Kripke model where $W = \{w_1, w_2, w_3\}$, as shown in Figure 4.3. If an IC provides the same IC value for each individual not only inside a possible world but also across possible worlds, then the IC is a diachronic IC; otherwise it is a synchronic IC. Basically, the definition of IC obeys Leibniz's rule with no exceptions and by holding time-invarient identity. Also regarding trans-world identity, two individuals are identical if they always have a sameness relation with a common IC value across possible worlds. I redefine Equations 4.2 and 4.3 with respect to the following points.

- Every state of possible affairs in time, space, or any possibility, is interpreted as possible worlds in modalities instead of considering a time line inside possible worlds.
- IC is assumed as a property of sorts because only sortal concepts carry or supply ICs.
- IC is considered as a unary function of language $\mathcal{L}^E$ in order to represent it as an OWL datatype property (`owl:DatatypeProperty`). And IC values are considered as datatype values.
- The sameness formula Σ is reformulated in terms of equality between the IC values of two common individuals of a sort, and the identity of an individual is entailed by this.

Note that if we consider temporal aspect for some tense-intensive domains, it is necessary to fuse temporal logic[21] into language $\mathcal{L}^E$.

[21]Temporal logic (also called tense logic) is used to describe any modal logic-based system of rules and symbolism for representing, and reasoning about, propositions qualified in terms of time.

Definition 14 (Identity Condition) *Identity Condition (IC) of a sort is a datatype property, which provides a unique IC value to each individual of the sort. Formally, if ι is an IC of sort s (denoted by p_s), then it satisfies one of the following conditions.*

$$\Box\forall x, y[p_s(x) \wedge E(x) \wedge p_s(y) \wedge E(y) \wedge x = y \rightarrow \iota(x) = \iota(y)] \tag{4.4}$$

$$\Box\forall x, y[p_s(x) \wedge E(x) \wedge p_s(y) \wedge E(y) \wedge \iota(x) = \iota(y) \rightarrow x = y] \tag{4.5}$$

Equation (4.4) states that "The IC of a sort must *necessarily* provide the same IC value for the same individual of the sort". Equation (4.5) states that "The IC of a sort must be necessarily *sufficient* to recognize two individuals which both actually exist and own the same IC value as the same individual".

Example 2 *Suppose that hasISBN is the IC of sort* `PublishedBook`*. So that, it is necessary to have the same ISBN*[22] *for the same published book, or two individual books with the same ISBN can be identified as the same published book in every possible world. Someone may use a global product bar-code to identify each copy of the same Published-Book (say an individual of PublishedBookCopy). For other examples, hasFingerprint, hasURI, and hasLatitudeLongitude can be used as the ICs of* `Person`*,* `WebResource`*, and* `Location`*, respectively.*

Note that ICs should be globally identifiable for individuals. For example, `Student` possesses property 'hasStudentID', however it is world-variant and cannot be used as an IC. I call it *local IC*, and use it to identify individuals only inside a possible world.

Example 3 *Suppose that $studentID$ gives a unique identification number to each student in both University 1 (or U_1) and University 2 (or U_2).*

$$\begin{cases} In\ U_1, hasStudentID(c_1\colon Student) = \text{'320025'} \\ In\ U_2, hasStudentID(c_1\colon Student) = \text{'210'} \end{cases}$$

However, the *studentID* value of student c_1 in U_1 is different from U_2. By $Student \sqsubseteq Person$ with $\iota_{Person} = hasFingerprint$, the fingerprint of c_1 should be the same in both U_1 and U_2.

$$\begin{cases} In\ U_1, \text{hasFingerprint}(c_1\colon Student) = 8A08\ 617D\ 9FC3\ D57E \\ In\ U_2, \text{hasFingerprint}(c_1\colon Student) = 8A08\ 617D\ 9FC3\ D57E \end{cases}$$

Example 4 *Consider two university libraries: library1 (Lib_1) and library2 (Lib_2). Suppose that $CatalogID$ gives a unique identification number to each book in each library.*

$$\begin{cases} In\ Lib_1, hasCatalogID(c_1\colon LibBook) = \text{'C51-D'} \\ In\ Lib_2, hasCatalogID(c_1\colon LibBook) = \text{'E11-G'} \end{cases}$$

[22] International Standard Book Number (ISBN) is a unique identifier for commercially published books. Currently, ISBN is a 10-digits code that includes four parts: group code, publisher, title, and a check digit.

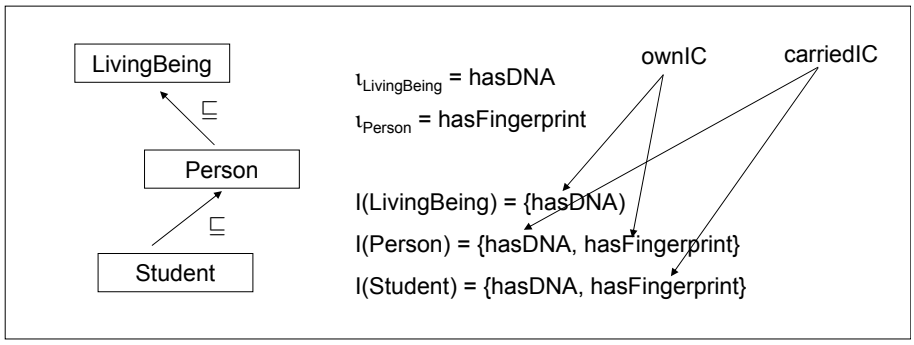

Figure 4.4: OwnIC and CarriedIC through IC inheritance

Since the catalogID of books are locally enumerated in each library, the *CatalogID* value of c_1 is possibly different from library1 to library2. When $LibBook \sqsubseteq PublishedBook$ with $\iota_{PublishedBook} = hasISBN$, similarly both library books have the same ISBN.

$$\begin{cases} In\ Lib_1, \text{hasISBN}(c_1\colon LibBook) = \text{0-837-120079-2} \\ In\ Lib_2, \text{hasISBN}(c_1\colon LibBook) = \text{0-837-120079-2} \end{cases}$$

More precisely, local ICs are able to identify individuals locally or only inside a possible world. In the case of IC, it provides a unique IC value to each individual and by that IC, individuals are globally identifiable. A discussion of local ICs vs global ICs can be seen in [151, 152, 153, 156].

Multiple ICs

The IC of a sort allows inheritance through subsumption relationships.

Definition 15 (IC Inheritance) *In a sortal taxonomy, IC are carried from a super-sort to its sub-sorts through subsumption relationships and this is called IC inheritance. I call a set of ICs that all belong to a specific sort s an IC set denoted by $I(s)$. Thus, for two sorts s_1 and s_2, IC inheritance in a formal notation is*

$$\textit{if}\ \ s_2 \sqsubseteq s_1\ \ \textit{then}\ \ I(s_2) \supseteq I(s_1).$$

If sort s originates an IC, then the IC is called the *ownIC* of s and is denoted by ι_s. If a sort inherits an IC from a super-sort through subsumption relationship, then the IC is called *carriedIC* denoted by ι.

Example 5 *As shown in Figure 4.4,*

$$I(Person) = \{hasDNA, hasFingerprint\}$$

$$I(Student) = \{hasDNA, hasFingerprint\}$$

By Student $\sqsubseteq$ Person $\sqsubseteq$ Living_being where the ownIC of `Person`*, denoted by ι_{Person}, is hasFingerprint and the ownIC of* `LivingBeing`*, denoted by $\iota_{LivingBeing}$, is hasDNA.* `Person` *not only originates an IC, but also carries an IC from* `LivingBeing`*. However,* `Student` *does not originate an IC, and thus it does not have an ownIC.* `Student` *carries the ICs from* `Person`*. In an alternative way, we say* `Person` *supplies its ownIC to* `Student` *and* `Student` *carries the IC of* `Person`*.*

Some sorts may originate more than one ownIC. For example, not only `fingerprint` but also `iris_pattern` or `palm vein pattern`[23] are originated by sort `Person`. Thus, `Person` has multiple ownICs. When a sort has multiple ICs, though the IC values provided by each IC are semantically different, each IC is able to identify every individual of the sort. For example, the `fingerprint` and `iris_pattern` of a person are different but they both are able to identify that person. In summary, a sort can have multiple ICs—more than one IC–in two ways: (1) by IC inheritance or (2) by multiple ownICs (see [151, 152]).

ICs are Intrincis or Extrinsic

There is an issue: ICs are either *intrinsic* or *extrinsic*. Guarino & Welty discussed this issue as follows:

> "*Global unique IDs are used either in object-oriented systems to uniquely identify an object or in database systems to identify data records. Globally unique IDs provide ICs for data records, but not for the entities in the world the records describe. Two or more descriptions may refer to the same entity and our notion of IC is concerned with this entity. Globally unique IDs (and primary keys) are rather extrinsic that are required by a system to be unique. Our notion of IC is based mainly on intrinsic properties. However, this is not to say that the former type never uses intrinsic properties nor the latter never uses extrinsic ones. In practice, conceptual modellers may need both*" [147].

Some individuals are quite concrete, like a particular person, or a particular copy of a book. Some are more abstract, like the subject matter covered by a book. The important property of most individuals is that they have an identity, which allows them to be distinguished from one another and to be counted. Modeling of individuals is therefore made easier if they have unique identifiers, like ISBN for published books. Unfortunately, this may not always be the case. For example, if one sees two brand new copies of a book on a bookshelf, which may not be distinguishable by any property known to us, one can still say that they are different copies of the book. In information management systems, and sometimes in the real world, this leads us to devise some kind of "extrinsic" identification scheme. For example, books on the library shelf are assigned a copy number. In this paper, as in object-oriented software systems, we will tend to assign arbitrary internal identifiers to objects, such as C51-E or OOSBOOK23.

According to Borgida and Brachman [52], a general heuristic is that if we (people) expect certain notions to be identifiable, then they must be modeled as individuals. Moreover, they must be identifiable through unique IC values or identifiers. Therefore, in this research, both intrinsic and extrinsic properties are used as ICs if they satisfy Equation (4.4) or (4.5). For examples, fingerprint is intrinsic but ISBN is rather extrinsic.

4.2.4 Existential Rigidity

Rigidity is strictly related to the philosophical notion of *essence*. Essentiality is a relationship between an individual and a concept. Lowe [49] defined essentiality as follows:

[23] Fujitsu has announced its contactless palm vein authentication technology at `http://www.fujitsu.com/global/news/pr/archives/month/2005/20050630-01.html`.

Definition 16 (Essentiality) *If a concept is essential for an individual, then the individual is always an instance of the concept in every possible world. In a formal notation, concept p_ϕ is essential iff:*

$$\Box\forall x p_\phi(x). \tag{4.6}$$

The notion of rigidity originally introduced in [140] is very much related to Lowe's essentiality. A rigid concept was first axiomatized in OntoClean [145] as given below.

$$\forall x[p_\phi(x) \rightarrow \Box p_\phi(x)] \tag{4.7}$$

Equation 4.7, concept p_ϕ is rigid if it is essential for all of its individuals. Kaplan [19] pointed out that Equation 4.7 is weak to describe the standard use of rigidness for universals, and proposed the following.

$$\forall x[\Diamond p_\phi(x) \rightarrow \Box p_\phi(x)] \ \ [\textbf{basic rigidity}] \tag{4.8}$$

This amounts to saying the extension (set of individuals) of a rigid concept is the same in all accessible worlds. In 2000 and after, the later OntoClean papers treated all OntoClean formulas (or axioms) as necessary, and added time to reflect more accurately the way time and modality are related [148, 147, 149].

$$\Box\forall x, t[p_\phi(x,t) \rightarrow \Box\forall t' p_\phi(x,t')] \ \ [\textbf{temporal rigidity}] \tag{4.9}$$

Equation 4.9 states that the extension of a temporally rigid concept must be the same for all time points and all possible worlds.

In general, the rigid designation in modal context is "*it designates the same thing in all possible worlds*". However, Kai-Yee Wong [100] mentioned what Saul Kripke likes to say about rigidness is with *existence conception* [190, 191, 192, 193, 194]. The following passage—excerpted from Wong's discussion [100]—is an original explanation by Saul Kripke for this point:

> *"....when I use the notion of rigid designator, I do not imply that the object referred to necessarily exists. All I mean is that in any possible world where the object in question does exist, in any situation where the object would exist, we use the designator in question to designate that object. In a situation where the object does not exist, then we should say that the designator has no referent and that the object in question so designated does not exist [193]...." (Saul Kripke, 1971)*
>
> *"...a designator rigidly designates a certain object if it designates that object wherever the object exists [194]..." (Saul Kripke, 1972)*

Andersen and Menzel [214], pointed out that Equation 4.9 does not accurately capture the intuition of Kripke's rigid designator expressed as, "*An individual of a rigid concept can not cease to be an individual of that concept, unless it ceases to exist*", since Equation 4.9 requires an individual to be an instance of the concept always and in all possible worlds, for example, if Person is a rigid concept, Aristotle must be a person, even in a possible world in which he does not exist [148]. To address this, Andersen and Menzel [214] proposed the following formula.

$$\forall x, t[\Diamond p_\phi(x,t) \rightarrow \Box\forall t'(E(x,t') \rightarrow p_\phi(x,t'))][\textbf{temporally existential rigidity}] \tag{4.10}$$

Carrera et al., also proposed a similar idea:

$$\forall x, t[\Diamond(E(x,t) \wedge p_\phi(x,t)) \rightarrow \Box\forall t'(E(x,t') \rightarrow p_\phi(x,t'))] \tag{4.11}$$

with a slightly stronger restriction on existence.

Both Andersen & Menzel and Carrara et al., pointed out that their accounts of rigidness, by introducing actual existence in the antecedents, say nothing about what happens to entities when they do not exist, leaving open the possibility that an individual of a rigid concept could change its membership when it does not exist.

Regarding the above quoted reference, I also agree to consider the actual existence in rigidity. Removing the time parameter from Equation 4.10, Welty & Andersen [35] proposed the existential rigidity:

$$\forall x[\Diamond p_\phi(x) \rightarrow \Box(E(x) \rightarrow p_\phi(x))] \quad [\textbf{existential rigidity}] \tag{4.12}$$

This tells us that a concept carries existential rigidity when an individual of the concept exists in any accessible world and instantiates the concept. This characterization is useful for concepts defined in ontologies that consider only single states of affairs and treat time, space, possibility, etc., as modalities.

Since I have fixed my scope to sorts and the actual existence of individuals, here I apply Equation 4.12 and define the precise meanings of existentially-rigid sort and existentially-anti-rigid sort according to the Kripke semantics given in language $\mathcal{L}^E$.

Definition 17 (Existential Rigidity) *For any sort s, s is existentially* **rigid** *iff*

$$\forall x[\Diamond p_s(x) \rightarrow \Box(E(x) \rightarrow p_s(x))], \tag{4.13}$$

otherwise, s is existentially **anti-rigid** *iff*

$$\forall x[\Diamond p_s(x) \rightarrow \Diamond(E(x) \wedge \neg p_s(x))]. \tag{4.14}$$

In the rigid case, if every individual of a sort in world w exists in every accessible world w' such that wRw', the individual is always a member of the sort. In the anti-rigid case, this is not so.

Example 6 *Let us consider Person as a sort. Every person is a person in every possible world if s/he exists there. Thus, we can define* `Person` *as a rigid sort. However, we cannot expect every person is always a student. Therefore,* `Student` *is an anti-rigid sort while* `Person` *is a rigid sort. Moreover, a person may have different roles in different worlds, such as* `Student`, `Part-timeEmployee`, *and* `ResearchStudent`. *As shown in Figure 4.5, John is defined as a student in world w, but he may be changed in world w' to a part-time employee, and to a research student in world w''. However, he is still a Person in every world. Thus, Person is a rigid sort, and others are anti-rigid sorts.*

Of course, any person can cease being a person after death because no living being can live forever. Thus, I consider the actual existence of an individual in an accessible world, for the rigidity of a sort.

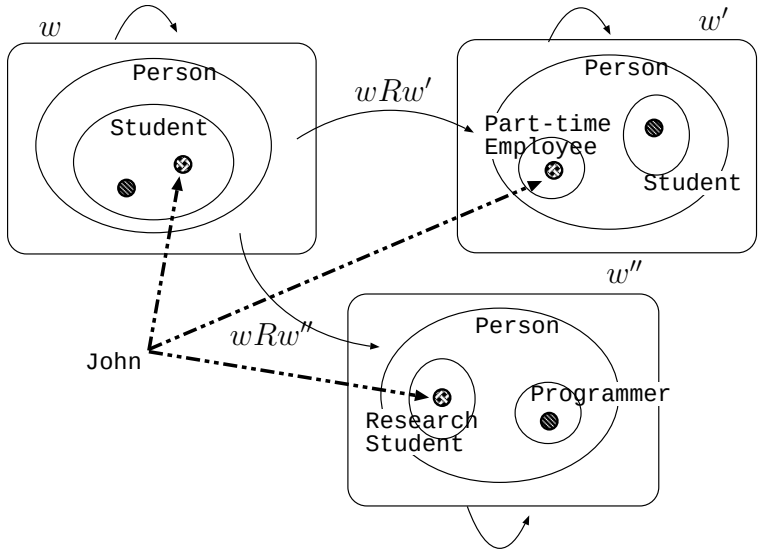

Figure 4.5: Rigid sorts and anti-rigid sorts

Example 7 *For an alternative example of rigid and anti-rigid sorts in the wine domain, let us define that* `DessertWine` *is a wine which is served for dessert,* `CookingWine` *is a wine which is used in cooking of certain dishes, and* `AppetizerWine` *is a wine which is served to stimulate the appetite before a meal. Whilst Wine is a rigid sort, DessertWine is an anti-rigid sort, because any individual of DessertWine in a world may stop being DessertWine in any accessible world. Suppose that 'Santa Margharita, Italy 2004' is defined as a DessertWine in restaurant w but that may be a CookingWine at restaurant w′, and an AppetizerWine in restaurant w″. However, it is still a wine in all restraunts.*

4.2.5 External and Existential Dependency

Dependency expresses the external and existential dependent relation of a certain sort to another disjoint sort whose individuals are neither a part nor a constituent of any individual of the sort. In order to define the notion of dependence, I need to discuss part and constituent relations between individuals of sorts. Moreover, I consider the notion of existential dependency between sorts. Here, I define the existential dependency based on [61].

Definition 18 (Existential dependency) *Sort s is existentially dependent on sort s′ iff, as a matter of necessity, some individuals of sort s′ exist whenever an individual of sort s exists, formally*

$$\Box\forall x[p_s(x) \wedge E(x) \rightarrow \exists y[P_{s'}(y) \wedge E(y)].$$

For example, there is at least one school for a student or there is at least one supplier for a customer.

Concerning the notions of part and constituent, there is a long history in philosophy, linguistics and cognitive sciences. The study of parthood (part-whole) relations can be traced back to the early days of philosophy, beginning with the presocratic atomists and continuing throughout the writings of Plato, Aristotle, Leibniz and the early Kant, to

cite just a few. The first attempt at a rigorous formulation of the theory was made by Edmond Husserl, but the first complete theory of parts, named *Mereology*, was proposed in 1916 by the Polish philosopher Stanislaw Lesniewski [195], who used the part-whole relation as a substitute for class membership in standard set theory. This theory was later elaborated by Leonard and Goodman in their "The Calculus of Individuals" [76].

In all philosophical theories of parts, including Lesniewski's mereology, the relation of (proper) part is in a partial ordering, i.e., an irreflexive, antisymmetric and transitive relation. For three individuals c_1, c_2, c_3, a proper part holds the following conditions.

- Every individual is not a part of itself.
- If c_1 is a part of c_2 then c_2 is not a part of c_1.
- If c_1 is a part of c_2 and c_2 is a part of c_3 then c_1 is a part of c_3.

For example, (a) a cerebellum is a part of a brain, (b) a brain is a part of a person, (c) a gear is a part of car engine, and (d) an engine is a part of a car. From these premises, we can conclude a cerebellum is a part of a person, and a gear is a part of a car. A variety of part theory can be learned in [61].

By attempting to differentiate between the linguistic expressions *part of* and *constituent of*, I show the following two propositions: one which is indeed a proper part relation and one which represents a case of a constituent.

1. A faculty is a part of university.
2. Clay is a constituent of a statute which is made of it.

In the later case, if A is the same lump of clay as long as it constitutes the same statue B, A would have necessarily the same properties as B and have a complete life-time internal dependency. For instance: (a) if a piece of B is removed, B is still the same statue and so is A still the same lump of clay, since it still constitutes the same statue; (b) If the form of B is altered, B ceases to exist and so does A, since it no longer constitutes the same statue [61]. In summary, a part-of relation states the component relation between two individuals, and constituent states the composition relation between individuals. The external dependency between two sorts concerns neither part nor constituent relation between their individuals.

Definition 19 (Externally Dependent) *Sort s is externally dependent on another sort s' if, for all individuals x of s, necessarily some individual y of s' exist, which is neither a part nor a constituent of x:*

$$\forall x \Box[p_s(x) \wedge E(x) \rightarrow \exists y(p_{s'}(y) \wedge E(y) \wedge p_d(x, y)] \tag{4.15}$$

where s and s' are disjoint.

I modify the original dependent definition[24] by defining existential dependency and by adding an External Dependency Relationship (EDR) denoted by p_d, so that Equation 4.15 states the external dependency of s to s' explicitly.

Example 8 `Student` *is externally dependent on* `School` *with "*`EnrollIn`*" relationship. This means we do not define a person who does not enroll in a school as a student. Thus, EnrollIn is called the EDR of* `Student` *to* `School`*. Similarly* `Parent` *and* `Child`*,* `Customer` *and* `Supplier`*, are some examples of externally dependent sorts.*

[24] $\forall x \Box[p_\phi(x) \rightarrow \exists y(p_\psi(y) \wedge \neg Part(y, x) \wedge \neg Constituent(y, x))]$ [147]

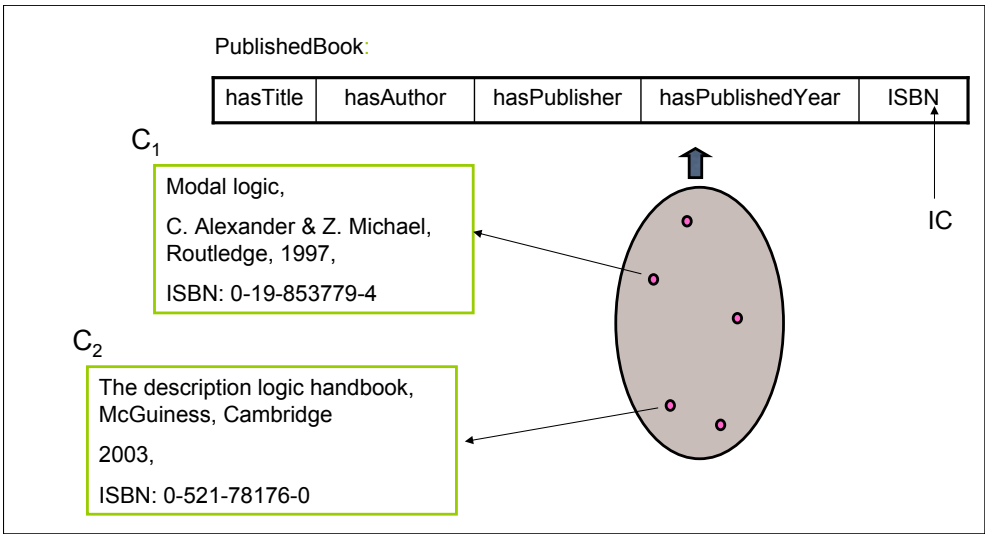

Figure 4.6: An example for sort, properties, and individuals

4.3 Modelling Semantically-enriched Ontologies

A central concern of this section is to model semantically-enriched ontologies by applying some ontological foundations. For this purpose, I aim at providing a philosophical classification of sorts.

Regarding both Frame and OWL specifications presented in Section 2.3, there are three fundamental modeling components in developing ontologies: *classes* for concepts (and sorts), *properties* for attributes and relations (called intensional knowledge) of concepts, and *individuals* for instances (called extensional knowledge) of each concept. An ontology with a universe of discourse constitutes a *populated ontology*, or ontology base.

4.3.1 Description of a Sort

In a sortal taxonomy $\langle S, \sqsubseteq \rangle$, S is a set of sorts. I call a set of properties that constitutes the intensional semantics of a sort *individual-level* properties, because the semantics of a certain indivdual is described in terms of these properties. For a sort $s \in S$, there is a set of individual-level properties $P^D(s)$. For example,

$$P^D(PublishedBook) = \{hasTitle, hasAuthor, hasPublisher, hasPublishedYear, \text{hasISBN}\}.$$

Every individual of a sort has a specific value for each property of the sort. Among the given properties, let us assume that `hasISBN` is defined as the IC of `PublishedBook`. By the definition of IC (Definition 14), every individual instantiated to `PublishedBook` must possess a unique ISBN value, as described in Figure 4.6.

For every $p \in P^D(s)$, there is a specific domain $\mathcal{D}_s$ and range $\mathcal{R}_p$ such that $p : \mathcal{D}_s \Rightarrow \mathcal{R}_p$. The domains and ranges for the example properties of PublishedBook are illustrated in Figure 4.7. I divide $P^D(s)$ into two kinds.

- **Object properties** $P_j(s) \subseteq P^D(s)$ are the properties that relate two individuals, that is, $p_j : \mathcal{D}_s \Rightarrow \mathcal{R}_j$ for every $p_j \in P_j(s)$.
  For example, $P_j(PublishedBook) = \{hasAuthor, hasPublisher\}$

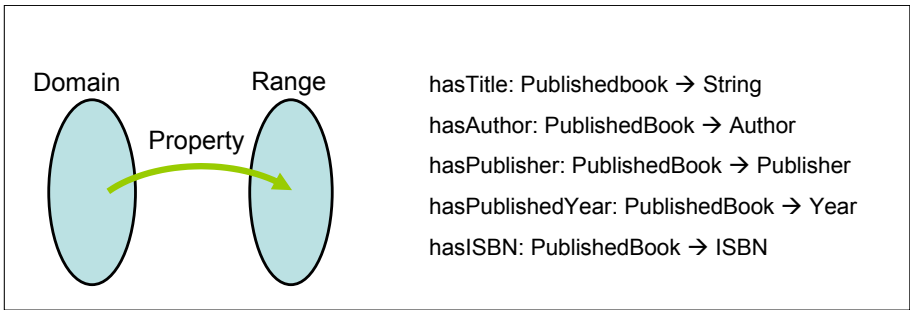

Figure 4.7: The domain and range of a property

- **Datatype properties** $P_t(s) \subseteq P^D(s)$ are the properties that relate an individual to a data value defined in a standard or user-defined datatype such as `String`, `Integer`, `Boolean`, `Date`, `Year`, `ISBN`, `URI`, etc., that is, $p_t \colon \mathcal{D}_s \Rightarrow \mathcal{R}_t$ for every $p_t \in P_t(s)$.
  For example, $P_t(PublishedBook) = \{hasTitle, hasPublishedYear, \text{hasISBN}\}$

Thus, the individual-level properties of sort s consists of object properties as well as datatype properties, that is $P^D(s) = P_j(s) \cup P_t(s)$. IC denoted by ι is a kind of datatype property. Then, the IC of `PublishedBook` can be described as $\iota_{\texttt{PublishedBook}} = \texttt{hasISBN}$. I call the value returned by an IC for an individual *"IC value"*, and a set of identity conditions for sort s *"IC set"* denoted by $I(s)$ such that $I(s) \subseteq P_t(s)$. $I(s) \neq \emptyset$ for every sort s because every sort originates or carries at least one IC. For every $\iota \in I(s)$, $\iota \colon \mathcal{D}_s \Rightarrow \mathcal{R}_\iota$. Moreover, every IC ι has an inverse functional property such that $\overleftarrow{\iota} \colon \mathcal{R}_\iota \Rightarrow \mathcal{D}_s$. As shown in Figure 4.8, isISBNof: ISBN $\Rightarrow$ *PublishedBook* is the inverse function of hasISBN: *PublishedBook* $\Rightarrow$ ISBN. Thus, *one-to-one* relations such as

$$\text{hasISBN}(c_1 \colon PublishedBook) = \text{0-123-45678-9}, \text{ and}$$

$$\text{isISBNof}(\text{0-123-45678-9}) = c_1 \colon PublishedBook$$

By Definition 14, one-to-one relation is fixed between the domain and range of an IC, as depicted in Figure 4.8. This characteristic differentiates ICs from other properties.

One of the major concerns in this research is if ontological concepts are considered as sorts, then how to represent them in terms of OWL. I intuitively found that a sort can be represented as an OWL class (`owl:Class`) with the restriction having at least one IC (ownIC or carriedIC). Also, `owl:Datatypeproperty` can be used to represent IC with the restriction of `owl:FunctionalProperty` and `owl:InverseFunctionalProperty` regarding its characteristic of one-to-one functional. In addition, = 1 cardinality is set up to restrict every individual possesses at least one IC value. I present the description of sort PublishedBook in the form of OWL syntax in Figure 4.9.

4.3.2 A Classification of Sorts

My objective of modeling semantically-enriched ontologies is to provide a well-structured taxonomy and adequate semantics for ontologies concerning the issue of semantic heterogeneity in ontology matching. Therefore, I provide a classification of sorts and define concept-level properties of sorts, according to the classification. Then, I present a model (called EnOntoModel) of semantically-enriched ontologies in the next section.

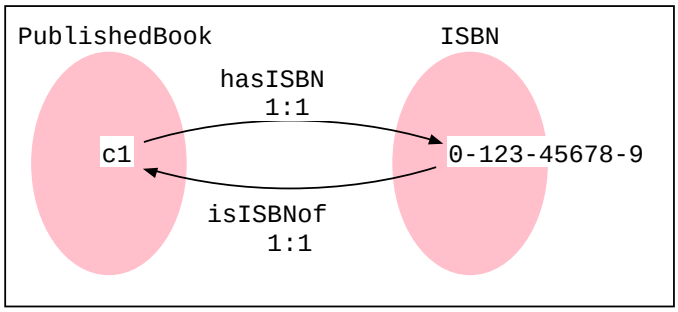

Figure 4.8: IC is one-to-one functional

```
<rdf:RDF xmlns="http://www.owl-ontologies.com/Ontology1156144011.owl#"
    xml:base="http://www.owl-ontologies.com/Ontology1156144011.owl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#">
   <owl:Ontology rdf:about="">
      <owl:imports rdf:resource="http://protege.stanford.edu/plugins/owl/protege"/>
   </owl:Ontology>
   <owl:ObjectProperty rdf:ID="hasAuthor">
      <rdfs:domain rdf:resource="#PublishedBook"/>
      <rdfs:range rdf:resource="#Person"/>
   </owl:ObjectProperty>
   <owl:DatatypeProperty rdf:ID="hasISBN">
      <rdf:type rdf:resource="&owl;FunctionalProperty"/>
      <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
      <rdfs:domain rdf:resource="#PublishedBook"/>
      <rdfs:range>
         <rdfs:Datatype>
            <xsp:minLength rdf:datatype="&xsd;int">10</xsp:minLength>
            <xsp:base rdf:resource="&xsd;string"/>
         </rdfs:Datatype>
      </rdfs:range>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="hasPublishedYear">
      <rdfs:domain rdf:resource="#PublishedBook"/>
      <rdfs:range rdf:resource="&xsd;gYear"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="hasTitle">
      <rdfs:domain rdf:resource="#PublishedBook"/>
      <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:Class rdf:ID="Person"/>
    <owl:Class rdf:ID="PublishedBook">
      <rdfs:subClassOf rdf:resource="&owl;Thing"/>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#hasISBN"/>
          <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
        </owl:Restriction>
      </rdfs:subClassOf>
    </owl:Class>
</rdf:RDF>
```

sort

IC

Figure 4.9: A representation of Sort and IC in OWL

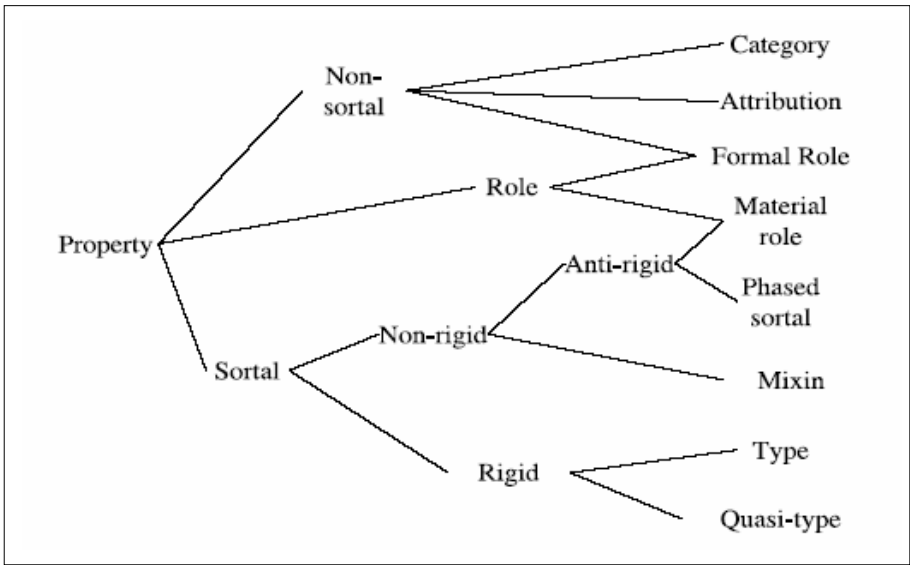

Figure 4.10: A classification of ontological concepts in OntoClean [147]

In OntoClean, Guarino & Welty [147] provided a classification of ontological concepts including sortals and non-sortals as shown in Figure 4.10. Guizzardi [61] designed a classification of universals[25] as ontological distinctions for conceptual modeling by UML[26]. I show this classification in Figure 4.11.

Based on the classifications of sorts in OntoClean and by Guizzardi, I define four categories of sort: type, quasi-type, role, and phase. Before defining each category of sort, I need to clarify some critical properties. They are Identity Condition (IC), Common Value Attribute (CVA), External Dependency Relation (EDR), and Common Constraint (CC). For IC, I refer to Definition 14.

- Common Value Attribute (CVA): A datatype property (represented by `owl:DatatypeProperty`) is defined as a CVA for a sort, if the property provides a common attribute value for all individuals belonging to the sort. For example, property `hasColor` is a CVA for sort `WhiteWine` because it provides a common color value "White" for every individual of `WhiteWine`. Similarly, `hasColor` is a CVA for `RedWine` by providing the same color value "Red" to every individual of `RedWine`. CVA is denoted by p_a.

- External Dependency Relation (EDR): An object property (represented by `owl:ObjectProperty`) is defined as an EDR for a sort, if the property provides an existentially and externally dependent relationship between two individuals which are neither a part nor a constituent to each other. For example, property `EmployFor` is an EDR for sort `Employee` because there probably exists at least one employer (firm or organization) for any employee. This may need an assumption: "every employer is neither a part nor a constituent of an employee". EDR is denoted by p_d.

[25] In metaphysics, a universal is a type, a property, or a relation. We can map the category of *universal* into Object Oriented classes, Entity types of ER-models, and OWL classes.

[26] Unified Modeling Language

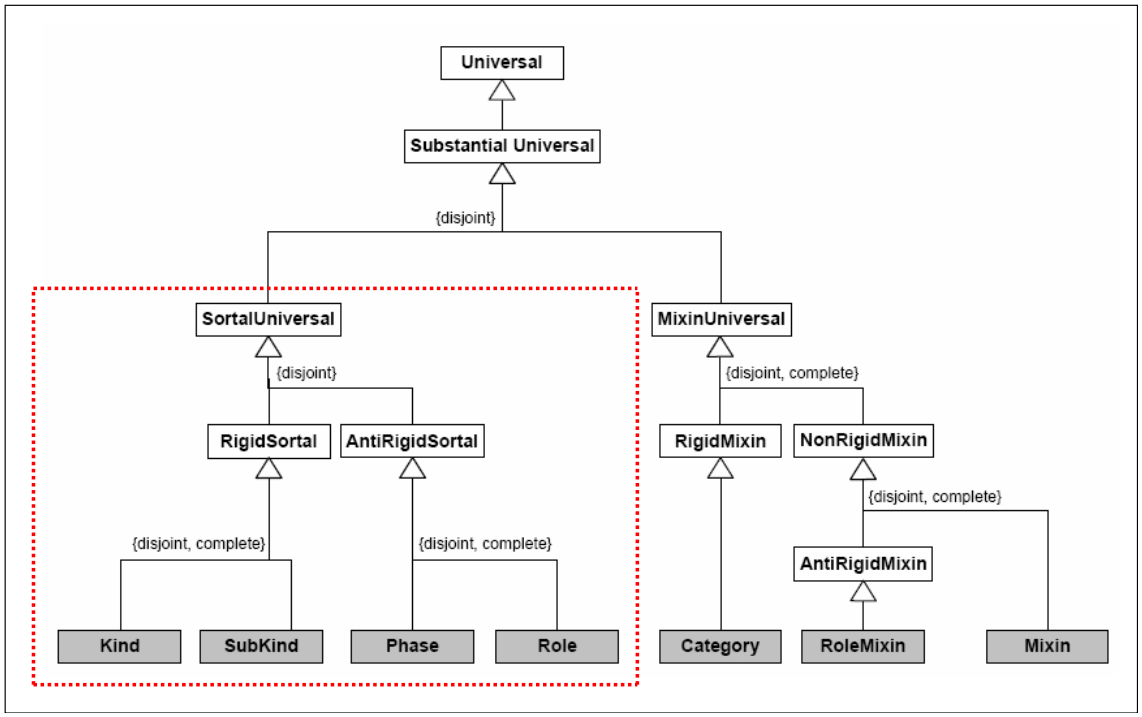

Figure 4.11: A classification of universals by Guizzardi [61]

- Common Constraint (CC): A datatype property (represented by `owl:DatatypeProperty`) is defined as a CC for a sort, if the property provides a common constraint such as the same boolean value, data value range, or qualification, that can distinguish the individuals of the sort from being the instances of a disjoint sibling sort. For example, property `hasAge` is a CC for sorts `Boy` and `Man` with an age constraint. CC is denoted by p_c.

Definition 20 (Type sort) *If a sort is existentially rigid and it originates (or supplies) an IC, then the sort is called a type sort.*

Some examples of type sort are `Person`, `PublishedBook`, and `Wine`, with ICs hasFingerprint, hasISBN, and hasWineName[27], respectively. type sort is also known as *rigid substance sortal* that supplies a principle of identity for its individuals [60]. Since every individual is assumed to be an identifiable object, any individual must be an instance of a type sort, directly or indirectly.

Definition 21 (Quasi-type sort) *If a sort is existentially rigid but it does not originate an IC, then it is called a quasi-type sort.*

More precisely, quasi-type sorts are partitions[28] of a type sort, specialized with a CVA. For example,
(a) `MalePerson` and `FemalePerson` are the quasi-type sorts of `Person` with CVA

[27]hasWineName includes winery, appellation, and a vintage. For example, "Joseph Drouhin 2004 Chablis Premier Cru" is the name of a wine produced from 'Joseph Drouhin', appellation is 'Chablis Premier Cru', and vintage year is '2004'.

[28]The partions of a type sort form a complete generalization and they are disjoint from each other.

`hasGender(c:MalePerson, ''Male'')` and `hasGender(c:FemalePerson, ''Female'')`;
(b) `RedWine` and `WhiteWine` are the quasi-type sorts of `Wine` with CVA `hasColor(c:RedWine, ''Red'')` and `hasColor(c:WhiteWine, ''White'')`;
(c) `PublishedBookInLogic` and `PublishedbookInNon-Logic` are the quasi-type sorts of `PublishedBook` with CVA `hasSubject(c:PublishedBookInLogic,''Logic'')` and `hasSubject(c:PublishedBookInNon-Logic,''Non-Logic'')`.
The quasi-type sorts of a certain type sort are disjoint to each other, that is, if an individual person is modeled as an instance of `MalePerson` then he cannot be an instance of `FemalePerson`.

Definition 22 (role sort) *If a sort is existentially anti-rigid and it is externally dependent on another sort by holding an EDR, then the sort is called a role sort. Moreover, the domains of role sorts are not necessarily disjoint.*

`Student`, `Employee`, and `Customer`, are some examples of role sorts, that is (a) a student is a person who enrolls in a school or university, (b) an employee is a person who is hired by an organization to perform a job, and (c) a customer is a person who buys a product from a supplier. Then, the following EDRs will hold for each of them:
(a) `enrollIn(Student, School)`
(b) `employFor(Employee, Employer)`
(c) `BuyProduct(Customer, Supplier)`
In a reverse way, `SupplyProduct(Supplier, Customer)` can be used as an EDR, meaning that every supplier is externally dependent on some customers. An individual can be a member of more than one role sort subsumed by the same type sort, that is, a person can be a student as well as an employee.

Definition 23 (Phase sort) *If a sort is existentially anti-rigid and does not need an EDR like role sort, then the sort is called a phase sort. phase sorts constitute possible stages in the history of a super-sort they specialize, by holding a Common Constraint (CC). Thus, they are disjoint to each other.*

For example, (a) `Girl`, `Teenager`, and `Woman`, are the possible stages of `FemalePerson`, with age constraint such that a female person under 12 years old is a girl, over 18 years old is a woman, or between 12 and 18 years old is a teenager; (b) `Caterpillar` and `Butterfly` are phase sorts of `Lepidopteran` with wing constraint, that is, the boolean value "false" will be assigned for `Caterpillar` and "true" will be assigned for `Butterfly` concerning CC `hasWing`; (c) `UndergraduateStudent`, `MasterStudent`, and `DoctoralStudent`, are phases of university student life with an enrollment constraint such as "enroll for undergraduate degrees", "enroll for graduate degrees", and "enroll for Phd".

Contrary to role sort, an individual cannot belong to more than one phase sort. Whilst quasi-type sorts are the partitions of a type sort only, phase sorts can be the partitions of any other sort category. A major distiction between a quasi-type sort and a phase sort is rigidity. A quasi-type sort is rigid, however a phase sort is anti-rigid.

According to the above definitions, S can be divided into four subsets:

$$S = S_{\text{type}} \cup S_{\text{quasi-type}} \cup S_{\text{role}} \cup S_{\text{phase}}$$

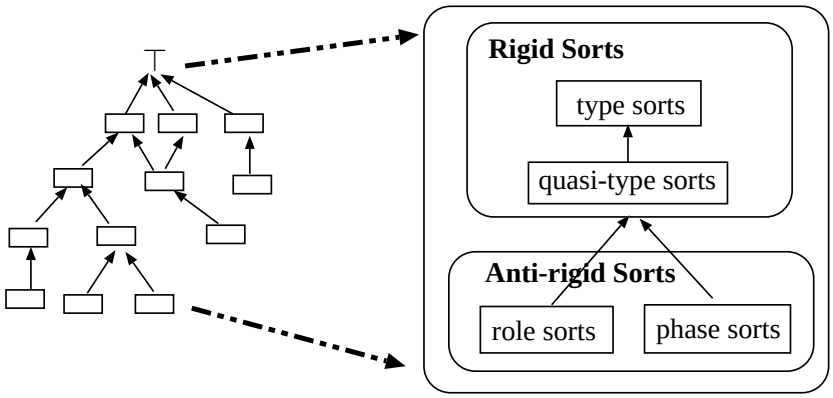

Figure 4.12: A typical structure of sortal taxonomy

where S_{type} is a set of type sorts, $S_{quasi-type}$ is a set of quasi-type sorts, S_{role} is a set of role sorts, and S_{phase} is a set of phase sorts. I claim that each subset of S is disjoint to the others, because their modality and identifiable characteristics are different. This disjointness is proved as follows.

- By Definition (17), if sort $s \in S$ is existentially anti-rigid, then s is not rigid, and vice versa.
- By Definitions (20) & (21), if sort $s \in S$ is a quasi_type sort, then it is not a type sort, and vice versa.
- by Definitions (22) & (23), if sort $s \in S$ is a phase sort, then it is not a role sort, and vice versa.

A typical structure of the above classification is depicted in Figure 4.12. It can also be called a skeleton of sortal taxonomies which preserve the condition "*anti-rigid sorts never subsume rigid sorts*" [147]. I do not mean that every ontology needs to complete this classification. An important fact is to obey the definition of each sort category and not to violate the taxonomic structure given in Figure 4.12.

Concerning identity, there is an important notion in ontological conceptualization.

> "No individual can instantiate both of two sorts if they have different criteria of identity associated with them (Lowe, 1989)" [49].

Referring to the above quoted statement, it is clear that a type sort cannot be subsumed by another type sort which has an incompatible IC, that is, if any individual of the sort does not possess an IC value supplied by such IC. For example, if an individual is instantiated as a car then it cannot be a computer, because the ICs of `Car` and `Computer` are not compatible to each other. This is known as *IC incompatibility* [147].

Therefore, I additionally employ the two conceptual constraints described below.

- **Constraint1**: Every top-most sort of a sortal taxonomy in a given ontology must be a type sort which originates (or supplies) an IC to identify an individual globally in multiple worlds, regarding identity for every individual.
- **Constraint2**: A type sort is not allowed to have multiple subsumption relationships such that $s_3 \not\sqsubseteq s_1$ and $s_3 \not\sqsubseteq s_2$ if all are type sorts and $s_1 \asymp s_2$, because no individual

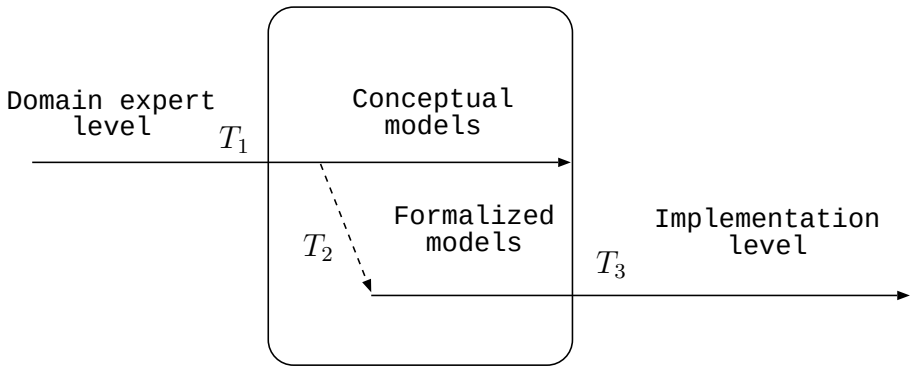

Figure 4.13: A transition of essential processes in ontology development adopted from Blum (1996)

possesses two incompatible IC values (e.g. an alcoholic drink cannot be defined as both wine and whisky).

4.3.3 Concept-level Properties of Sorts

According to METHONTOLOGY [110], there are five major activities in the development of ontologies. Those activities are (a) specification, (b) conceptualization, (c) formalization, (d) implementation, and (e) maintenance. Figure 4.13 [11] illustrates the adoption of Blum's essential process model [25] of software engineering to ontological engineering. The transformation T_1, which refers to the conceptual modeling process, can be seen as a transformation of an idea of a domain into a conceptual model that describes such an idea. The transformation T_2 converts the conceptual model into a formalized model. The transformation T_3 transforms the formalized model into a coded model which can be executed in a computer.

In order to apply the classification of sorts into the formal conceptual model, I create a modeling component, that is, concept-level properties of sorts. This is my novel idea of embedding the sort classification into ontology model. Here, I like to distinguish between concept-level properties and individual-level properties. For an ontological concept/class, a set of individual-level properties (known as intensional knowledge) are defined for the precise semantics (or meaning) of the concept. They are abstracted from the specification of individuals of the concept. In an alternative speaking, these properties are used to describe data of each specific individual. Inheritance is allowed among individual-level properties along subsumption relationships. For sort s, $P^D(s)$ is a set of individual-level properties. When $s_2 \sqsubseteq s_1$, $P^D(s_2) \supseteq P^D(s_1)$. Concept-level properties are different from individual-level properties. They are defined only for the conceptual knowledge of a sort such as it is a type sort and it has an ownIC, etc. Inheritance is restricted among concept-level properties along subsumption relationships. Let $P^C(s)$ be a set of concept-level properties for sort s. For any $s_2 \sqsubseteq s_1$, $P^C(s_2) \not\supseteq P^C(s_1)$.

At the current state, I define two concept-level properties based on the sort classification. They are the category and classification constraint of a sort. More clearly, the category of a sort describe a particular sort is classified under which sort category such as type, quasi-type, role, or phase. When a sort is classified under a specific category, there is a specific constraint that forces individual-level properties defined for the sort to be

Table 4.1: Two kinds of concept-level properties by sort classification

Sort category	**Constraint**
type	ownIC: $\iota_s \in P^D(s)$
quasi-type	CVA: $p_a \in P^D(s)$
role	EDR: $p_d \in P^D(s)$
phase	CC: $p_c \in P^D(s)$

semantically adequate. Table 4.1 describes sort categories and their specific constraints as follows:

1. If sort s is conceptualized as a type sort then the sort category of s will be "type". Every type sort originates an IC, thus the constraint for s is $P^D(s)$ must include ι_s.

2. If sort s is conceptualized as a quasi-type sort then the sort category of s will be "quasi-type". Every quasi-type sort is restricted to possess a CVA, thus the constraint for s is $P^D(s)$ must consist of p_a.

3. If sort s is conceptualized as a role sort then the sort category of s will be "role". Every role sort is restricted to possess an EDR, thus the constraint for s is $P^D(s)$ must consist of p_d.

4. Similarly, if sort s is conceptualized as a phase sort then the sort category of s will be "phase". Every phase sort is restricted to possess a CC, thus the constraint for s is $P^D(s)$ must consist of p_c.

Concept-level properties describe the conceptual knowledge for concepts themselves and not for their individuals. The concept-level properties of a sort controls not only subsumption relationship of the sort, but also what kinds of individual-level properties should be explicitly defined for a precise semantics of the sort. Suppose that sort s_1 is going to be defined as a role sort. Then, the conceptual modeler has to think of the following questions:

- What are the property EDR and dependent sort of s_1, that prove s_1 to be a role?
- Is there a type sort s_2 such that $s_1 \sqsubseteq s_2$?
- What is the ownIC of s_2 which can globally identify every individual of s_1?

Expressed another way, the concept-level properties enrich the semantics of sorts.

In the implementation level, I use `owl:DatatypeProperty` to implement concept-level properties. Since sort categories are disjoint to each other, each sort can be conceptualized under only single category, and not more than one. Thus, sort category is a functional property. In the case of classification constraint, it records the name of specific domain-level property according to the defined constraint. I define these two concept-level properties to use in later matching process.

4.3.4 A Conceptual Model of Semantically-enriched Ontologies

Ontologies are the conceptualized models of a domain that basically consists of a set of concepts, their subsumption relationships, a set of individual-level properties, and some axioms. Thus, I model a source ontology as a quaduple $O = \langle S, \sqsubseteq, P^D, A \rangle$ where S is a set of concepts, $\langle S, \sqsubseteq \rangle$ is a taxonomy of subsumption relationships $\sqsubseteq$ between any two concepts, P^D is a set of individual-level properties, and A is a set of ontological axioms and constraints. The formalization of each modeling component of O is assumed to be the same as OWL-DL ontologies [52].

Following by the theoretical foundations discussed in the above sections, mainly the classification of sorts, now I define a formal model of semantically-enriched ontologies. The model is called EnOntoModel, in which concept-level properties are additionally embedded in O to enrich the semantics of concepts.

Definition 24 (EnOntoModel) *EnOntoModel is a quintuple $O^E = \langle S, \sqsubseteq, P^C, P^D, A \rangle$ where S is a non-empty set of sorts, $\langle S, \sqsubseteq \rangle$ is a taxonomic structure of S with subsumption relationship $\sqsubseteq$, P^C is a set of concept-level properties such that $P^C(s)$ for each sort $s \in S$, P^D is a set of individual-level properties such that $P^D = \{P^D(s) \mid s \in S\}$, and A is a set of ontological axioms and constraints.*

The following axioms and constraints are defined as standards for any EnOntoModel-based ontology.

A={
[$a1$] For any $s_1, s_2, s_3 \in S_{type}$, if $s_1 \asymp s_2$ then $s_3 \not\sqsubseteq s_1$ and $s_3 \not\sqsubseteq s_2$ then
[$a2$] For any $s_1 \in S_{type}, s_2, s_3 \in S_{quasi-type}$, if $s_2 \sqsubseteq s_1$ and $s_3 \sqsubseteq s_1$ then $s_2 \asymp s_3$
[$a3$] For any $s_1 \in S, s_2, s_3 \in S_{phase}$, if $s_2 \sqsubseteq s_1$ and $s_3 \sqsubseteq s_1$ then $s_2 \asymp s_3$
[$c1$] For any $s \in S_{type}$, $\iota_s \in P^D(s)$ and '=1' cardinality for ι_s
[$c2$] For any $s \in S_{quasi-type}$, $p_a \in P^D(s)$
[$c3$] For any $s \in S_{role}$, $p_d \in P^D(s)$
[$c4$] For any $s \in S_{phase}$, $p_c \in P^D(s)$
}

Moreover, other domain-dependent constraints can be defined in ontologies. A simple domain ontology named `Research-Community.owl` is presented below as an example of EnOntoModel.

Example 9 (`Research-Community.owl`) *Domain concepts defined in the ontology are listed according to their sort categories, as follows:*

$S_{type} = \{ResearchProject, SoftwareTool, ResearchPublication, Event, Person, LegalOrganization\}$

$S_{quasi\text{-}type} = \{CommercialProject, AcademicProject, OntologyEditor, NLParser, Reasoner, JournalPaper, ProceedingsPaper, Conference, Workshop, Seminar, Enterprise, Association, ResearchInstitute, University\}$

$S_{role} = \{Employee, Researcher, FacultyMember, AdminMember, ResearchStudent, Author, Secretary, Manager, President\}$

$S_{phase} = \{AssociateProfessor, Professor, MasterStudent, PhdStudent, ScholarshipEligible, ScholarshipNon\text{-}eligible\}$

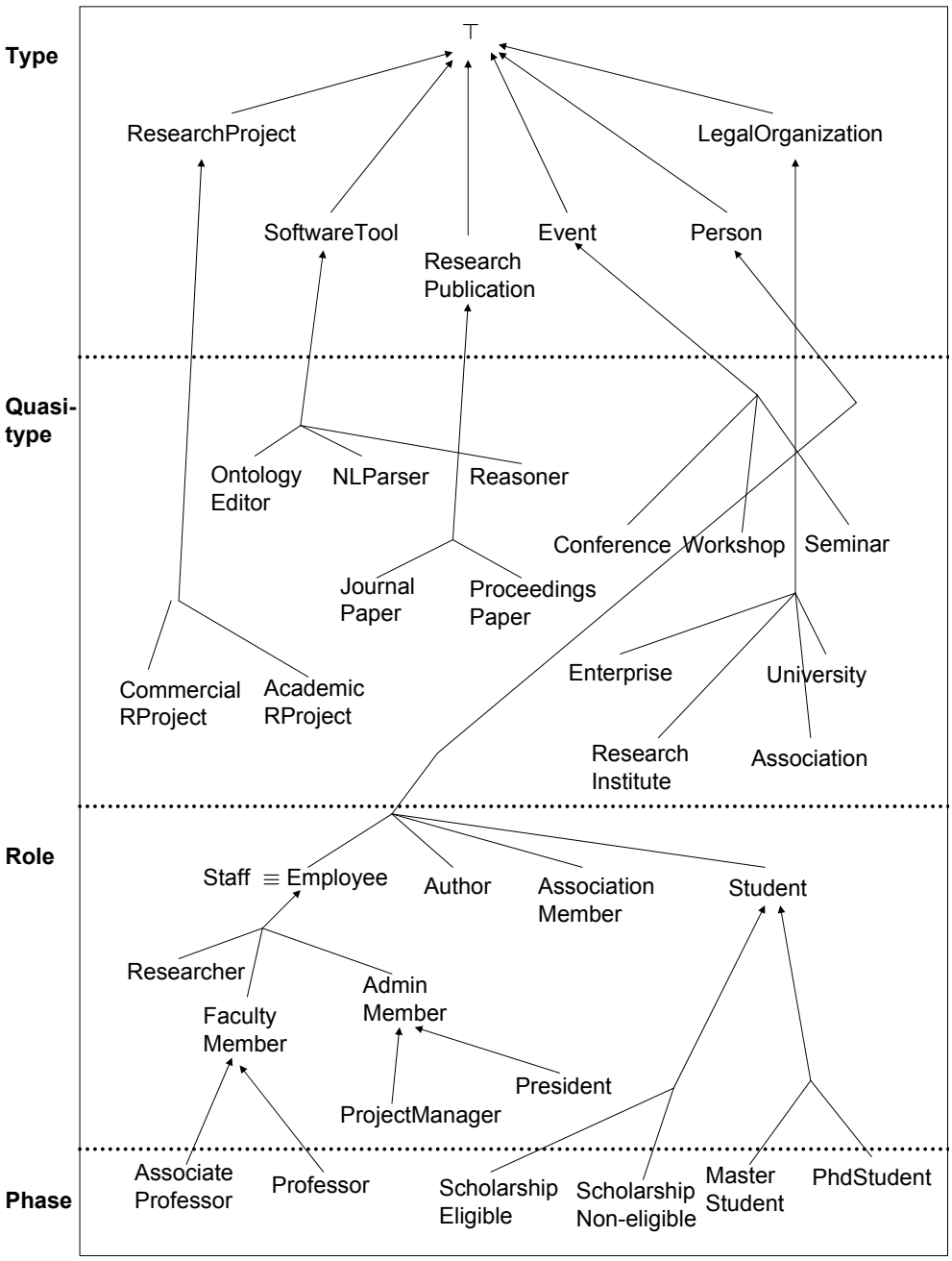

Figure 4.14: The taxonomy of `Research-Community.owl`

Table 4.2: Type sorts and their ownICs

Type Sort	**OwnIC**(denoted by ι_s)
ResearchProject	{hasRProjName, hasRProjOrganizer}
SoftwareTool	{hasSTName, hasSTDeveloper}
ResearchPublication	{hasRPubTitle, hasRPubAuthors, hasRPubTypeID}
Event	hasETitle
Person	hasFingerPrint
LegalOrganization	hasOrgName

Table 4.3: Quasi-type sorts and their CVAs

Quasi-type Sort	**CVA**(denoted by p_a)
CommercialRProject	hasRProjType="commercial"
AcademicRPoject	hasRProjType="academic"
OntologyEditor	hasSTCategory="ontology editor"
NLParser	hasSTCategory="NL parser"
Reasoner	hasSTCategory="reasoner"
JournalPaper	hasRPubType="journal issue"
ProceedingsPaper	hasRPubType="proceedings"
Conference	hasEType="conference"
Workshop	hasEType="workshop"
Seminar	hasEType="seminar"
Enterprise	hasOrgGroup="enterprise"
ResearchInstitute	hasOrgGroup="research institute"
University	hasOrgType="university"
Association	hasOrgType="association"

The taxonomic structure of `Research-Community.owl` is described in Figure 4.14 which follows the standard axioms and constraints of EnOntoModel. Moreover, other domain-dependent constraints such as `FacultyMember` $\asymp$ `Student`, `Staff` $\equiv$ `Employee`, etc., are also defined.

Let me explain the classification system of given concepts. According to Constraint1 defined in Section 4.3.2, the root of each sub-taxonomy is restricted as a type sort that supplies an IC for its individuals. Table 4.2 describes the ownIC of each type sort. For extrinsic ICs, I use tuples of individual-level properties as ICs. By a certain IC, the individuals of each type sort are identifiable. For example, the IC value of each `ResearchProject` individual is a couple of research project name and organizer name, such as ``The 21st Century COE Program'' +``Japan Advanced Institute of Science and Technology''. It is similar that each event has a unique event title such as ``AAAI-06: Twenty-First National Conference on Artificial Intelligence'' for the IC of AAAI-06 which is an instance of `Conference`. Here, `hasRPubTypeID` relates between a certain research publication and the identifier of a collection in which it is published (an ISSN of a journal issue or an ISBN of a proceedings). A list of quasi-type sorts and their CVAs with restricted values are described in Table 4.3. The quasi-type sorts are the partitions of a type sort, but do not originate a new IC and only carry the

Table 4.4: Role sorts and their EDRs

Role Sort	**EDR** (denoted by p_d)	**Dependent Sort**
Employee	employIn	LegalOrganization
Researcher	hasRProj	ResearchProject
FacultyMember	employIn	University
AdminMember	administerIn	LegalOrganization
ProjectManager	manage	ResearchProject
President	headOf	LegalOrganization
Author	hasRPublication	ResearchPublication
AssociationMember	isMemberOf	Association
Student	EnrollIn	University

Table 4.5: Phase sorts and their CCs

Phase Sort	**CC** (denoted by p_c)
AssociateProfessor	hasFacultyPosition="associate professor"
Professor	hasFacultyPosition="professor"
ScholarshipEligible	hasAge="$\leq$35"
ScholarshipNon-eligible	hasAge="$\geq$36"
MasterStudent	EnrollForDegree="graduate degree"
PhdStudent	EnrollForDegree="phd"

IC from their type sort. All quasi-type sorts are rigid, that is, every individual of a quasi-type sort is always an individual of that quasi-type sort. For example, every enterprise is always an enterprise but neither a university nor an association, untill it ceases from being an enterprise.

For role sorts, I describe their EDRs together with dependent sorts in Table 4.4. In this domain, `FacultyMember` is defined as a person who employes in a university, that is, there is no association member without any association and if a member exists then there must be an association. Thus, `AssociationMember` and `Association` are externally dependent to each other. Since all role sorts are anti-rigid, the individuals of every role sort is not stable and they change world by world. Moreover, an individual can be an instance of more than one role sort such as a person can be both a professor and a president in a possible world.

Table 4.5 list the phase sorts and their CCs. Also, phase sorts are anti-rigid and they changes by worlds. They are the partitions of a certain super-sort and thus they are disjoint to each other. For example, the set of associate professors are always disjoint from the set of professors in every possible world, that is, if a person is employed as an associate professor then he/she cannot be a professor in the same world. `AssociateProfessor` and `Professor` are the serialized stages of `FacultyMember`. Then, concept-level properties of each sort can be defined as follows:

1. For any type sort $s \in S_{type}$, $P^C(s) = \{\text{sort-category} = \text{“}type\text{”}, ownIC = \iota_s\}$.
2. For any quasi-type sort $s \in S_{quasi-type}$, $P^C(s) = \{\text{sort-category} = \text{“quasi-type”}, CVA = p_a\}$.

3. For any role sort $s \in S_{role}$, $P^C(s) = \{\text{sort-category} = \text{“}role\text{”}, ownIC = p_d\}$.

4. For any phase sort $s \in S_{phase}$, $P^C(s) = \{\text{sort-category} = \text{“}phase\text{”}, ownIC = p_c\}$.

The set of individual-level properties, P^D, with specific domains and ranges are also defined for `Research-Community.owl`. I list some of them in Table 4.6. The ranges of some properties are restricted with allowed values. For example, only “female”, and “male” are allowed in the range of `hasGender`. According to the given properties, the meaning of each sort $s \in S$ is interpreted unambiguously. As an example, every `PhdStudent` is a student who enrolls for PhD degree, and has a research work and a supervisor for that. The set of individual-level properties that belong to PhdStudent through subsumption relationships such that $PhdStudent \sqsubseteq Student \sqsubseteq Person$ is:

$$P^D(PhdStudent) = \{hasPName, hasGender, hasBirthDate, hasPEmail, enrollIn,$$

$$hasFingerprint, hasStudentID, enrollForDegree, \text{is-supervised}, hasRPub\}.$$

PhdStudent has an IC, `hasFingerPrint`, which is carried from `Person`. Also, the description of `PhdStudent` can be written in terms of DL notation [52], as follows.

$$PhdStudent \equiv Student \sqcap \forall(hasPName \sqcap hasGender \sqcap hasBirthDate \sqcap$$

$$hasPEmail \sqcap enrollIn \sqcap =1 hasFingerPrint \sqcap hasStudentID \sqcap$$

$$enrollForDegree \sqcap \text{is-supervised} \sqcap hasRPub)$$

Recall that deciding whether a sort is a type sort or another kind of sort, does not fully depend on the common sense of its name. More precisely, a sort is classified according to the properties and constraints defined for it. Two similar domain ontologies may have different taxonomies with some common sorts. I claim that the sort categories of two semantically similar (or equal) sorts in both ontologies should be the same. On the contrary, if two sorts belong to different kinds of sort category, then they cannot be the same sort, because their semantics have different classification constraints such as ownIC, CVA, EDR, or CC. However, several forms of semantic heterogeneity can appear between two semantically similar sorts.

Suppose that Figure 4.15 is a sub-taxonomy of another ontology which is conceptualized for a similar domain like `Research-Community.owl`. There may be some corresponding sorts between those two ontologies. For example, `DoctoralStudent` can be a correspondence of `PhdStudent` from `Research-Community.owl` because they are classified in the same sort category. However, it is visible that both sorts have terminological heterogeneity and taxonomical heterogeneity.

In summary, the main idea of EnOntoModel is that domain concepts are represented as sorts and classified into four groups according to the philosophical notions. Then, concept-level properties enrich the semantics of sorts. EnOntoModel provides a number of advantages as follows:

- Individuals of a sort are countable and identifiable. Thus, each individual holds a unique IC value, and the sort possesses at least one IC directly or indirectly.

Table 4.6: Some individual-level properties of `Research-Community.owl`

Property	Domain	Range
hasRProjName	ResearchProject	String
hasRProjOrganizer	ResearchProject	LegalOrganization
hasRProjType	ResearchProject	String
hasEstablishedDate	ResearchProject	Date
hasRProjManager	ResearchProject	ProjectManager
MemberOfRProj	ResearchProject	∃ Person
hasRProjSeminar	ResearchProject	Seminar
hasSTName	SoftwareTool	String
hasSTDeveloper	SoftwareTool	LegalOrganization
hasSTCategory	SoftwareTool	String
hasRPubTypeID	ResearchPublication	String
hasRPubCategory	ResearchPublication	String
hasRPubTitle	ResearchPublication	String
hasRPubAuthor	ResearchPublication	Author
hasPublishingDate	ResearchPublication	Date
hasEventType	Event	String
hasEventPlace	Event	Location
hasEventOrganizer	Event	LegalOrganization
hasEventStartDate	Event	Date
hasEventCloseDate	Event	Date
hasEventFunder	Event	LegalOrganization
hasEventContact	Event	String
hasLatitude	Location	Integer
hasLongitude	Location	Integer
hasOrgName	LegalOrganization	String
hasAddress	LegalOrganization	String
hasLocation	LegalOrganization	{hasLatitude, hasLongitude}
hasOrgGroup	LegalOrganization	String
hasHomePage	LegalOrganization	URL
headOf	LegalOrganization	President
hasPName	Person	String
hasGender	Person	{female, male}
hasPEmail	Person	Email
hasBirthDate	Person	Date
hasFingerPrint	Person	FingerPrint
EmployIn	Employee	LegalOrganization
hasEmploymentPosition	Employee	String
hasFacultyPosition	FacultyMember	String
supervise	FacultyMember	Student
is-supervised	Student	FacultyMember
enrollIn	Student	University
hasStudentID	ResearchStudent	Integer
enrollForDegree	MasterStudent	“graduate degrees”
enrollForDegree	PhdStudent	“PhD”
hasRPub	PhdStudent	ResearchPublication

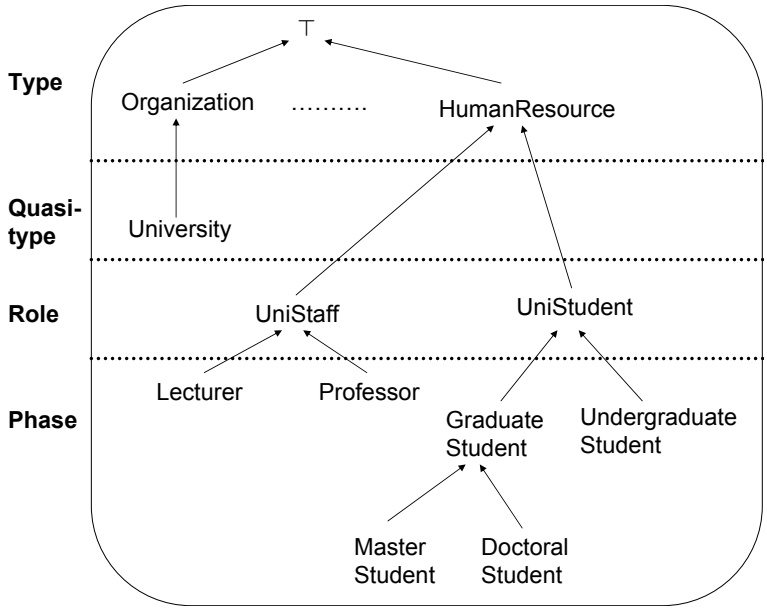

Figure 4.15: An example of the similar domain ontology

- Domain concepts are structured as taxonomies in the form of subsumption relationships. By EnOntoModel, conceptual levels among domain concepts are further divided in terms of sort categoriess such as type, Quasi-type, role, and phase. Because these sort categories are disjoint to each other, they aid in in determining the scopes of correspondences in ontology matching. Said another way, EnOntoModel intends to accelerate the matching process.

- By the conceptual constraints, enriched ontologies can be verified for conceptual consistency.

- Each sort possesses a specific property by the conceptual constraints, by which, correspondence between two similar sorts can be detected efficiently by using a restricted property instead of all available properties.

4.4 Implementation of a Sortal Meta-class Ontology

For the usability of my enrichment theory, how users can enrich their ontologies based on EnOntoModel becomes critical. I solved this by implementing a sortal meta-class ontology named `sort.owl` as an open source interface of EnOntoModel. In this section, I explain the implementation framework of the sortal meta-class ontology using Protégé OWL API and a representation of this ontology in OWL-DL.

Note that a meta-class is a specification of classes and it supports runtime access to meta-data associated with classes. Each meta-data element is referred to a property. In Protégé, `meta-class` is a frame interface that is used to define user-defined classes in ontologies. There are two basic reasons why I selected Protégé-OWL editor for the implementation of the sortal meta-class ontology.

1. Protégé-OWL editor allows creation of customized *meta-classes*.

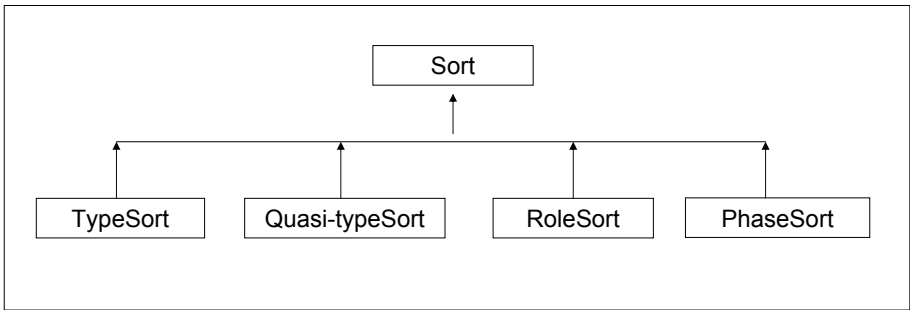

Figure 4.16: A structure of sortal meta-classes

2. In addition, Protégé provides Protégé Axiom Language (PAL) to define internal constraints, and to embed these constraints in OWL format.

4.4.1 Purpose and Scope

There are two purposes in implementing a sortal meta-class ontology.

- The first purpose is to support for the usability of EnOntoModel.
- The second purpose is to provide conceptual analysis of the enriched ontologies.

The basic motivation of my enrichment is to clarify and enrich the semantics of concepts for the issue of semantic heterogeneity. The role of ontologies is to provide a well-defined structure of domain knowledge that acts as the heart of any system of knowledge representation on that domain for the purposes of reasoning, knowledge sharing, and integration. Thus, it is also essential to verify taxonomies that provide a substantial structural information of ontologies. Properly structured taxonomies help bring substantial order to elements of a model and play a critical role in reuse and integration tasks. Improperly structured taxonomies have the opposite effect, making models confusing and difficult to reuse and integrate. I define conceptual analysis as follows:

> **Conceptual analysis** *is a framework for cleaning the taxonomic structure of ontologies by validating subsumption relationships.*

It is also known as *ontological analysis* [147]. By the second purpose, I embed a system of conceptual analysis in the meta-class ontology, using Protégé Axiom Language (PAL) that is an internal language of protégé to author constraints which axiomatize subsumption relationships between super-classes and sub-classes logically.

The scope of this meta-class ontology is bounded to sorts, and the ontology employs some conceptual constraints in order to maintain subsumption consistency.

4.4.2 Design and Implementation

There are two major components in the meta-class ontology: (a) specification of sortal meta-classes, and (b) axiomatization for conceptual analysis.

Table 4.7: Meta-classes and their properties

Meta-class	Properties
`TypeSort`	`sort-category` has "type"
	nameofOwnIC
`Quasi-typeSort`	`sort-category` has "quasi-type"
	nameOfCVA
`RoleSort`	`sort-category` has "role"
	nameOfEDR
`PhaseSort`	`sort-category` has "phase"
	nameOfCC

Specification of Sortal Meta-classes

Meta-class ontology consists of five meta-classes labeled `Sort`, `TypeSort`, `Quasi-type Sort`, `RoleSort`, and `PhaseSort`, as shown in Figure 4.16. `Sort` meta-class is defined as the root of other four meta-classes. It has a datatype property named `sort-category` to define the category of a certain sort. Then, four meta-classes are defined as the sub-classes of `Sort` meta-class and they will be mainly used to represent ontological concepts as sorts. By property inheritance, `sort-category` is inherited to all sub-classes.

I describe the specification of four meta-classes in Table 4.7. For each of four meta-classes, there is an additional property such as `nameOfOwnIC`, `nameOfCVA`, `nameOfEDR`, and `nameOfCC`. This means, for any instance class of `TypeSort` meta-class, an ownIC must be necessarily defined. Thus, the name of ownIC is recorded as a property in `TypeSort` meta-class in order to utilize in the matching process. It is also similar for other meta-classes.

Figure 4.17 shows a screenshot of the specification of `RoleSort` meta-class in Protégé. This specification states that `RoleSort` is a sub-class of `Sort`, each instance of `RoleSort` meta-class has at least one EDR, and the value of `sort-category` is restricted by "role" for every role sort.

Axiomatization for Conceptual Analysis

I define five PAL constraints in the meta-class ontology for the purpose of conceptual analysis. These constraints are written based on (a) ontological assumption "anti-rigid sort never subsumes rigid sorts", and (b) the disjointness between rigid sorts and between anti-rigid sorts. The names and meanings of these PAL constraints are:

1. `notRoleToType`: a role sort never subsume a type sort;
2. `notQuasi-typeToType`: a quasi-type sort never subsume a type sort;
3. `notPhaseToType`: a phase sort never subsume a type sort;
4. `notRoleToQuasi-type`: a role sort never subsume a quasi-type sort; and
5. `notPhaseToQuasi-type`: a phase sort never subsume a quasi-type sort.

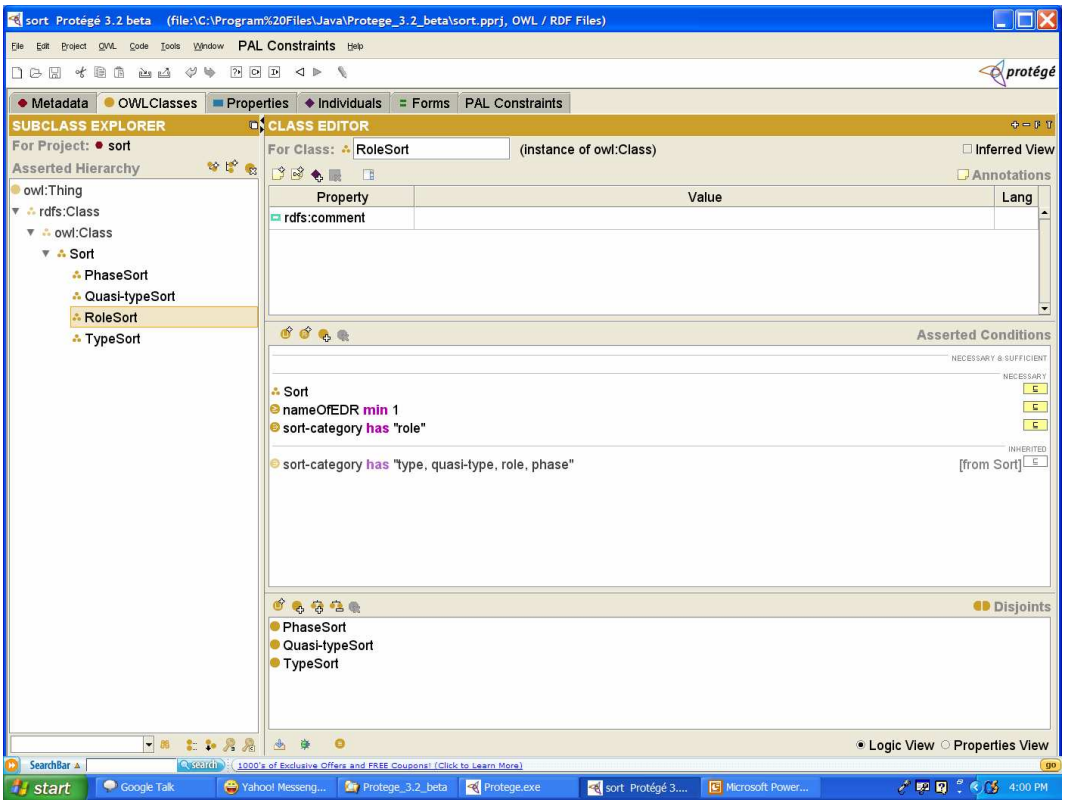

Figure 4.17: A specification of RoleSort meta-class in Protégé

Table 4.8: The meanings of some PAL keywords

PAL keywords	**Meaning**
`super`	super-class
`sub`	sub-class
`?sub`	sub-class variable
`subclass-of ?sub ?super`	?sub is the sub-class of ?super
`forall`	$\forall$
`exists`	$\exists$
`not`	$\neg$
`and`	$\wedge$
`or`	$\vee$
`own-slot-not-null`	if the given property (or slot) is not null

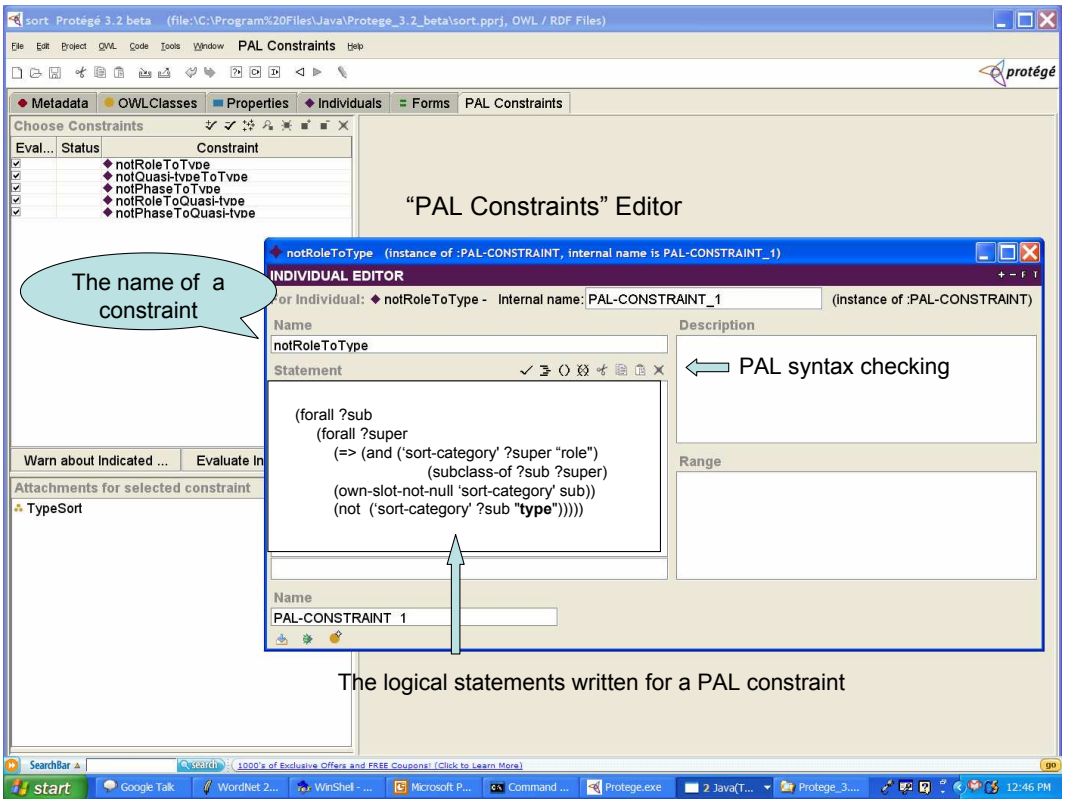

Figure 4.18: The screenshot of "PAL constraint" editor

In order to understand well the meaning of a PAL constraint, I first express the meanings of some PAL keywords in Table 4.8. To write a PAL constraint, we need to make sure `PAL constraints` tab is successfully installed and visible in the user interface of Protégé editor. A screenshot of PAL constraint edior is shown in Figure 4.18. The editor provides not only for writing the logical statements of a constraint, but also for syntax checking of the statements.

The meaning of the PAL constraint written in Figure 4.18 is "every sub-class which has sort category "type" can not be subsumed by a super-class that has sort category "role". Briefly, that states "a role sort never subsume a type sort". The statements of all five constraints are described in Figure 4.19. I built meta-class ontology `sort.owl`, and uploaded it in Protégé ontology library[29] as an open source. Thus, users can download the meta-class ontology via Protégé ontology library. The complete source code is attached in Appendix.

4.5 Development of Semantically-enriched Ontologies

In this section, I present the steps to develop semantically-enriched ontologies based on EnOntoModel. I demonstrate the implementation framework of EnOntoModel-based ontologies using Protégé OWL API and a representation of these ontologies in OWL. The major steps of semantic enrichment process are illustrated in Figure 4.20.

1. First, users need to open a new project in Protégé for a source ontology called

[29] `http://protege.cim3.net/cgi-bin/wiki.pl?ProtegeOntologiesLibrary`

```
<protege:PAL-CONSTRAINT rdf:ID="PAL-CONSTRAINT_1">
<protege:PAL-STATEMENT rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                    (forall ?sub  (forall ?super
                                        (=> (and (‘sort-category' ?super "role“
                                        (subclass-of ?sub ?super)
                                         (own-slot-not-null ‘sort-category' ?sub))
                   (not   (‘sort-category' ?sub "type")))))
</protege:PAL-STATEMENT>
<protege:PAL-NAME rdf:datatype="http://www.w3.org/2001/XMLSchema#string">notRoleToType
</protege:PAL-NAME>
</protege:PAL-CONSTRAINT>
<protege:PAL-CONSTRAINT rdf:ID="PAL-CONSTRAINT_2">
<protege:PAL-STATEMENT rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                   (forall ?sub  (forall ?super
                                        (=> (and (‘sort-category' ?super "quasi-type"
                                        (subclass-of ?sub ?super)
                                        (own-slot-not-null ‘sort-category' ?sub))
                   (not   (‘sort-category' ?sub "type")))))
</protege:PAL-STATEMENT>
<protege:PAL-NAME rdf:datatype="http://www.w3.org/2001/XMLSchema#string">notQuasi-typeToType
</protege:PAL-NAME>
</protege:PAL-CONSTRAINT>
<protege:PAL-CONSTRAINT rdf:ID="PAL-CONSTRAINT_3">
<protege:PAL-STATEMENT rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                   (forall ?sub  (forall ?super
                                         (=> (and (‘sort-category' ?super "phase"
                                         (subclass-of ?sub ?super)
                                         (own-slot-not-null ‘sort-category' ?sub))
                    (not     (‘sort-category' ?sub "type")))))
</protege:PAL-STATEMENT>
<protege:PAL-NAME rdf:datatype="http://www.w3.org/2001/XMLSchema#string">notPhaseToType
</protege:PAL-NAME>
</protege:PAL-CONSTRAINT>
<protege:PAL-CONSTRAINT rdf:ID="PAL-CONSTRAINT_4">
<protege:PAL-STATEMENT rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                   (forall ?sub  (forall ?super
                               (=> (and (‘sort-category' ?super "role"
                                         (subclass-of ?sub ?super)
                                         (own-slot-not-null ‘sort-category' ?sub))
                     (not  (‘sort-category' ?sub "quasi-type")))))
</protege:PAL-STATEMENT>
<protege:PAL-NAME rdf:datatype="http://www.w3.org/2001/XMLSchema#string">notRoleToQuasi-type
</protege:PAL-NAME>
</protege:PAL-CONSTRAINT>
<protege:PAL-CONSTRAINT rdf:ID="PAL-CONSTRAINT_5">
<protege:PAL-STATEMENT rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                    (forall ?sub (forall ?super
                               (=> (and (‘sort-category' ?super "phase"
                                          (subclass-of ?sub ?super)
                                          (own-slot-not-null ‘sort-category' ?sub))
                     (not  (‘sort-category' ?sub "quasi-type")))))
</protege:PAL-STATEMENT>
<protege:PAL-NAME rdf:datatype="http://www.w3.org/2001/XMLSchema#string">notPhaseToQuasi-type
</protege:PAL-NAME>
</protege:PAL-CONSTRAINT>
```

Figure 4.19: Five PAL constraints in the meta-class ontology

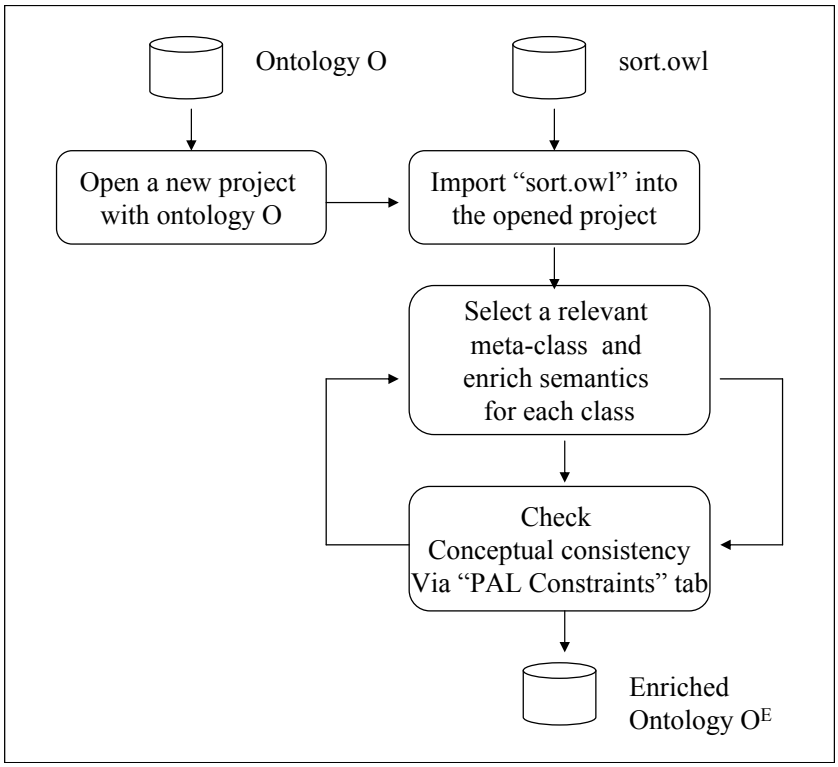

Figure 4.20: Major steps of semantic enrichment process

O. After opening a new project with ontology O, the classes and properties (both object and datatype properties) can be viewed via a click on `OWL Classes` and `Properties` respectively as shown in Figure 4.21. `Object` tab is for individual properties and `Datatype` tab is for datatype properties.

2. Second, it is necessary to import `sort.owl` into the opened project via the import service of Protégé, as shown in Figure 4.22.

3. Third, the meta-class of each ontological class needs to be changed from standard class, `owl:class`, to one of the sortal meta-classes via `change metaclass` option of Protégé as shown in Figure 4.23. For this selection, users need the background knowledge of sort classification. By the selection of meta-class, the sort category value of each ontological class will be assigned automatically. Then, the user needs to assert necessary concept-level properties together with individual-level properties, according to the constraints given in Table 4.7.

4. Fourth, the conceptual consistency of semantic enrichment can be evaluated by invoking the PAL constraints defined in `sort.owl`, via the `PAL constraints` tab of Protégé. For this verification, users need to run a DIG reasoner: `Racer`[30] or `Pellet`[31]. Racer is the default reasoner in Protégé. The reasoner URL of `Racer` is `http://localhost:8080`. In the case of `Pellet`, users need to change the reasoner URL to `http://localhost:8081`, via `Preferences` option from `OWL`

[30] `http://www.sts.tu-harburg.de/ r.f.moeller/racer/`
[31] `http://www.mindswap.org/2003/pellet/`

menu. I list the steps of conceptual analysis through `PAL constraints` tab, below.

(a) Add all constraints to the list (click on third button from `Choose Constraints` menu bar).

(b) Select the constraints you want to verify, by clicking a checkbox for each constraint.

(c) Select taxonomies and sorts to be evaluated for the selected constraints, via a click on `Attachments for selected constraints`. If users intend to verify all defined sorts, this step will not be needed.

(d) Then, execute the selected PAL constraints via a click on `Evaluate selected constraints`, as shown in Figure 4.24.

If there are some sorts which violate some PAL constraints, a list of the sorts will be displayed on the right-hand side. Then, users can view and correct them untill they are consistent. An iterated process may need between Steps 3 and 4.

5. Finally, the semantically-enriched ontology, O^E, can be successfully generated in OWL via `Show RDF/XML source code` option from `Code` menu. Then, enriched-version O^E can be used for the later matching process. I show a part of enriched ontology `Research-Community.owl` in OWL source code in Figure 4.25, where each sort $s \in S$ is represented as an instance class of a certain sortal meta-class, for example, `ResearchProject` is an instance of meta-class TypeSort, `Enterprise` is an instance of meta-class Quasi-typeSort, `Employee` is an instance of meta-class RoleSort, and `Professor` is an instance of meta-class PhaseSort. The complete source code of `Research-Community.owl` is uploaded in Protégé ontology library[32].

[32] `http://protege.cim3.net/cgi-bin/wiki.pl?ProtegeOntologiesLibrary`

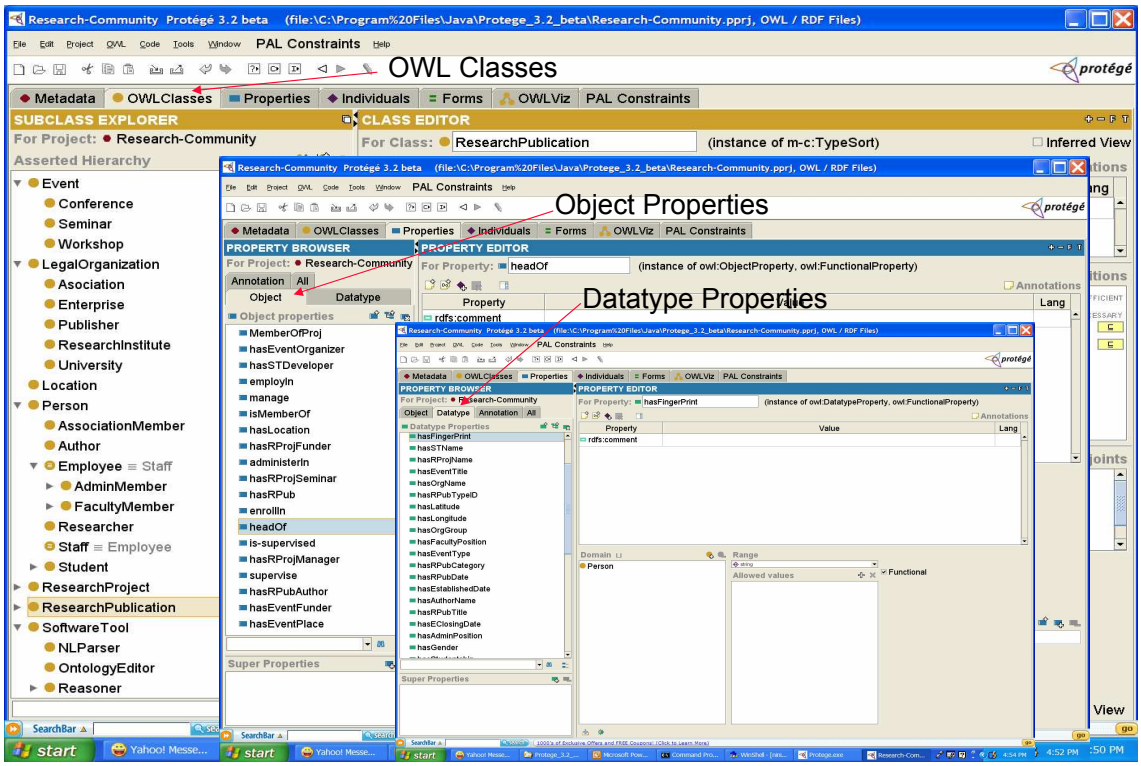

Figure 4.21: A view of classes and properties in Proteégé

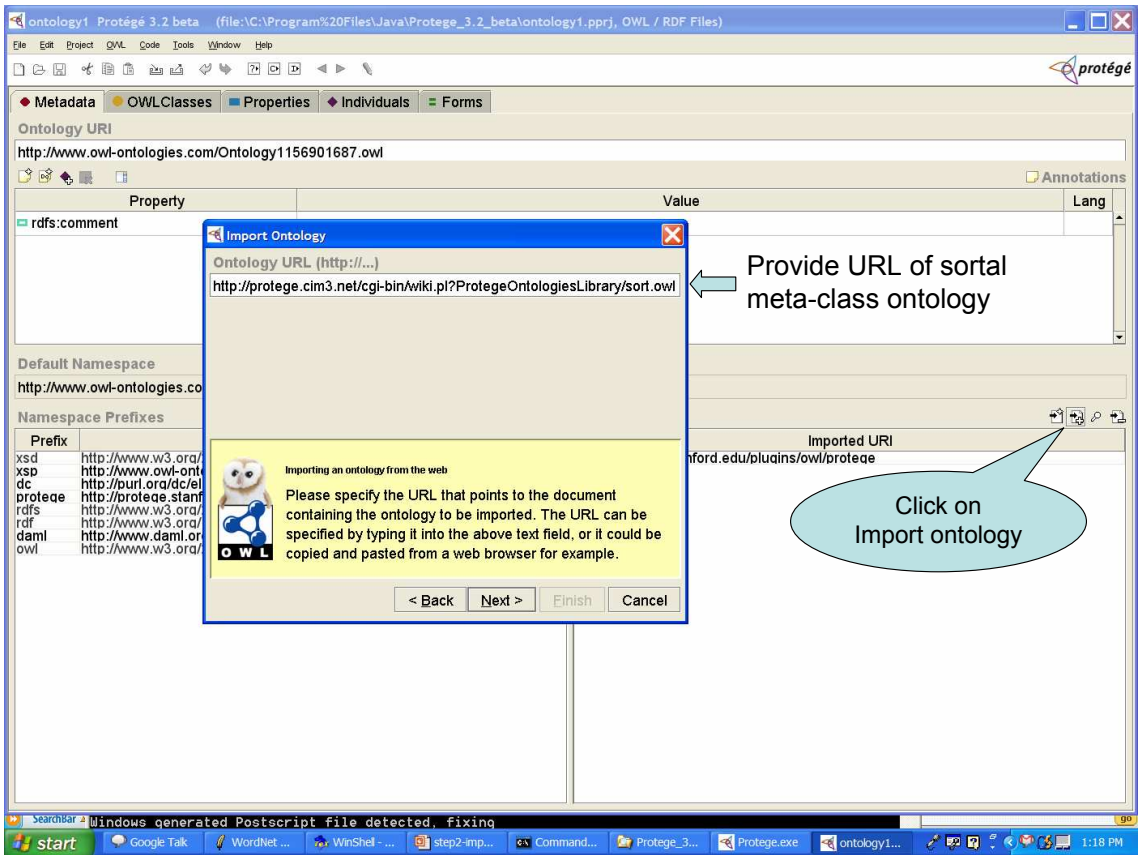

Figure 4.22: A screenshot for importing ontology

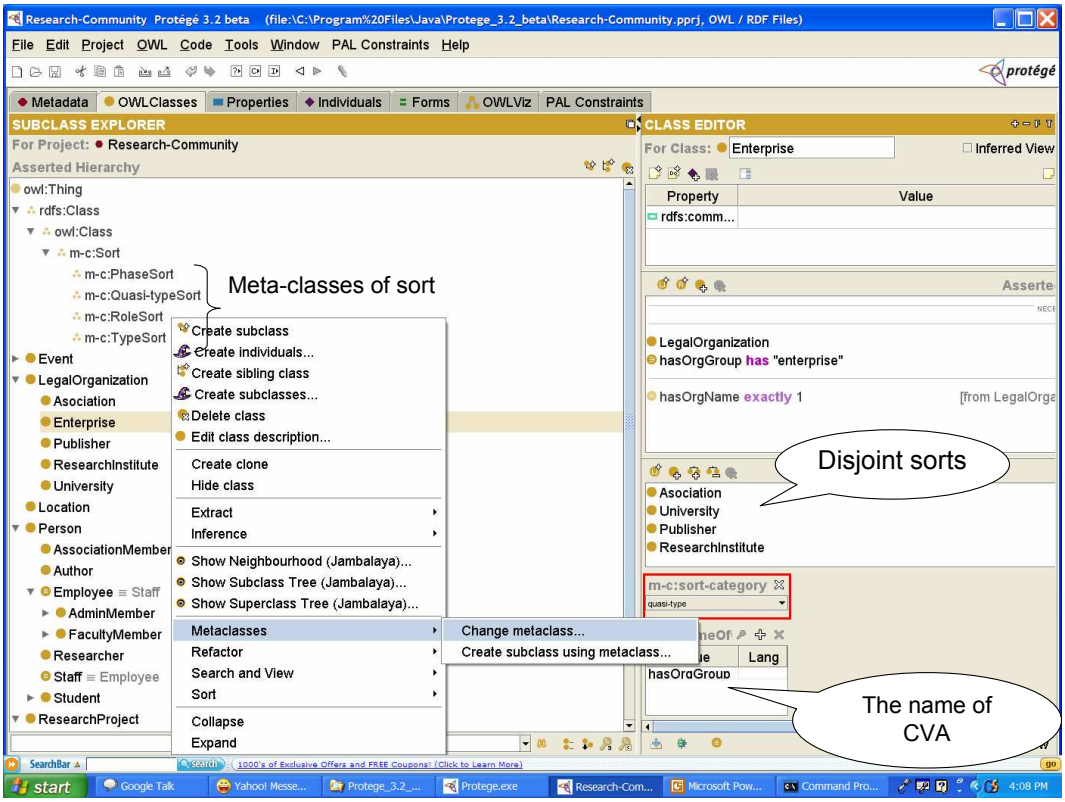

Figure 4.23: A screenshot of semantic enrichment in Protégé

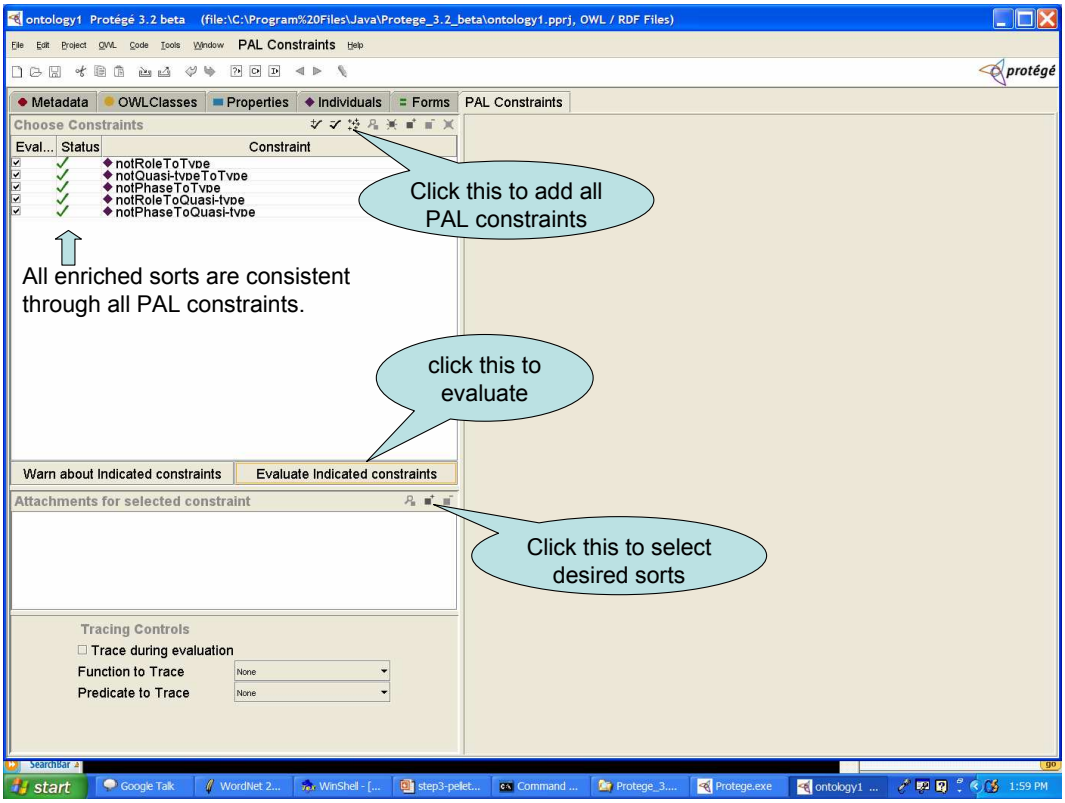

Figure 4.24: A screenshot of conceptual analysis using PAL constraints

```
<rdf:RDF xml:base="http://www.owl-ontologies.com/Research-Community.owl">
<owl:Ontology rdf:about=""><owl:importsrdf:resource="http://protege.stanford.edu/plugins/owl/protege"/>
</owl:Ontology>
……
<m-c:TypeSort rdf:ID="ResearchProject">
<m-c:sort-category rdf:datatype="http://www.w3.org/2001/XMLSchema#string">type</m-c:sort-category>
      <m-c:nameOfOwnIC rdf:datatype="http://www.w3.org/2001/XMLSchema#string">hasRProjName
</m-c:nameOfOwnIC>
<m-c:nameOfOwnIC rdf:datatype="http://www.w3.org/2001/XMLSchema#string">hasRProjOrganizer
</m-c:nameOfOwnIC>
<rdfs:subClassOf>
      <owl:Restriction>
            <owl:onProperty><owl:DatatypeProperty rdf:ID="hasRProjName"/></owl:onProperty>
            <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
      </owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#Event"/><owl:disjointWith rdf:resource="#Person"/>……

<m-c:Quasi-typeSort rdf:ID="Enterprise">
<m-c:sort-category rdf:datatype="http://www.w3.org/2001/XMLSchema#string">quasi-type</m-c:sort-category>
<m-c:nameOfCVA rdf:datatype="http://www.w3.org/2001/XMLSchema#string">hasOrgGroup</m-c:nameOfCVA>
<rdfs:subClassOf rdf:resource="#LegalOrganization"/>
      <owl:Restriction>
            <owl:onProperty><owl:DatatypeProperty rdf:about="#hasOrgGroup"/></owl:onProperty>
            <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string">enterprise</owl:hasValue>
      </owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#University"/><owl:disjointWith rdf:resource="#Association"/>……

<m-c:RoleSort rdf:ID="Employee">
      <owl:equivalentClass><m-c:RoleSort rdf:ID="Staff">
      <rdfs:subClassOf rdf:resource="#Person"/>
<m-c:sort-category rdf:datatype="http://www.w3.org/2001/XMLSchema#string">role</m-c:sort-category>
<m-c:nameOfEDR rdf:datatype="http://www.w3.org/2001/XMLSchema#string">employIn</m-c:nameOfOwnIC>
<owl:ObjectProperty rdf:ID="employIn">
      <rdfs:domain rdf:resource="#Employee"/>
      <rdfs:range rdf:resource="#LegalOrganization"/>
</owl:ObjectProperty>

<m-c:PhaseSort rdf:ID="Professor">
<rdfs:subClassOf rdf:resource="#FacultyMember"/>
<m-c:sort-category rdf:datatype="http://www.w3.org/2001/XMLSchema#string">phase</m-c:sort-category>
<m-c:nameOfCC rdf:datatype="http://www.w3.org/2001/XMLSchema#string">hasFacultyPosition
</m-c:nameOfOwnIC>
<rdfs:subClassOf>
      <owl:Restriction>
            <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string">professor</owl:hasValue>
            <owl:onProperty rdf:resource="#hasFacultyPosition"/>
      </owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#AssociateProfessor"/>
……
```

Figure 4.25: An example of enriched concepts in OWL

Chapter 5

EnOntoModel-based Ontology Matching

In this chapter, my idea of matching between semantically-enriched ontologies is presented. The proposed matching method is intended to support semantic interoperability through heterogeneous ontologies. Parts of this chapter have been published before [155, 157].

5.1 Overview

My approach of ontology matching is based on an enrichment process. Figure 5.1 depicts an overview of enrichment-based matching. In my approach, firstly ontologies need to enrich their semantics. Then matching process finds semantic correspondences between two enriched ontologies. As the focal point of this ontology matching is for semantic heterogeneity, an enrichment process prior to matching process intends to clarify the semantics of concepts and their taxonomic structures. Here, recall that my definition of ontology matching together Figure 5.2.

> *Given two ontologies O_1 and O_2, ontology matching means for each concept (node) in ontology O_1, a corresponding concept (node) which has the same or similar semantics is discovered in ontology O_2, and vice verse.*

There are several purposes for performing ontology matching. We can divide them into two general cases: *query answering* for information exchange, retrieval, or integration, and *ontology merging*. For query answering, a matching process is executed only for desired

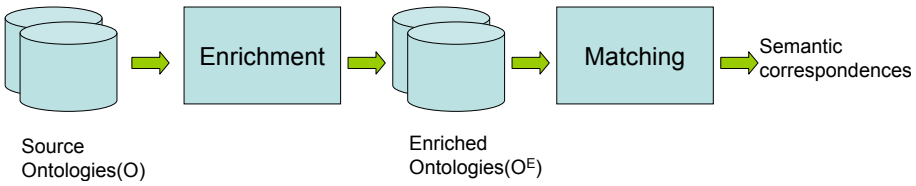

Figure 5.1: An overview of enrichment-based matching

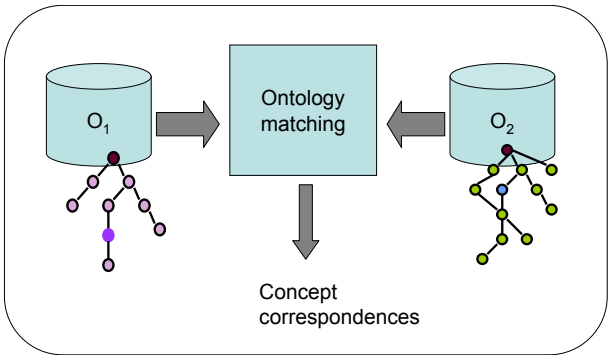

Figure 5.2: A view of ontology matching

concepts (or candidate sorts). The Web portals[1] and softbots[2] also use this for information searching and gathering. In the case of merging, the process needs to discover all possible correspondences between two ontologies. Then, some articulation rules are needed to integrate corresponding concepts in a balanced way. The objective of ontology merging is particularly for knowledge reuse in ontology management. For either purpose, ontology matching comprises how a class from one ontology can be semantically matched to a class of the next ontology in an automatic or semi-automatic way. In this chapter, I will mainly discuss EnOntoModel-based ontology matching for query answering.

In matching for query answering, there are generally two settings [138]:

- Peer-to-Peer Matching: Matching is performed between two local (or adjacent) ontologies.
- Matching via a Global Ontology: Matching is performed among local ontologies via a global ontology.

Figure 5.3 illustrates these settings where symbols 'L', 'G', and 'Q', denote for local ontologies, global ontology, and query process, respectively. Global ontology is an integrated view of conceptualization that encompasses in all local ontologies. Because my focus is for matching between two heterogeneous ontologies in order to provide the interoperability between two information systems where the ontologies are used, my matching architecture is similar to Peer-to-Peer approach. If each information system employs multiple local ontologies, then a global ontology must be already constructed and some articulation rules between the global ontology and local ontologies must be well-defined. For semantic interoperability, a matching process needs to perform between two global ontologies. There after, an additional matching step will need to run between a global ontology and a certain local ontology using the defined articulation rules, to accomplish interoperability.

Concerning semantic correspondence between concepts, there are two possible relationships. They are subsumption relationship ($\sqsubseteq$) and equality ($\equiv$). In this research work, my matching focuses only for equality and the term 'correspondence' means for semantically-equality between two concepts. Thus, I define the equality between two sorts as follows:

[1]A Web portal is a site on the World Wide Web that typically provides personalized capabilities to its visitors, providing a pathway to other content.

[2]A Software robot that aggregates Web search services for users.

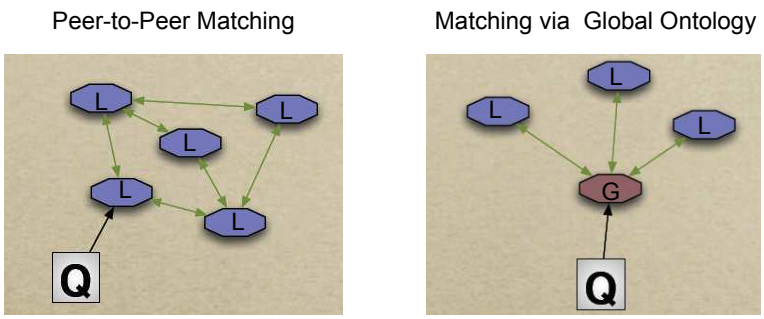

Figure 5.3: Two mapping settings in query answering

Definition 25 (Sort Equality) *For two sorts s and s', there is an equality:*

$$s \equiv s' \quad \textit{iff} \quad \forall x[p_s(x) \leftrightarrow p_{s'}(x)].$$

5.2 Matching Architecture

A general architecture of EnOntoModel-based ontology matching is given in Figure 5.4. Suppose that there are two information systems driven in a common or overlapped domain. The systems are constructed using two heterogeneous ontologies: O and O'. Here, I concern wide-scale heterogeneity between these ontologies, that is, several kinds of heterogeneity such as terminological heterogeneity, taxonomical heterogeneity, schematic heterogeneity, and instantiation heterogeneity, may exist in the ontologies. Recall that, for two concepts, terminological heterogeneity occurs when they have different names, taxonomical heterogeneity occurs when they have different subsumption structures, schematic heterogeneity comes when they are defined with different sets of individual-level properties, and instantiation heterogeneity appears when they are instantiated with different sets of individuals.

In the architecture, I assume that source ontologies are populated, that is, some individuals are defined as the instances of each concept/class. In practical cases, information systems employ ontologies together with application databases. Therefore, the concept of a populated ontology is fundamental to several approaches to ontology mapping.

A requirement of this mapping architecture is ontologies need to enrich their semantics in terms of EnOntoModel. I have presented the enrichment framework of ontologies using Protégé OWL API. Following to the enrichment steps described in Figure 4.20, the enriched versions of source ontologies, O^E and O'^E, are generated. Each enriched ontology consists of a set of sorts (S), a taxonomic structure ($\langle S, \sqsubseteq \rangle$), a set of concept-level properties P^C, a set of individual-level properties P^D, and axioms A. The major characteristics of enriched ontologies are:

- Domain concepts are divided into four disjoint sort categories and conceptual consistency among the sorts is verified by using some philosophical constraints.
- The semantics of each sort, $s \in S$, is defined by a set of individual-level properties ($P^D(s)$), as well as a set of concept-level properties ($P^C(s)$).

For the purpose of information sharing and integration, semantic interoperability is required between the systems. In order to achieve this semantic interoperability, the re-

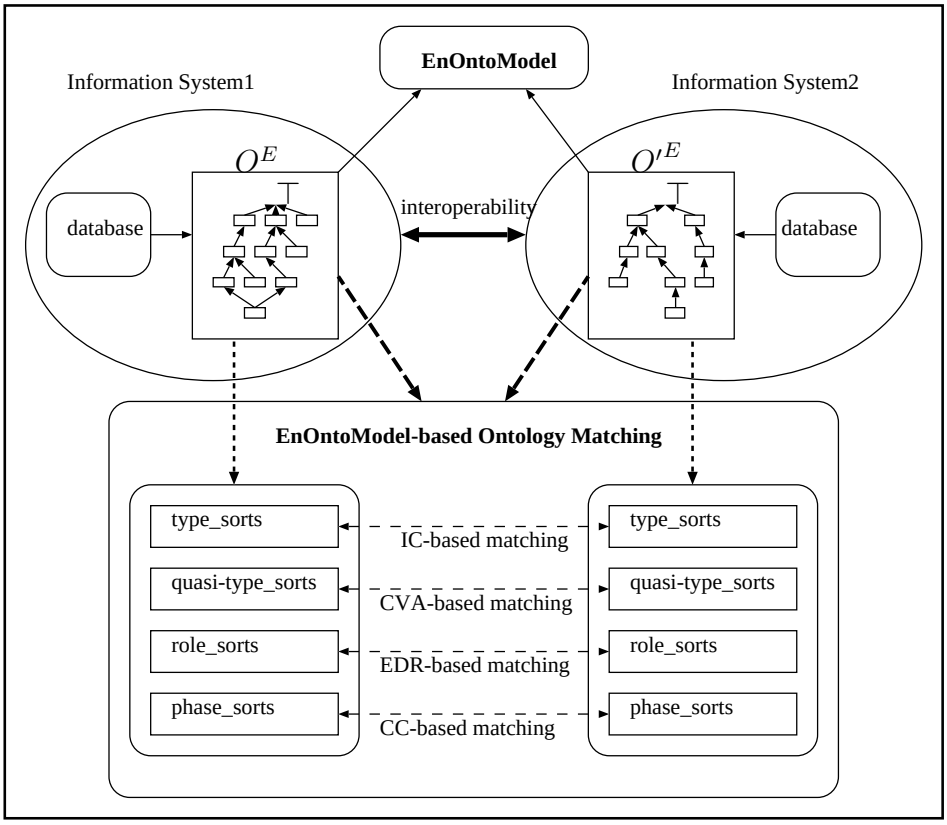

Figure 5.4: A general architecture of EnOntoModel-based ontology matching

sponsibility of ontology matching is to find semantically similar sorts between two ontologies. Two enriched ontologies, O_E and O'_E together with related databases, are the input of EnOntoModel-based ontology matching process. Since sorts defined in each enriched-ontology are systematically divided into four groups, the taxonomy of each ontology can be viewd as a four-layered sorts, as shown in Figure 5.4.

A novel idea of this matching architecture is, for a query, direct matching can be driven between sorts defined in the same classification group, in stead of matching to all sorts by traversing taxonomies completely. This is an advantage over other existing mapping methods. This advantage comes from the following postulate.

> *For ontology matching, I claim that there is no semantic correspondence between rigid sorts and anti-rigid sorts, nor between rigid sorts (type sorts and quasi-type sort) nor between anti-rigid sorts (role sorts and phase sorts), because their modality and classification constraints are different from each other.*

According to the above postulate, there is no chance of sort correspondences between different sort categories, that is, a type sort cannot have a correspondence to a quasi-type sort, a role sort, or a phase sort. It is similarly for quasi-type sorts, role sorts, and phase sorts. Figure 5.5 illustrates the above postulate and I apply that as matching heuristics in my method. Consequently, it can flatten iterations of a matching process and possibly reduce complexity. A detailed technique of EnOntoModel-based matching method is explained in the next section.

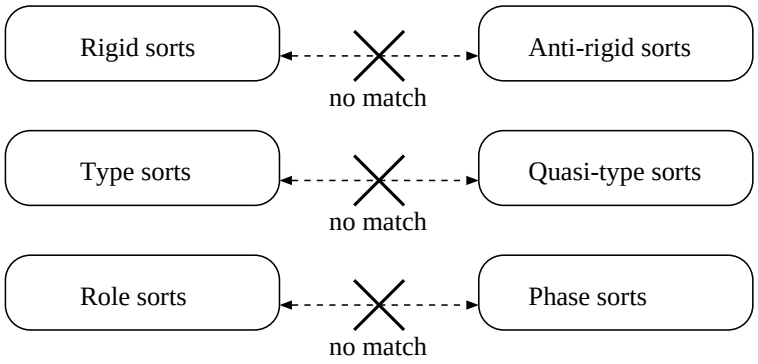

Figure 5.5: EnOntoModel-based matching heuristics

5.3 Matching Method

Let $O^E = \langle S, \sqsubset, P^C, P^D, A\rangle$ and $O'^E = \langle S', \sqsubset', P'^C, P'^D, A'\rangle$ be the logical view of two enriched heterogeneous ontologies. In the matching method, I consider a mapping function $f : s \in S \Longrightarrow s' \in S'$ where f performs a matching process to find a semantically corresponding sort s' for s. f is further divided into four sub-functions: (1)type sort matching f_{type}, (2) quasi-type sort matching $f_{quasi-type}$, (c) role sort matching f_{role}, and (4) phase sort matching f_{phase}. The process flow diagram of matching function f is shown in Figure 5.6. Regarding a query processing, a candidate sort $s \in S$ will be the input of the matching function f and an appropriate sub-function is decided based on the sort category of s. Finally, the result of correspondence will be retrieved from a specific sub-function. If there exist a sort, $s' \in S'$, which is semantically equivalent to s, then the related information will be executed as the query definition. In this work, I present the theoretical aspect of my matching method regarding enriched ontologies. Before a detailed presentation of matching functions, I first discuss why ICs are useful for matching between sorts.

5.3.1 Why ICs are useful for matching between sorts?

There are some ways for us to regard that two sorts given in different ontologies are semantically equivalent. The first possible candidate of such equality can occur when two given sorts have the same set of individuals, that is,

$$\text{if } \ [\![s]\!] = [\![s']\!] \text{ then } s \equiv s' \tag{5.1}$$

where $[\![s]\!]$ denotes for a set of individuals instantiated to sort s. When two sorts are satisfied by this condition of equality between their extensional knowledge, it is easy to determine their equality. However, in practice, we cannot expect the exactly same instantiation in open and incomplete domains.

Example 10 *Suppose there are two well-defined ontologies* $O = \langle S, \sqsubset, P^D, A\rangle$ *and* $O' = \langle S', \sqsubset', P'^D, A'\rangle$. *Their taxonomies are as depicted in Figure 5.7. And assume that the following sets of properties are defined for each sort in each ontology.*

In ontology O*,*

$$P^D(\texttt{WebResource}) = \{hasURL\}$$

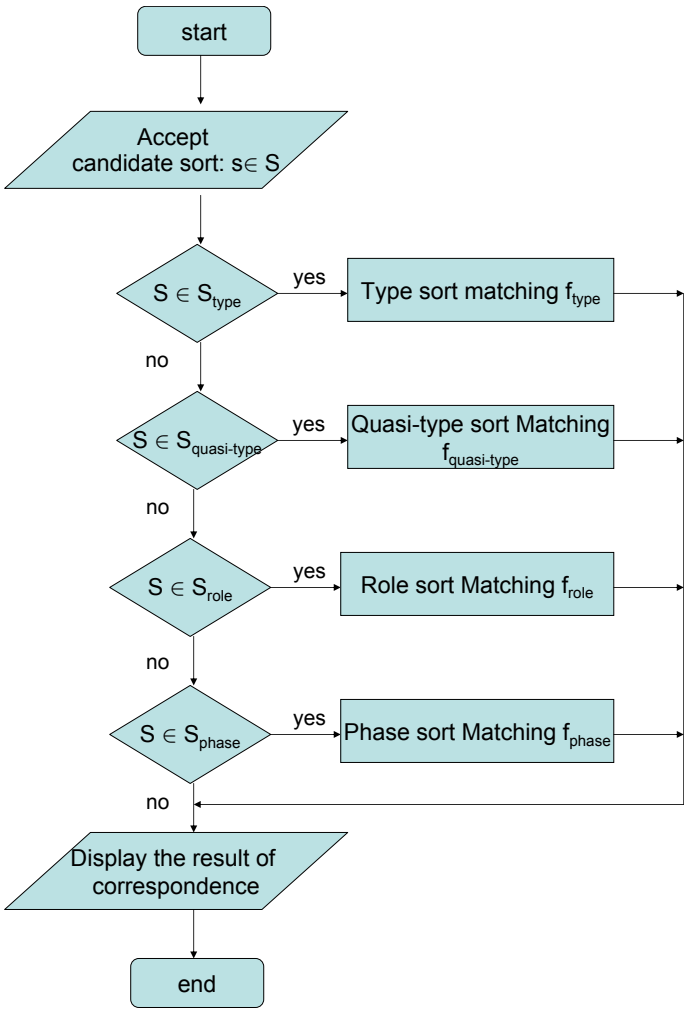

Figure 5.6: The process flow diagram of matching function f

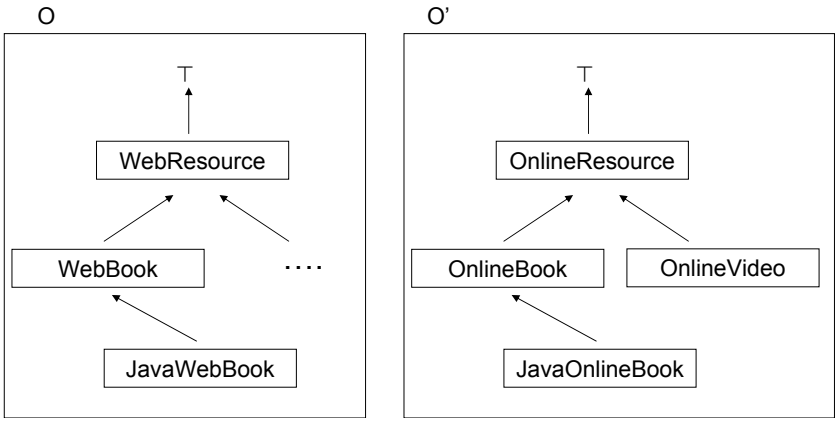

Figure 5.7: Taxonomies of O and O'

$P^D(\texttt{WebBook}) = \{hasURL, hasTitle, hasAuthor, hasCategory = \text{“}Book\text{”}\}$

$P^D(\texttt{JavaWebBook}) = \{hasURL, hasTitle, hasAuthor, hasCategory = \text{“}Book\text{”},$

$hasSubject = \text{“}Java\text{”}\}$

In ontology O',

$P^D(\texttt{OnlineResource}) = \{webAddress\}$

$P^D(\texttt{OnlineBook}) = \{webAddress, resourceType = \text{“}Book\text{”}, name, writer,$

$publishedYear\}$

$P^D(\texttt{OnlineVideo}) = \{webAddress, resourceType = \text{“}Video\text{”}, title, director,$

$VideoStars, publisher, publishedDate\}$

$P^D(\texttt{JavaOnlineBook}) = \{webAddress, resourceType = \text{“}Book\text{”}, name, writer,$

$publishedYear, bookCategory = \text{“}Java\text{”}\}$

The sets of individuals instantiated to `JavaWebBook` *and* `JavaOnlineBook` *via a web search by Google*[3] *are as follows:*

$[\![\texttt{JavaWebBook}]\!] = \{$
c_1=*{Java Application Development on Linux,*
`http://www.phptr.com/content/images/013143697X/downloads/013143697X_book.pdf`*, C. Albing and M. Schwarz, Book, Java}*,
c_2=*{Essentials of the Java Language,*
`http://java.sun.com/developer/onlineTraining/BasicJava/index.html`*, M. Pawlan, Book, Java}*
$\}$
$[\![\texttt{JavaOnlineBook}]\!] = \{$
c_3=*{The Java^{TM} Tutorial,*
`http://java.sun.com/docs/books/tutorial/index.html`*,*
Sun Developer Network, 2000, Book, Java},
c_4=*{Creating Web Applets with Java^{TM},*
`http://docs.rinet.ru/webApp/index.html`*,*
D. Gulbransen and K. Rawlings, 2001, Book, Java},
c_5=*{Essentials of the Java Language,*
`http://java.sun.com/developer/onlineTraining/BasicJava/index.html`*, M. Pawlan, 1999, Book, Java}*
$\}$

[3] `http://www.google.com`

In Example 10, the semantics of `JavaWebBook` and `JavaOnlineBook` can be interpreted as follows:

"`JavaWebBook` is any `WebResource` which has properties `hasURL`, hasTitle, `hasAuthor`, `hasCategory="Book"`, and `hasSubject="Java"`."

"`JavaOnlineBook` is a set of `OnlineResource` individuals that possess specific values for properties `webAddress`, `name`, `writer`, and `publishedYear`, together with two constraints `resourceType="Book"` and `bookCategory="Java"`."

By the first candidate given by Equation 5.1, $[\![\texttt{JavaWebBook}]\!] \neq [\![\texttt{JavaOnlineBook}]\!]$ even though their semantics is similar to each other. Kalfoglou and Scholemmer [220] applies this candidate together with global ontology setting in their matching method called IF-Map. This candidate is possible from the aspect of enclosed domain but it is rare to hit such equality in an open and incomplete domain.

The second possible candidate of sort equality can obtain when two given sorts share a common set of individual-level properties, in a formal notation:

$$\text{if } \; P^D(s) = P^D(s') \text{ then } s \equiv s' \tag{5.2}$$

where $P^D(s) = P^D(s')$ iff for every $p \in P^D(s)$ there exists $p' \in P^D(s')$ such that $p \approx p'$.

So, how can we determine that two properties $p \in P^D(s)$ and $p' \in P^D(s')$ are semantically the same, $p \approx p'$? In general, two properties can have the sameness relation when they both provide the same value for every individual of each sort:

$$p \approx p' \text{ iff } \forall x[p_s(x) \wedge p(x) = p'(x)] \; \wedge \; \forall y[p_{s'}(y) \wedge p(y) = p'(y)].$$

This second candidate is a usage of intensional knowledge defined for each sort. However again, there are some variations in practice. In Example 10, we can see some similar and different properties between `JavaWebBook` and `JavaOnlineBook` given in $P^D(\texttt{JavaWebBook})$ and $P^D(\texttt{JavaOnlineBook})$. Without the same domain and range, it is hard to determine the sameness between two properties. The difficulties by this second candidate can be summarized as follows:

1. The number of properties defined for a sort in each ontology may be different, that is, $|P^D(s)| \neq |P^D(s')|$ where $|P^D(s)|$ and $|P^D(s')|$ are the numbers of properties defined for sort s in ontology O, and sort s' in ontology O', respectively.

2. The domain of properties may not exactly equal, that is, $[\![s]\!] \neq [\![s']\!]$.

3. Consequently, the ranges of properties may be different.

Because a sort can be independently defined in ontology, not only the comparison between two different sets of individuals, but also the property by property comparing between two different sets of properties, are complex and inefficient.

Thus, how is the IC of each sort for this equality? Regarding the characteristic of sort "every individual is identifiable and every sort carries an IC for that", each sort can possess at least one IC. In Example 10, the ICs of `JavaWebBook` and `JavaOnlineBook` can be defined as $\iota_{\texttt{JavaWebBook}} = \texttt{hasURL}$ and $\iota_{\texttt{JavaOnlineBook}} = \texttt{webAddress}$ with a specific domain and range as follows:

$\texttt{hasURL} : \texttt{JavaWebBook} \Rightarrow \texttt{URL}$

$\texttt{webAddress} : \texttt{JavaOnlineBook} \Rightarrow \texttt{URL}$

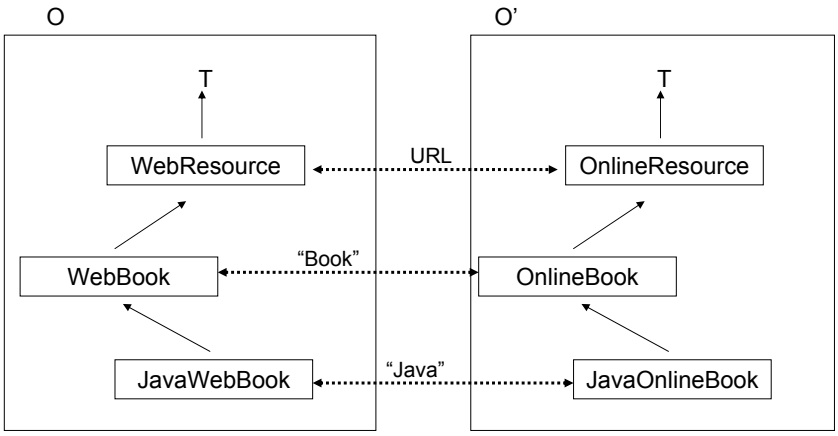

Figure 5.8: sort correspondences between O and O'

According to the definition of IC (Definition 14), the IC of a sort provides a unique IC value for each individual of the sort. Both ICs `hasURL` and `webAddress` provide unique URLs to their individuals. A common IC value is sufficient to determine the identity between two individuals. For example,

$$\texttt{hasURL}(\texttt{c}_2\colon \texttt{JavaWebBook}) = \texttt{webAddress}(\texttt{c}_2\colon \texttt{JavaOnlineBook}) = \texttt{v}_1\colon \texttt{URL}$$

where v_1=`http://java.sun.com/developer/onlineTraining/BasicJava/index.html`. Therefore, c_2 and c_5 represent for the same individual. Two ICs can be the same when they supply the same IC value for each individual of the sorts. `hasURL` and `webAddress` can be determined for their sameness by analyzing the URL and web address of each individual.

When two sorts share a semantically equivalent IC, there are two possible reasons:

- both sorts are subsumed by a similar sort which supplies that IC, or
- the sorts originate that IC and they are the same sort.

It is possible to trace the ICs are whether carried or originated by tracking the path of subsumption relationships. According to the given properties in Example 10, `hasURL` and `webAddress` are possibly the ownICs of `WebResource` and `OnlineResource`. Then, their sub-sorts carry the ICs through subsumption relationships, and thus each individual of a sort possess a unique URL. Figure 5.8 shows the possibility of correspondences between sorts based on their ICs and common attribute values.

In summary, the reason I contend the mapping by ICs is advantageous instead of comparing all properties and individuals is two-fold:

(a) Picking up only ownIC among properties, we do not need to compare all the properties nor the property values.

(b) Also, we do not need to care individual names/labels; furthermore, even the IC names do not need to be same in different ontologies as far as they return same values.

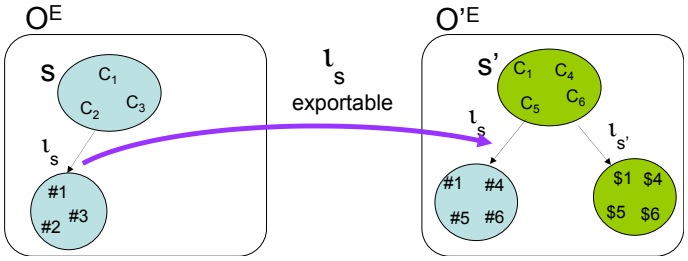

Figure 5.9: The IC of sort s, ι_s, is exportable to sort s'

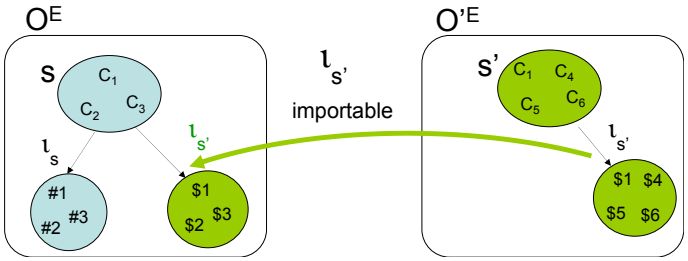

Figure 5.10: The IC of sort s', $\iota_{s'}$, is importable to sort s

5.3.2 IC-based Type Sort Matching

Among four sub-functions of sort matching f, type sort matching is the most fundamental process. *type sort Matching* is a mapping function that finds the correspondence of a type sort $s \in S_{\text{type}}$ in S'_{type}, where S_{type} denotes a set of type sorts, that is

$$f_{type} : s \in S_{type} \Longrightarrow s' \in S'_{type}.$$

Since each type sort originates an IC, the main idea in determining correspondences between type sorts is based on analyzing whether the ownICs of two type sorts are exportable and importable to each other or not. I give the formal definitions of exportable and importable below.

Definition 26 (Exportable) *If the ownIC of sort $s \in S_{type}$, ι_s, can identify and distinguish all the individuals of another sort $s' \in S'_{type}$, then ι_s is* exportable *to s', formally,* $\forall x, y[p_{s'}(x) \wedge p_{s'}(y) \wedge x = y \rightarrow \iota_s(x) = \iota_s(y))]$.

Figure 5.9 illustrates ι_s is exportable to sort s', that is, ι_s can provide a unique IC value for each individual of sort s' similarly like $\iota_{s'}$.

Definition 27 (Importable) *If the ownIC of sort $s' \in S'_{type}$, $\iota_{s'}$, can identify and distinguish all the individuals defined for sort $s \in S_{type}$, then $\iota_{s'}$ is* importable *to s, formally,* $\forall x, y[p_s(x) \wedge p_s(y) \wedge x = y \rightarrow \iota_{s'}(x) = \iota_{s'}(y))]$.

Figure 5.10 illustrates that $\iota_{s'}$ is importable to sort s, that is, $\iota_{s'}$ can provide a unique IC value for each individual of sort s similarly like ι_s. However, for the individuals of each sort, the IC values provided by both exportable IC and importable IC will not be the same, if these two ICs were semantically different.

Here after, I divided the relation between two ICs which are both exportable and importable, into sameness and interchangeability.

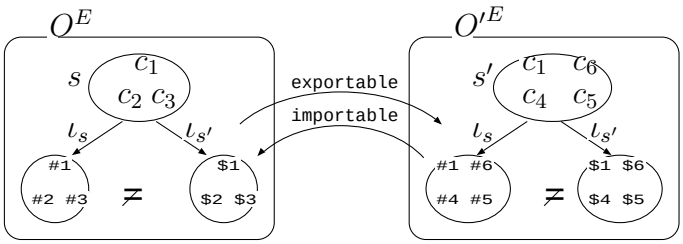

Figure 5.11: Interchangeability between ι_s and $\iota_{s'}$

Definition 28 (Interchangeability of ICs) *If the ownICs ι_s and $\iota_{s'}$, of two sorts s and s', are both exportable and importable, then the ICs have an interchangeability relation:*

$$\iota_s \overset{\bowtie}{=} \iota_{s'} \quad \textit{iff} \quad \begin{array}{l} \forall x, y[p_s(x) \wedge p_s(y) \wedge x = y \rightarrow \iota_{s'}(x) = \iota_{s'}(y)] \wedge \\ \forall x', y'[p_{s'}(x') \wedge p_{s'}(y') \wedge x' = y' \rightarrow \iota_s(x') = \iota_s(y')]. \end{array}$$

Figure 5.11 depicts interchangeability between ι_s and $\iota_{s'}$. By interchangeability, the IC values provided by the ownICs for each individual are unique but they are different.

Definition 29 (Sameness of ICs) *If the ownICs of two given sorts provide the same IC value for every individual of the sorts, then the ICs have a sameness relation:*

$$\iota_s \overset{\bullet}{=} \iota_{s'} \quad \textit{iff} \quad \forall x[p_s(x) \wedge \iota_s(x) = \iota_{s'}(x)] \ \wedge \forall y[p_{s'}(y) \wedge \iota_s(y) = \iota_{s'}(y)].$$

Figure 5.12 illustrates the sameness relation between ι_s and $\iota_{s'}$. In the case of *sameness*, both ownICs must provide the *same IC value* for the same individual, in addition to being exportable and importable.

Sameness is actually a kind of interchangeability. I prove if two ownICs: ι_s and $\iota_{s'}$ are in a sameness relation, then they are interchangeable. According to the definition of IC (Definition 14),

$$\iota_s(c_1{:}\, s) \neq \iota_s(c_2{:}\, s) \quad \textit{iff} \quad c_1 \neq c_2.$$

If ι_s and $\iota_{s'}$ are in a sameness relation, then:

$$\begin{cases} \iota_s(c_1{:}\, s) = \iota_{s'}(c_1{:}\, s), \\ \iota_s(c_2{:}\, s) = \iota_{s'}(c_2{:}\, s), \end{cases}$$

and thus,

$$\iota_{s'}(c_1{:}\, s) \neq \iota_{s'}(c_2{:}\, s) \quad \textit{iff} \quad c_1 \neq c_2.$$

The same is true for $c'_1, c'_2 \in s'$. Therefore, ι_s and $\iota_{s'}$ are interchangeable. This is the end of the proof.

Here after, the formal definitions of sort equality by sameness and interchangeability relations are provided.

Definition 30 (Sort Equality by Sameness of ICs) *For sorts s and s' with ownICs ι_s and $\iota_{s'}$ respectively, sorts are equal with the sameness relation of their ownICs: $s \overset{\bullet}{\equiv} s'$ iff $\iota_s \overset{\bullet}{=} \iota_{s'}$.*

In Example 10, there is a sort equality be sameness of ICs between `WebResource` and `OnlineResource` such that `WebResource` $\overset{\bullet}{\equiv}$ `OnlineResource` with `hasURL` $\overset{\bullet}{=}$ `webAddress`.

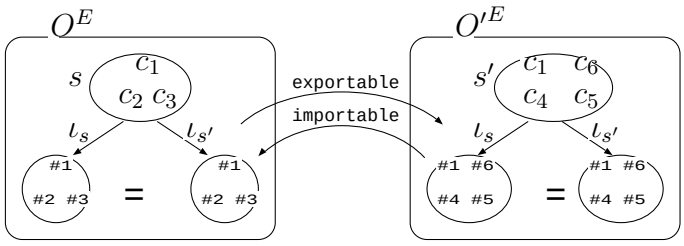

Figure 5.12: Sameness between ι_s and $\iota_{s'}$

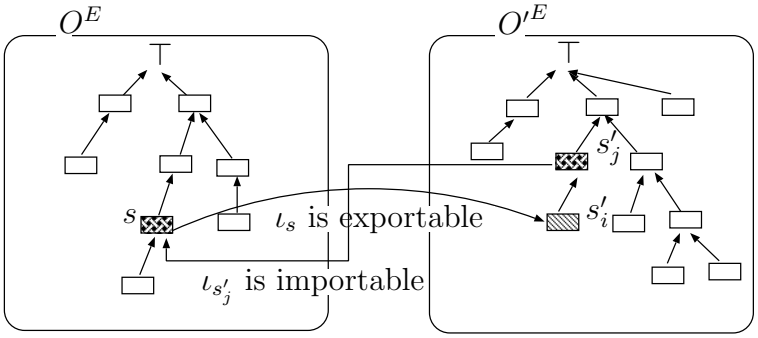

Figure 5.13: IC-based type sort matching

Definition 31 (Sort Equality by Interchangeability of ICs) *For any two sorts s and s' with ownICs ι_s and $\iota_{s'}$, there is sort equality by the interchangeability relation between their ownICs: $s \overset{\bowtie}{\equiv} s'$ iff $\iota_s \overset{\bowtie}{=} \iota_{s'}$.*

Example 11 *Suppose that* `hasFingerPrint` *and* `hasIrisPattern` *are defined as the ownICs of* `Person` *and* `HumanBeing`*, and both ICs can identify their related individual persons interchangeably but not supplying the same IC value. Also suppose that* `hasOrgName` *and* `TitleOfOrganization` *are the ownICs of* `LegalOrganization` *and* `Organization` *and they both supply the same organization name or ID to each individual. Then, sort equalities can be determined as follows:*
(a) `LegalOrganization` $\overset{\bullet}{\equiv}$ `Organization` *with* `hasOrgName` $\overset{\bullet}{=}$ `TitleOfOrganization`
(b) `Person` $\overset{\bowtie}{\equiv}$ `HumanBeing` *with* `hasFingerPrint` $\overset{\bowtie}{=}$ `IrisPatternOf`

Note that $\overset{\bullet}{\equiv}$ and $\overset{\bowtie}{\equiv}$ are the variations of sort equality $\equiv$ by the sameness and interchangeability of ownICs. The source of these variations come from multiple ownICs. It is obvious that a sort can originate more than one ownIC, for example, sort `Person` originates three ownICs such as `hasFingerPrint`, `hasIrisPattern`, and `hasPalmVeinPattern`. Thus, the equality between two similar sorts cannot be determined by the sameness between their ownICs if each ownIC is semantically different to each other, that is, the IC values provided by such ownICs for an individual are not the same.

Now, I will present the procedure of IC-based type sort mapping. Figure 5.13 depicts the approach of IC-based type sort matching. In the process, a bottom-up searching approach is applied because ICs are inherited from top to bottom. Our intuition in this is if an IC is exportable to a sort at the bottom-level, then it is a proof that a type sort exists at the upper-level of the same branch. In order to find the correspondence of a

candidate sort $s \in S_{type}$ in O', the matching process needs to analyze the related ownICs are exportable as well as importable. However, it is unreasonable to check whether $\iota_{s'}$ is importable to sort s if ι_s failed to be exportable to sort s'_i. Therefore, the search process will detect a next possible sort $s'_i \in S'_{type}$ whenever ι_s is not exportable. When ι_s is successfully exportable to s'_i, the export process continues to s'_j such that $s'_j \sqsupseteq s'_i$ till the process assures a top-most exportable sort for ι_s. By this approach, the matching process can reduce complexity. I summarize the procedure of IC-based type sort mapping, below.

1. Choose a sort ,s'_i, from S'_{type} by any chance.
2. Test whether the ownIC of sort s, ι_s, is exportable to s'_i.
3. If yes, find s'_j such that $s'_j \sqsupseteq s'_i$, and test the top-most exportable sort s'_j for ι_s, then test whether the ownIC of sort s', $\iota_{s'}$, is importable to s or not.
   (a) If yes, there is interchangeability, then test for sameness.
      i. If yes, there is sort equality by sameness between s and s'_j such that $s \overset{\bullet}{\equiv} s'_j$.
      ii. Otherwise, there is sort equality by interchangeability between s and s'_j such that $s \overset{\bowtie}{\equiv} s'_j$.
   (b) Otherwise, try such $s'_j \sqsubseteq s'_i$ on other branches for interchangeability.
4. Otherwise, go to (1) to select a next possible sort s'_i.

5.3.3 Matching of Quasi-type sorts, Role sorts, and Phase sorts

Among all matching functions, type sort matching is the most fundamental in my method. According to the classification structure among sorts, quasi-type sorts, role sorts, and phase sorts are subsumed by type sorts. In an alternative speaking, type sorts supply ICs for the identity of individuals and all sorts carry ICs from the type sorts which subsume them. All matching functions obey the following heuristic.

> *No correspondence (or match) can be found between two sorts if there is no sort equality between their type sorts.*

Therefore, each matching function invokes type sort matching in order to fix the scope of possible matches. By the classification of sorts, it comes easy to detect the most pertinent type sort which supplies an IC to the candidate sort. Two basic ideas in each sub-function are (1) fixing the scope of possible matches by detecting correspondence between super-sorts, especally between type sorts, and (2) analyzing the sorts exist in the scope for a correspondence, via a special property such as CVA, EDR, and CC.

Quasi-type sort Matching

Quasi-type sort Matching is a mapping function that finds the correspondence of quasi-type sort $s \in S_{\text{quasi-type}}$ in $S'_{\text{quasi-type}}$, that is

$$f_{quasi-type} : s \in S_{quasi-type} \Longrightarrow s' \in S'_{quasi-type}.$$

There are two main steps in the process of quasi-type sort matching function.

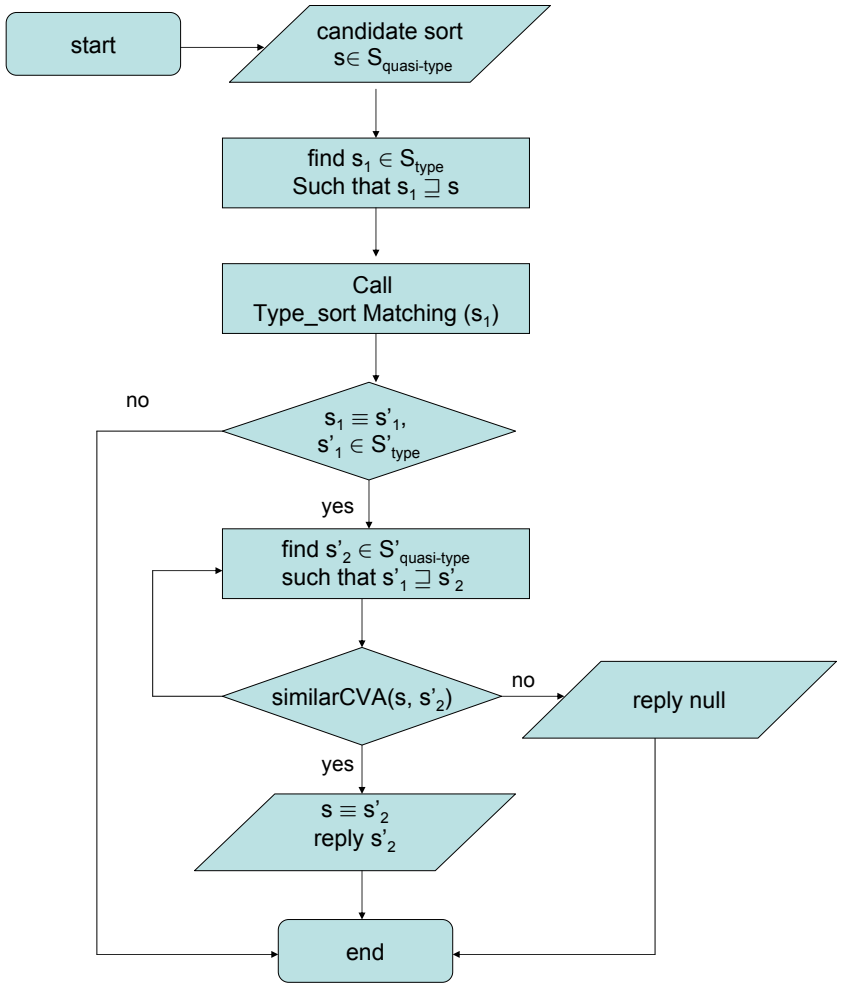

Figure 5.14: The process flow diagram of quasi-type sort matching

- First, the scope of possible matches in $S'_{\text{quasi-type}}$ is decided by finding type sort $s'_1 \in S'_{type}$ which has a corresponding type sort $s_1 \in S_{type}$ such that $s \sqsubseteq s_1$.
- After that, the correspondence of s is determined by a similar CVA in both $P^D(s)$ and $P^D(s')$.

Figure 5.14 describes the detailed process flow of quasi-type sort matching. A quasi-type sort, $s \in P_{quasi-type}$, will be the input of this process. The function finds type sort s_1 which subsumes s such that $s_1 \sqsupseteq s$. Then, the function is proceeded to search the corresponding type sort of s_1 in O'^E, that is $s'_1 \in S'_{type}$. For this search, quasi-type sort matching calls type sort matching function. If s'_1 is successfully found, then the scope of possible matches in O' for sort s is fixed by selecting quasi-type sorts, $s'_2 \in S'_{quasi-type}$, which all are subsumed by s'_1; otherwise the function will end. There is a loop for checking whether s and any of s'_2 have a similar CVA or not. I use a simple string matching approach to determine for a similar CVA. If a corresponding sort s'_2 is found then the quasi-type sort matching function will return s'_2; otherwise null.

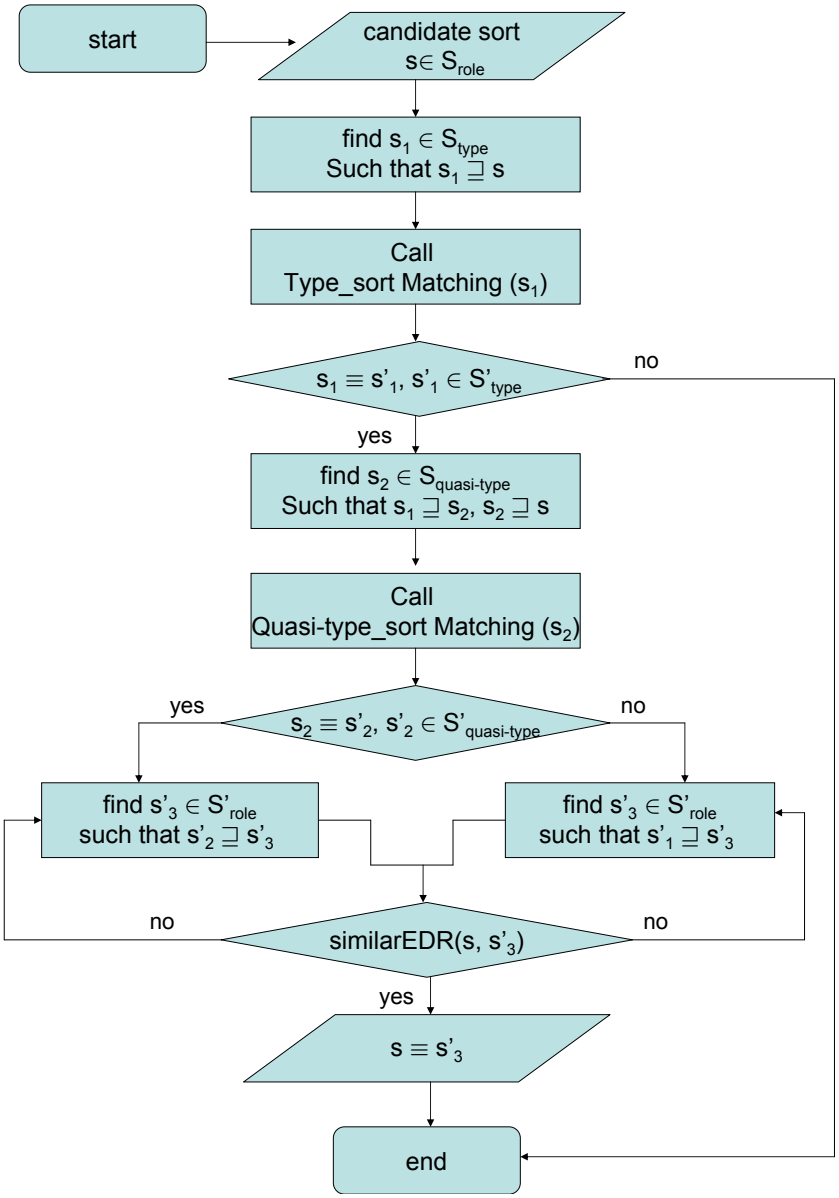

Figure 5.15: The process flow diagram of role sort matching

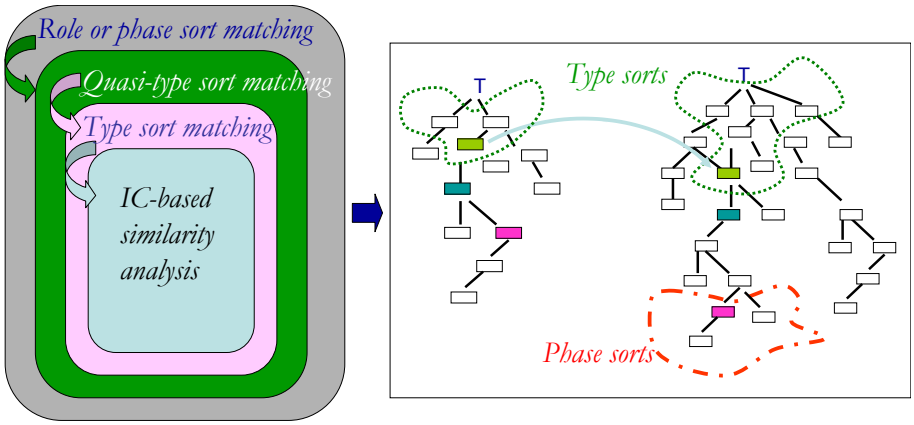

Figure 5.16: A view of phase sort matching through divide-and-conquer approach

Role sort Matching

Role sort Matching is a mapping function that finds the correspondence of role sort $s \in S_{\text{role}}$ in S'_{role}, that is

$$f_{role} : s \in S_{role} \Longrightarrow s' \in S'_{role}.$$

Figure 5.15 shows the process flow of roleMatching function. There are three steps in the role sort matching process.

- First, select sort $s_1 \in S$ such that $s \sqsubseteq s_1$, to find a corresponding type sort $s'_1 \in S'_{type}$ by invoking type sort matching function.
- Second, if type sort matching successfully returns a corresponding type sort s'_1, then determine the scope of possible matches by searching a corresponding quasi-type sort $s'_2 \in S'_{quasi-type}$.
- Third, according to the result of corresponding quasi-type sort s'_2, examine the scope of possible matches among role sorts again. If there is $s_2 \equiv s'_2$ then the scope of possible matches will be $s'_2 \sqsupseteq s'_3$, otherwise $s'_1 \sqsupseteq s'_3$. And the correspondence between s and any of s'_3 will be determined by analyzing for a common EDR in both $P^D(s)$ and $P^D(s')$.

Since EDR is a relation between a sort and its dependent sort, the role matching function needs to analyze whether two given EDR satisfies sameness relation such that $p_d \approx p'_d$. Figure 5.15 shows the process flow of role sort matching.

Phase sort Matching

Phase sort Matching is a mapping function that finds the correspondence of phase sort $s \in S_{\text{phase}}$ in S'_{phase}, that is denoted as

$$f_{phase} : s \in S_{phase} \Longrightarrow s' \in S'_{phase}.$$

The process of phase sort matching is almost similar to the steps of role sort matching, except analyzing CC instead of EDR to determine the correspondence of s. Therefore, please refer to Figure 5.15 for the process flow diagram of f_{phase} by substituting `similarCC`(s, s'_3) in the place of `similarEDR`(s, s'_3). A general view of phase sort matching function is shown in Figure 5.16.

5.4 Matching Algorithm

Following to the matching processes presented in the previous section, I describe the algorithm of mapping function, f, in the following.

function $f(s, O^E, O'^E)$
begin
  if $s \in S_{\text{type}}$ **then** $s' := f_{type}(s, O^E, O'^E)$;
  if $s \in S_{\text{quasi-type}}$ **then** $s' := f_{quasi-type}(s, O^E, O'^E)$;
  if $s \in S_{\text{role}}$ **then** $s' := f_{role}(s, O^E, O'^E)$;
  if $s \in S_{\text{phase}}$ **then** $s' = f_{phase}(s, O^E, O'^E)$;
  return s';
end;

function $f_{type}(s, O^E, O'^E)$
begin
  For $s' \in S'_{\text{type}}$ **do**/% bottom-up approach%/
    if interchangeability(s, s')=yes OR sameness(s, s')=yes **then**
      return s';exit;
  end for;
  return null;
end;

function $f_{quasi-type}(s, O^E, O'^E)$
begin
  $s' :=$ null;
  if $\mid S'_{\text{quasi-type}} \mid \neq 0$ **then** $s'_1 := f_{type}(s_1, O^E, O'^E)$, $s \sqsubseteq s_1$;
    if $s'_1 \neq$ null **then**
      For $s'_2 \in S'_{\text{quasi-type}}$ such that $s'_2 \sqsubseteq s'_1$ **do**
        if similarCVA(s, s'_2)=yes **then** $s' := s'_2$;exit;
      end for;
  return s';
end;

function $f_{role}(s, O^E, O'^E)$
begin
  $s' :=$ null;
  if $\mid S'_{\text{role}} \mid \neq 0$ **then** $s'_1 := f_{quasi-type}(s_1, O^E, O'^E)$, $s \sqsubseteq s_1$;
    if $s'_1 \neq$ null **then**
      For $s'_2 \in S'_{\text{role}}$ such that $s'_2 \sqsubseteq s'_1$ **do**
        if similarEDR(s, s'_2)=yes **then** $s' := s'_2$;exit;
      end for;
  return s';
end;

function $f_{phase}(s, O^E, O'^E)$
begin
  s' := null;
  if $\mid S'_{\text{phase}} \mid \neq 0$ **then** $s'_1 := f_{quasi-type}(s_1, O^E, O'^E)$, $s \sqsubseteq s_1$;
    if $s'_1 \neq$ null **then**
      For $s'_2 \in S'_{\text{phase}}$ such that $s'_2 \sqsubseteq s'_1$ **do**
        if similarCC(s, s'_2)=yes **then** $s' := s'_2$;exit;
      end for;
  return s';
end;

function sameness(s, s') /% interchangeability plus equal IC values %/
begin
  if interchangeability(s, s') = yes
  then
    $\iota_s := ownIC(s)$;$\iota_{s'} := ownIC(s')$;$flag$:= yes;
    for $c \in s$ **do**
      if $\iota_s(c{:}\, s) \neq \iota_{s'}(c{:}\, s)$ **then** $flag$ = no; **end for**;
    if $flag$ = yes **then**
      for $c' \in s'$ **do**
        if $\iota_s(c'{:}\, s') \neq \iota_{s'}(c'{:}\, s')$ **then** $flag$ = no; **end for**;
    return $flag$;
end;

function interchangeability(s, s') /% both exportable and importable %/
begin
  ι_s := ownIC(s);
  mutual_flag := no;
  if exportable(ι_s, s')= yes
  then
    s'' := search_upward(s', ι_s);
    $\iota_{s''} := ownIC(s'')$;
    if importable($\iota_{s''}$, s)= yes **then**/% $s \stackrel{\bowtie}{=} s''$ %/
      $mutual_flag := yes$; $s' := s''$;
  end if;
  return mutual_flag;
end;

function exportable(ι_s, s') **begin**
  export_flag := yes;
  for $c_1 \in s$ **do**
    for $c_2 (\neq c_1) \in s$ **do**
      if $\iota_{s'}(c_1{:}\, s) = \iota_{s'}(c_2{:}\, s)$ **then** export_flag := no;
  return export_flag;
end;

```
function importable(ι, s)
begin
    return exportable(ι, s);
end;

function upper_sort(s')
begin
    return s'' such that s'' ⊒ s' ∈ S';
end;

function search_upward(s'_1, ι_s)
begin
    exportable_top-most := s'_1;
    s'_2 := upper_sort(exportable_top-most);
    while s'_2 ≠ nil and exportable(ι_s, s'_2) = yes do
        exportable_top-most := search_upward(s'_2, ι_s);
    return exportable_top-most;
end;
```

The corresponding sorts between two heterogeneous ontologies can be found through the above four matching functions.

5.5 Evaluation

In this section, I evaluate the EnOntoModel-based matching method by calculating the mathematical complexity of matching function f.

Suppose that the maximum number of sorts in O^E and O'^E are N. Let m and n be the number of individuals for sorts s and s'. The retrieval of IC value for an individual takes a constant time. Also, the computation for the equality between two IC values takes a constant time. Therefore, the complexity to check whether two ICs supply the same IC value for an individual or not, will cost $\mathcal{O}(1)$. Consequently, the tests for exportable IC and importable IC would take $\mathcal{O}(m)$ and $\mathcal{O}(n)$ respectively. For the convenience of estimation, if we regard a binary tree for taxonomies, then the average depth would be $\log N$, and the approximate number of leaves would be $N/2$, as shown in Figure 5.17.

The mathematical complexity of matching function f, denoted by T_f, is calculated based on the complexity of four sub-functions: f_{type}, $f_{quasi-type}$, f_{role}, and f_{phase}. Since type sorts are necessry for EnOntoModel-based ontologies concerning the identity of individuals, the number of type sorts in both ontologies is not zero. Let the maximum number of type sorts be k, $1 \leq k \leq N$. Then, the maximum number of quasi-type sorts, role sorts, and phase sorts, in each ontology will be $N - k$.

Let T_{type} be the worse case complexity of type sort matching. T_{type} can be calculated based on the cost of exportable IC in maximum plus the cost of importable IC. The maximum steps to achieve a top-most exportable sort in O'^E would be $k/2 + \log\ k$. Then, the worse case complexity of type sort matching will be as follows:

$$T_{\text{type}} = (k/2 + \log\ k) \times m + n$$

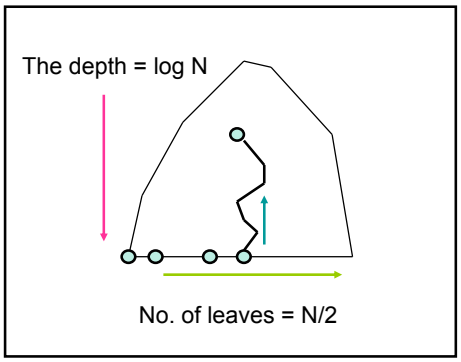

Figure 5.17: The average depth and number of leaves in a binary tree

where $(k/2 + \log k/2) \times m$ for the maximum number of comparisons in export test, and n for the maximum number of comparisons in import test.

Let $T_{\text{quasi-type}}$, T_{role}, and T_{phase}, are the worse case complexities of $f_{quasi-type}$, f_{role}, and f_{phase}, respectively. Then, the complexities of these sub-functions are as follows:

$$T_{\text{quasi-type}} = T_{\text{type}} \ + \ (N-k) \times \mathcal{O}(1)$$

$$T_{\text{role}} = T_{\text{quasi-type}} \ + \ (N-k) \times (m+n)$$

$$T_{\text{phase}} = T_{\text{quasi-type}} \ + \ (N-k) \times \mathcal{O}(1)$$

Note that the complexity of CVA or CC, is assumed to be $\mathcal{O}(1)$ because of direct attribute value or constraint checking. In the case of role sort matching, matching between two EDRs will cost $\mathcal{O}(m+n)$, due to checking whether two EDRs are semantically similar. Finally, T_f in the worse case will be as follows:

$$\begin{aligned} T_f &= T_{\text{type}} + T_{\text{quasi-type}} + T_{\text{role}} + T_{\text{phase}} \\ &= (\ ((k/2 + \log\ k) \times m + n) + ((N-k) \times m) + ((N-k) \times n)\ ) \\ &= \mathcal{O}(N \times m) \end{aligned}$$

Although the mathematical complexity of f_{type} is $\mathcal{O}(N \times m)$, in practical cases, we can expect that $(k/2 + \log k) < N$.

If T_f is applied for the complete ontology matching, T_c, of all available sorts between O^E and O'^E, then T_c would be $N \times T_{\text{type}} = \mathcal{O}(N^2 \times m)$. However, it can be reduced to $\mathcal{O}(N \log N \times m)$, because the matching functions need not be executed for the sub-sorts of every un-matched type sort.

5.6 Experimental Results

In this section, I present the experimental results of EOM and my experience of ontology mapping. The major objective is to evaluate the matching accuracy of EOM using real data sets. The implementation of EOM is coded in Java by utilizing Jena OWL API and Protégé OWL API.

In the experiments, I conducted evaluations in terms of two measurements: *precision* and *recall*. In general, precision and recall measures are designed to compare the set of correct correspondences in the program generated correspondences with expert-defined correspondences. Let me denote three sets: (a) C_e for expert-defined correct correspondences, (b) C_f for program-generated correspondences by using EOM function f, and (c) C_{fe} for the correctness of C_f in C_e such that $C_{fe} = C_f \cap C_e$. Then, the correctness of Precision and recall are defined as follows:

- Precision (P) is the percentage of correctness in the program generated correspondences. P is calculated by $\frac{|C_{fe}|}{|C_f|}$.
- Recall (R) is the percentage of the correctness of program generated correspondences, comparing with expert-defined correct correspondences. R is calculated by $\frac{|C_{fe}|}{|C_e|}$.

In order to get real data sets for this experiment, I analyzed a number of OWL ontology libraries on the Web, in particular Protégé OWL library[4]. Even though a large number of ontologies are avaliable, most of them are domain-dependent such as biomedical, chemical, geographic, micro-electronic, space & earth, PSM-Problem Solving Method, country, travel, wine, etc. Thus, it is not easy to choose some of them as mapping candidates without adequate background knowledge of each domain. Here, I selected two domains: academic research and wine. Then, I chose two ontologies for each domain as follow and prepared two data sets.

1. *Ontologies in Academic Research.* Ontologies describe concepts in academic research such as research areas, activities, publications, institutions, and kinds of community people. Namely, they are `ka.owl`[5] and `Research-Community.owl`. In `ka.owl`[6], the developer focuses on the concepts of academic research particularly in knowledge acquisition. `Research-Community.owl` is general rather than `ka.owl`. However, both ontologies are partially overlapped with some heterogeneities.

2. *Ontologies in Wine.* Wine ontologies describe concepts in wine domain that consist of winery, wine region, wine grape, wine categories, and recommended food of fine wines, etc. `wine.owl`[7] covers most of famous wineries and vineyards around the world. `French-Wine.owl` is developed by the author and it is specialized only for wines in France regions. The significant difference between two wine ontologies is the approach of conceptualization, that is, `wine.owl` is mainly focused on wine regions while wine categories in `French-Wine.owl` are based on wine grapes particularly grown in France regions. Thus, heterogeneity is highly involved between two similar concepts.

[4] `http://protege.cim3.net/cgi-bin/wiki.pl?ProtegeOntologiesLibrary`

[5] `ka.owl` is developed by Ian Horrocks who is a professor in the School of Computer Science at the University of Manchester, also a member of Information Management Group for Formal Methods and Bio and Health Informatics (see `http://www.cs.man.ac.uk/%7Ehorrocks/`).

[6] http://protege.cim3.net/file/pub/ontologies/ka/ka.owl

[7] `wine.owl` is developed from Stanford University (see `http://protege.cim3.net/file/pub/ontologies/wine/wine.owl`). The original version is in DAML and it is substantially changed to OWL version.

Ontology	$\mid S \mid$	$\mid S_{Type} \mid$	$\mid S_{\text{Quasi-type}} \mid$	$\mid S_{Role} \mid$	$\mid S_{Phase} \mid$	P^D	Individual
`ka.owl`	82	10	60	8	4	137	250
`Research-Community.owl`	35	6	14	9	6	59	80
`wine.owl`	105	26	79	0	0	67	186
`French-Wine.owl`	73	15	58	0	0	43	142

Table 5.1: Statistics of data sets

Ontology Mapping	$\mid C_e \mid$	$\mid C_f \mid$	$\mid C_{fe} \mid$	P	R
`ka.owl` and `Research-Community.owl`	26=6+11+5+4	20=5+7+4+4	18	90.00	69.23
`wine.owl` and `French-Wine.owl`	29=11+18+0+0	18=6+12+0+0	17	94.44	58.62

Table 5.2: Precision (P) and recall (R) on data sets

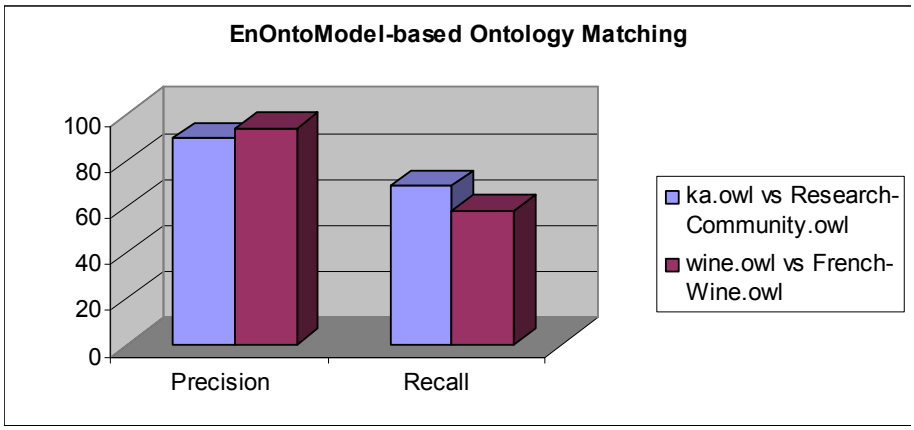

Figure 5.18: Experimental results in two domains

Note that the enriched versions of `ka.owl` and `wine.owl` are generated in the form of EnOntoModel. Table 5.1 shows the statistics of data sets given for each enriched ontology. I assigned 90% common individuals for both wine ontologies. However, the percentage of overlapped individuals between two academic research ontologies is about 40%. Table 5.2 shows the precision and recall of two mapping experiments. Then, Figure 5.18 illustrates the experimental results in a bar chart style.

What I learnt from this experiment is the precision of EOM method is sufficient, that is, the concept corresspondences found by the program is mostly correct. I contend to say this correctness is effected from enrichment because concepts and individuals are consistently defined in terms of specific properties such as ICs, EDR, CVA, and CC. Currently, the result of recall is not so strong. I observed that there is a little tricky in comparing two IC values—mostly string values in different orders and phrases—which are given in terms of compound ICs—a tuple of individual-level properties for IC. It is better to use an appropriate text analyzer instead of conventional string matching. Also, an issue with these measures is that the correctness of expert-defined correspondences is a subjective measure, which are slightly varied by expert to expert according to their knowledge. A better way is to let appropriate domain experts evaluate the method through ontologies used in real information systems, and then compare the results with other methods.

5.7 Related Work

As for related work, I will discuss in two divisions. The first division is a comparative discussion with existing mapping methods. The next division is difference between OntoClean and my research.

5.7.1 Ontology Matching

I have presented some matching tools and methods such as MAFRA, ONION, PROMPT, IF-Map, COMA++, QOM, and GLUE, in Section 3.3. Here, I describe a summary of their matching methods as related work to EnOntoModel-based matching.

In *MAFRA*, similarity between two concepts is calculated mainly using lexical analysis via WordNet, domain glossaries, bi-lingual dictionaries, and corpuses. There is no explicit deterministic heuristics other than lexical heuristics (or synonyms), in the semantic bridge construction.

ONION is an heuristic-based ontology mapping system to resolve terminological heterogeneity using two matching approaches: linguistic matching via WordNet[8] and instance-based matching via databases. ONION provided algorithms for how to calculate a similarity score between a little complex names of ontological concepts such as 'Department of Defense' and 'Defense Ministry'.

PROMPT is a semi-automatic and interactive tool suit for performing ontology merging, based on the Frame paradigm. For concept matching, AnchorPROMPT firstly detects linguistic similarity matches (called anchors) between domain concepts. It is a usual way of fixing the scope of possible correspondences. Secondly, AnchorPROMPT analyzes the paths of the input ontologies delimited by the anchors in order to determine concepts frequently appearing in similar positions on similar paths. Thirdly, PROMPT

[8] `http://wordnet.princeton.edu/`

Matching Methods	IBSA	LBSA	SBSA	TBSA	Meta-knowledge
MAFRA	*	*	*	*	
ONION	*	*			
PROMPT		*	*	*	
IF-Map	*				
COMA++		*	*	*	
QOM		*	*	*	
GLUE	*	*	*	*	
EOM	*		*	*	*

Table 5.3: Techniques of similarity analysis applied in each matching method

proposes some correspondences determined by analyzing the structural knowledge—both schematic and taxonomical knowledge—of terminologically similar concepts. The limitation of PROMPT is that the two ontologies in the merging process should be different versions of the same ontology. This limitation intends to reduce the complexity of mapping and merging between heterogeneous ontologies.

IF-Map is a channel-theory-based ontology mapping technique. In IF-Map, there are two assumptions: (1) using a common reference ontology for all local ontologies, and (2) considering an equal set of instances for the decision of concept mapping. Kalfoglou and Schorlemmer claim that IF-Map could provide fully automation for a matching process. However, the second assumption is a big restriction for the applicability of IF-Map concerning instantiation heterogeneity.

COMA++ supports higher-level strategies to address complex match problems, in particular fragment-based matching and the reuse of previous match results. Following the divide-and-conquer idea, it decomposes a large match problem into smaller subproblems by matching at the level of schema fragments. COMA++ encompasses two matching phases: (a) identifying similar fragments, and (b) matching fragments.

QOM focuses on less run-time complexity for mapping efficiency of large-size, light-weight ontologies. However, QOM mostly constitutes a straightforward name-based similarity computation via RDFS syntax in order to determine correspondences between two ontologies.

GLUE is a system that employs a multi-strategy machine learning technique with joint probability distribution. Firstly, GLUE identifies the similarities of instances. Secondly, it compares between relations, based on the similarity results of instances. GLUE uses two kinds of base learners: a name learner to encounter possible correspondences through a linguistic approach, and a number of content learners to predict the similarity between instances and between properties according to the types of properties. Finally, meta-learner combines the predictions of all base learners and determines concept correspondences.

Among the above mapping tools, the matching methods of PROMPT, COMA++, and GLUE are similar even though PROMPT and COMA++ do not apply instance-based analysis. The same point among them is that the scope of possible matches is predicted using name-based matching. EnOntoModel-based matching method uses concept-level properties (also called meta-knowledge) together with intensional and extensional knowledge of domain concepts. The usage of concept-level properties is a distinctive feature of my method from the others. I redescribe Table 3.1 by adding EOM and show it in

Table 5.3. I appreciate that each mapping method has its own advantages with different focuses, assumptions, and limitations.

In general, I contend two advantages for EOM over other mapping methods. The first advantage is a more reliable approach of predicting possible correspondences. As I have discussed, two concepts with the same name may have different semantics. Suppose that ontology developed by a certain university, where only graduate courses are available, uses concept name `Student` for a set of graduate students. Another ontology developed by a different university, where only undergraduate courses are available, may use the same name for different semantics. In the opposite case, two concepts might have different names. Because the meaning of concept names cannot completely express the semantics of concepts, a chance of correspondence between two terminologically quite different concepts is very less or not obtainable. In my method, domain concepts are systematically classified into sort categories according to EnOntoModel. Therefore, the scope of possible correspondences is already fixed. Moreover, my method can trim impossible correspondences mainly by analyzing the equalities of type sorts. The second advantage is less complexity that is achieved by the following points:

- The complexity is initially reduced by a direct matching between the same sort groups.
- The complexity is reduced by trimming impossible correspondences via type sorts.
- my method can determine sort correspondences by analyzing only a specific property, ownIC, CVR, EDR, or CC, while other methods need to analyze all defined properties.

In order to prove for less complexity, I examine the similarity analysis by the content learners of GLUE in terms of mathematical complexity. In GLUE, the similarity between two nodes (classes) is determined by the similarity of their attributes and relations with their neighbour nodes. Then, the similarity between two attributes is calculated by the similarity between their corresponding instances. Suppose that N_c, N_p, and N_i are the maximum number of nodes, properties (attributes & relations), and instances. Let us assume that the complexity of comparing two attribute values between two instances is $\mathcal{O}(1)$. Then, the complexity of calculating similarity between two instances will be $\mathcal{O}(N_p)$. And, $\mathcal{O}({N_p}^2 \times N_i)$ will be the complexity for the similarity between two nodes. Finally, the matching between two ontologies will take $\mathcal{O}(\log N_c \times {N_p}^2 \times N_i)$. In order to compare GLUE with my matching method, let us substitute N for every parameter; the cost of GLUE will become $\mathcal{O}(N^3 \log N)$, while my matching method costs $\mathcal{O}(N^2 \log N)$ because the method does not require comparing all properties belonging to each class.

Figure 5.19 illustrates the complexity difference between EnOntoModel-based matching and GLUE's content-based matching in a line chart style. The chart states that the complexity difference is especially by number of properties, assuming that number of sorts and instances are equal in each case. Whenever the number of properties belonging to sorts increases, then the complexity difference will increase proportionally. In this comparison, I consider IC as a compound property, that is, the number of individual-level properties compiled for an IC is ≥ 1.

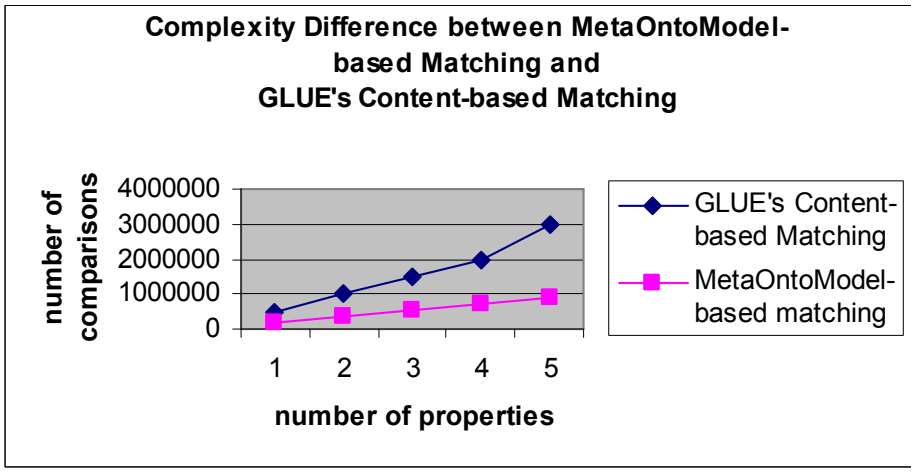

Figure 5.19: EnOntoModel-based matching vs GLUE's Content-based matching

5.7.2 Ontological Analysis

Welty and Guarino provided a methodology called OntoClean to perform ontological analysis—cleaning the taxonomic structure of ontologies by validating subsumption relationships. The methodology is based on some ontological notions drawn from philosophy. The foundation of OntoClean was started in 1994 [139, 140, 141, 142, 143, 144]. Then, the OntoClean methodology was first introduced in a series of papers in 2000 [145, 146, 147, 148, 149]. All references can be found at the home page of OntoClean[9].

OntoClean methodology consists of three main contributions: (a) defining four fundamental meta-properties: identity (**I**), rigidity (**R**), unity (**U**)[10], and dependency(**D**) for ontological concepts, (b) classifying ontological concepts into sortal and non-sortal concepts given by a quadruple of certain values[11] based on the meta-properties, and (c) providing some subsumption constraints to clean taxonomic structures of ontologies.

I observed that the philosophical notions behind OntoClean is useful not only for ontological analysis but also for conceptual modeling of ontologies. Therefore, I apply three[12] of OntoClean's meta-properties for ontology mapping in order to deal with semantic heterogeneity. Concerning the philosophical notions, I describe my add-ons over OntoClean as follows:

- The formalizations of OntoClean's meta-properties are confusing without a precise semantics by a formal logic language. Thus, I provided a First-order Quantified Modal Language $\mathcal{L}^E$ and redefine each notion explicitly.

[9] `http://www.ontoclean.org/`

[10] Unity is the notion of whole for each individual of a concept. Something is a *whole* if all of its parts are connected to each other and to nothing else [146]. The unity considers an internal relation between an instance and its parts, such as the parts of a human body are tightly connected to each other as a whole body of a human.

[11] A combination of meta-property values is presented as a classifier of concepts. For example, $(+\mathbf{R}+\mathbf{O}+\mathbf{I}-\mathbf{D})$ for type, $(+\mathbf{R}-\mathbf{O}+\mathbf{I}-\mathbf{D})$ for quasi-type, and $(\sim\mathbf{R}-\mathbf{O}+\mathbf{I}+\mathbf{D})$ for material role where **O** for ownIC, and $+\mathbf{R}$ and $\sim\mathbf{R}$ are rigidity values: rigid and anti-rigid respectively. For each meta-property, there are three different property values by attaching different symbols: $+$, $-$, and $\sim$.

[12] I indirectly apply unity in the meaning of individuals because every individual can be assumed as a whole which is countable and identifiable.

- In OntoClean, IC is formalized as a characteristic relation which is not precise to apply in computer systems. I provided an explicit formalization of IC using a unary function of language $\mathcal{L}^E$. In additon, I showed that ICs can be written in the form of `owl:DatatypeProperty` with restrictions of `owl:FunctionalProperty`, `owl:InverseFunctionalProperty`, and = 1 cardinality. Relatively, domain concepts/classes can be represented as sorts and sorts can be written in the form of `owl:Class` which is restricted by an IC.

- I could provide a model of semantically-enriched ontologies (named EnOntoModel), in which the philosophical notions are explicitly embedded into domain ontologies as concept-level properties.

- Also, I demonstrated how to enrich domain ontologies in the form of EnOntoModel through Protégé OWL API, and applied an idea similar to OntoClean's analysis in the meta-class ontology, to check conceptual consistency of enriched ontologies.

In summary, the philosophical notions are applied for ontological analysis in the work of OntoClean. Then, they are employed in my research for an efficient matching between heterogeneous ontologies through a semantic enrichment process.

5.8 Benefit and Cost of EnOntoModel

The fundamental objective of semantic enrichment is to provide adequate and precise semantics for domain concepts by fertilizing additional knowledge in the descriptions of concepts. In this research, the motivation of semantic enrichment is derived from matching between heterogeneous ontologies. In order to deal with wide-scale semantic heterogeneity, I proposed philosophy-based concept classification theory and EnOntoModel. Though some advantages particularly time cost and accuracy, are significant in EOM method, I admit the cost of enrichment for this. Thus, what is the development cost of EnOntoModel-based ontology and how to calculate it, is raised as an issue.

There are two ways in the development of enriched ontologies. The first choice is the development of enriched version for an existing OWL ontology. The second choice is the direct development of ontologies in the form of EnOntoModel. In the case of second choice, we first build a conceptually consistent and structured ontology, O^E. Then, developers can reformat O^E version to O by reformatting the meta-class of each domain concept to standard OWL class, and then removing the imported link of sortal meta-class ontology `sort.owl` via *remove* option of Protégé import. Note that the import link should not be moved out before reformatting concepts; otherwise concepts can be lost together with import withdrawal. Taxonomies and concept descriptions between O^E and O are almost the same except concept-level properties. By the first way, there may be different between two versions not only in taxonomies but also in concept descriptions, because enrichment process might affect to original version in order to clarify and enrich the semantics of concepts.

In order to deal with semantic heterogeneity, most matching methods need either a preprocessing before mapping or pruning off mapping results via expert-interaction, or both. For example, a mapping approach called Risk Minimization-based Ontology Matching (RiMOM) [47] is recently contributed concerning both name-based matching

and instance-based matching. In RiMOM, the developers use pre-pruning as well as post-pruning processes in their experiment. In EOM, we can say enrichment framework is a kind of pre-pruning process. However, this enrichment can improve not only mapping results between heterogeneous ontologies, but also conceptual consistency of ontologies.

Currently, I have no idea to estimate the cost of enrichment in mathmetical formula. However, the above discussion would be useful to judge the benefit and cost of EnOnto-Model.

Chapter 6

Conclusion

I have presented an approach of philosophy-based semantic enrichment and a method of enrichment-based ontology matching in order to deal with semantic heterogeneity. In this chapter, I conclude the thesis with a discussion of four topics. First, the main contributions of this thesis are summarized. Second, a list of advantages and limitations of semantic enrichment and matching method is given. Third, a brief history of my research progress is described according to the list of publications. Finally, a number of directions for future work are presented.

6.1 Summary of Contributions

First of all, recall the major aim of the thesis: "*how to deal with wide-scale semantic heterogeneity in matching between large ontologies*". I describe a summary of my contributions according to this major aim and the objectives defined in Section 1.3. In this thesis, finding semantic correspondences between two heterogeneous ontologies is approached with focus on a matching method between classified concepts using the most relevant properties which can certainly determine a correspondence between two semantically equivalent concepts. The whole approach can be partitioned into two phases: (1) the semantic enrichment phase and (2) the mapping phase. The former phase consists of modeling and implementing semantically-enriched ontologies. The later phase is for a matching method between enriched ontologies and an experiment. Now, I list the contributions of each phase in detail.

In phase (1), a distinctive approach of semantic enrichment is proposed. It is composed of two major parts: modeling and implementing.

- **Modeling :** I provided a formal semantically-enriched model of domain ontologies using some philosophical notions intuitively. This part includes the following contributions.

    1. Regarding the major aim of the thesis, I reformalized IC, existential rigidity, and external dependency, in a precise semantics.
    2. For this formalization, I presented a First-order Quantified Modal Language $\mathcal{L}^E$ that concerns varying domains of possible worlds together with the issue of actual existence in Kripke semantics. Moreover, the language focuses on a systematic formalization of ICs by considering a 2-sorted universe: individuals

and datatypes. A Kripke model that satisfies axioms $S5 + BF$ is provided to distingush rigid sorts from anti-rigid sorts, as well as ICs from local identifiers.

3. Firstly, I formalized IC by using a unary function of $\mathcal{L}^E$, denoted by ι, and thus the identity between two individuals could be determined by the entailment of equality between their IC values. Secondly, I represented IC as a property of a sort and distinguished it from other properties by the characteristic of one-to-one functional between its domain and range. Thirdly, I showed that ICs can be written in OWL by using `owl:DatatypeProperty` with three restrictions: `owl:FunctionalProperty`, `owl:InverseFunctionalProperty`, and $= 1$ cardinality. These ideas are very useful to encounter the identity of individuals in computer systems.
4. I claimed that domain concepts/classes can be represented as sorts with respect to the fact "the individuals given in a universe of discourse are countable and identifiable". Then, a formal description of sorts and the definition of subsumption relationships between sorts, are presented. Also, I observed that a sort can possess multiple ICs either IC inheritance or multiple ownICs. Moreover, I showed that sorts can be implemented by using `owl:class` with a restriction of containing at least one IC. This contribution is a foundational step to apply the pholosophical notions into formal ontologies in a practical way.
5. Following the classifications of ontological concepts in the literature of philosophy, especially by N. Guarino & C. Welty [147], and G. Guizzardi [61], I redefine four sort categories: type, quasi-type, role, and phase, explicitly using some conceptual constraints based on identity, existential rigidity, and external dependency. I proved that these sort categories are disjoint to each other by using rigidity and other distinctive properties such as ownIC, CVA, EDR, and CC. Although the former works proposed the classifications of sorts, I originally defined sort categories using such explicit properties. The advantageous point is that those sort categories can be coded by using ontology languages, particularly OWL. Also, a typical structure of sort categories is provided together with some subsumption constraints to apply in the later conceptual analysis.
6. I formalized the classification scheme of sorts as the concept-level properties (called meta-knowledge) of domain concepts. This idea of embedding a philosophy-based sort classification into a formal model of ontologies is also one of my novel contributions.
7. Finally, I defined EnOntoModel in which each sort is described with a set of individual-level properties, as well as concept-level properties. By this idea, the semantics of each concept is clarified and enriched. The main difference between ordinary ontologies and enriched ontologies is such concept-level properties. The contribution of EnOntoModel is my novelity for the purpose of enrichment-based ontology matching.

- **Implementing :** For the usability of my enrichment theory, how users can enrich their ontologies based on EnOntoModel becomes a critical issue.

1. I solved this issue by implementing sortal meta-class ontology named `sort.owl` as an open source interface for EnOntoModel. I presented the implementa-

tion framework of sortal meta-class ontology using the meta-class construct of Protégé knowledge model. Moreover, I constructed five PAL constraints and added them in the meta-class ontology for the conceptual analysis of enriched ontologies.

2. Also, I presented the development steps of semantically-enriched ontologies via Protégé OWL API, including conceptual analysis via "PAL constraints" plug-in.

In phase (2), Enrichment-based Ontology Matching (EOM) method is designed for an effective discovery of correspondences between heterogeneous ontologies and it is implemented in Java. This phase is accomplished with the following contributions.

1. I proved that there is no semantic correspondence between any two sorts of different sort categories, as shown in Figure 5.5. Thus, the matching method is divided into four sub-functions according to the sort categories. That is the first important idea of my matching method.

2. The second important idea is the usage of ICs as a major tool in determining semantic correspondences between type sorts, as well as other kinds of sorts.

3. The heuristic "*No correspondence can be found between two sorts if there is no sort equality between their type sorts*" is well-suited to define the scope of possible correspondences for a candidate sort. Thus, the third important idea is that matching functions are designed by using divide-and-conquer approach, precisely, each sub-function is invoked by another sub-function following the typical structure of sort categories shown in Figure 4.12.

4. EOM method is explained in the forms of process flow diagrams and algorithm.

5. The method is implemented in Java for matching between OWL ontologies by utilizing Jena OWL API and Protégé OWL API. Effeiciency of EOM is evaluated in terms of mathematical complexity and proved that this method could reduce the complexity of the matching process by comparing it with other methods, particularly GLUE's content learners. Moreover, an experiment is done in two real data sets, and the effectiveness of EOM is shown in terms of precision and recall.

6.2 Advantages and Limitations

The advantages of the EnOntoModel-based matching method over other mapping methods are as follows:

- Direct concept matching is initiated between the same sort categories, instead of a blind or exhaustive matching among all sorts.

- The scope of possible correspondences can be determined according to IC inheritance via type sorts. This approach is more rigorous than natural language approaches in the case of highly terminological heterogeneity.

- Semantic correspondence between two sorts is decided by direct matching between the properties, ownICs, CVA, EDR, and CC, instead of comparing all the properties belonging to the sorts. Consequently, the time complexity of EnOntoModel-based matching is less than others.

- Moreover, expert-interaction is not necessarily required to verify the matching results.

Also, some philosophical foundations are successfully added into formal ontologies to support semantic interoperability among information systems. Guarino & Welty first introduced these philosophical notions for ontological analysis. After them, Guizzardi introduced the same notions for the foundations of conceptual models, particularly UML diagrams, as a meta-model. I originally applied these ideas for ontology matching. EnOntoModel is an integrated research work of philosophy, conceptualization, formal ontologies, mathematical logic, and knowledge representation.

The limitation of this work is domain ontologies need to be developed or enriched in the form of EnOntoModel. Although I proved that my matching has an advantage in finding correspondences, the developer's effort required for the enrichment phase will be the cost of this enrichment-based matching method. However, ontologies are expert-defined conceptual models whilst folksonomies[1] are user-defined models. Therefore, it is optimistic that developers have sufficient conceptual knowledge to classify domain concepts into sort categories.

I also need to discuss the precision of correspondences by Definitions 30 and 31. Under open world assumption, a correspondence between two type sorts using this approach is always the best guess according to the known individuals and their IC values. However, this is a common issue for all methods that utilize populated ontologies in open world assumption.

This matching will be particularly attractive for information exchange in identity intensive domains such as E-commerce, social security, and trust-worthy services.

6.3 A Brief History of Research Progress

In this section, I describe a brief history of my research trend and progress according to the listed publications. This research started with a study in ontological analysis, more exactly OntoClean. I first worked out the presentation of ICs and sortal ontologies in terms of order-sorted logic (see [151, 152, 153]). Later, I moved my logic language to First-order Modal logic and started to employ IC-based heuristics for matching between sorts (see [154, 155]). In [156], I emphasized the classification of sorts based on some philosophical foundations, implementation of ICs and sorts in terms of OWL-DL, and development of sortal ontologies using Protégé OWL API. The current progress can be seen in [157, 158] where I applied QML in order to account for actual existence in Kripke semantics. Moreover, I could implement the meta-class ontology as an open source interface for EnOntoModel, and then provided a practical development of enriched ontologies.

[1] A folksonomy is an Internet-based information retrieval methodology consisting of collaboratively generated, open-ended labels (called tags) that categorize content such as Web pages, online photographs, and Web links. Two widely cited examples of websites using folksonomic tagging are `Flickr` and `Del.icio.us`.

In addition, I implemented EOM method in Java and showed experiment results in terms of precision and recall measurements. I integrated both analysis theory and mapping theory into formal ontologies concerning the issue of wide-scale semantic heterogeneity.

6.4 Future Directions

Firstly, I would like to discuss some issues in this research, mostly based on the reviews of my published papers and the former defense presentation.

1. The first issue is how to verify ICs given in enriched ontologies.

2. The second issue is how to assist modelers/developers in defining the IC value of a certain individual automatically (or semi-automatically) through a computer system.

3. The third issue is, how is the efficiency of EnOntoModel-based matching, in case modelers/developers classified domain concepts into sort categories incorrectly, and the PAL constraints could not detect such conceptual mistakes.

4. The fourth issue is empirical evaluation of EOM in comparing with other methods in terms of precision and recall.

The first three issues are important for the efficiency of EnOntoModel-based matching method. EnOntoModel encourages developers to populate ontologies. Concerning the first issue, ICs can be partially verified through the uniqueness of IC values defined for individuals by checking logical consistency via a DL reasoner. For the second issue, it is possible to generate the IC values of individuals after well-defining properties for IC, if a proper database is available. For the third issue, I am optimistic that PAL constraints defined in the meta-class ontology can detect such conceptual mistakes well, because domain concepts are classified in hierarchy with some constraints, and incorrect classification will raise conceptual inconsistency for all concepts through the same hierarchy. According to myself experience, it is rather hard to escape from the PAL constraints. However, additional constraints may need to be embeded by future experience.

I hope to implement my matching method as a Java plug-in, and do empirical evaluation by uploading as a protégé plug-in and let domain experts to evaluate it by using some ontologies that are developed for real information systems and operations. However, the implementation work for a plug-in, is not yet complete. I will continue this implementation as a part of future work.

For the evaluation of ontology mapping and alignment systems, a number of test cases are available due to EON Ontology Alignment Contest[2]. EnOntoModel-based ontology merging together with an alignment system of mapping results, is also a good direction. Currently, what I am interested in is to develop a multi-strategy dynamic matching method concerning incomplete knowledge in intelligent systems, and deciding a best-fit or first-fit matching strategy for information exchange.

[2] `http://oaei.ontologymatching.org/2004/Contest/`

Bibliography

[1] A. Bordiga, R. J. Beachman, D. L. McGuinness, L. A. Resnick :"CLASSIC: A Structural Data Model for Objects", In ACM SIGMOD International Conference on the Management of Data, Portland, Oregon, pp.58-67 (1989).

[2] A. Chagrov and M. Zakharyaschev :"Modal Logic", Oxford Science Publications (1997).

[3] A. Doan, P. Domingos, A. Halevy :"Reconciling Schemas of Disparate Data Sources: a machine learning approach", In Proceedings of ACM SIGMOD Conference, pp.509520 (2001).

[4] A. Doan, J. and Madhavan, P. Domingos, A. Halevy :"Learning To Map between Ontologies on the Semantic Web", World Wide Web Consortium 2002-WWW2002, ACM (2002).

[5] A. Farquhar, R. Fikes, J. Rice :"The Ontolingua Server: A Tool for Collaborative Ontology Construction", International Journal of Human Computer Studies, vol.46, no.6, pp.707-727 (1997).

[6] A. Gangemi, D. M. Pisanelli, G. Steve :"An Overview of the ONIONS Project: Applying Ontologies to the Integration of Medical Terminologies", Data & Knowledge Engineering, vol.31, no.(2), pp.183-220 (1999).

[7] A. Gómez-Pérez, N. Juristo, J. Pazos :"Evaluation and Assessment of Knowledge Sharing Technology", In Proceedings of Knowledge Building and Knowledge Sharing (KBKS'95), The Netherlands, IOS Press, pp.289-296 (1995).

[8] A. Gómez-Pérez, M. Fernández-López, A. de. Vicente :"Towards a method to Conceptualize Domain Ontologies", In ECAI'96 Workshop on Ontological Engineering, Hungray, pp.41-52 (1996).

[9] A. Gómez-Pérez :"Knowledge Sharing and Reuse", In Handbook of Expert Systems, CRC Chapter 10 (1998).

[10] A. Gómez-Pérez and M. D. Rojas :"Ontological Reengineering and Reuse", In 11th European Workshop on Knowledge Acquisition, Modeling and Management (EKAW'99), Springer-Verlag, LNAI.1621, pp.139-156 (1999).

[11] A. Gómez-Pérez, M. Fernández-López, O. Corcho :"Ontological Engineering with examples from the area of Knowledge Management, e-commerce and the Semantic Web", Springer (2004).

[12] A. Kalyanpur, N. Hashmi, J. Golbeck, B. Parsia :"Lifecycle of a Casual Web Ontology Development Process", In Proceedings of the WWW2004 Workshop on Application Design, Development and Implementation Issues in the Semantic Web, NYC, USA (May 18, 2004).

[13] A. Kalyanpur, E. Sirin, B. Parsia, J. Hendler :"Hypermedia Inspired Ontology Engineering Environment: SWOOP", In International Semantic Web Conference (ISWC) 2004, Hiroshima, Japan (November 7-11, 2004).

[14] A. Kalyanpur, B. Parsia, J. Hendler : "A Tool for Working with Web Ontologies", In Proceedings of the International Journal on Semantic Web and Information Systems, Vol.1, No.1 (2005).

[15] A. Kalyanpur, B. Parsia, E. Sirin, B. Cuenca-Grau, J. Hendler :"Swoop: A Web Ontology Editing Browser", Journal of Web Semantics Vol.4, no.2 (2005).

[16] A. Kalyanpur, B. Parsia, E. Sirin, J. Hendler :"Debugging Unsatisfiable Classes in OWL Ontologies", Journal of Web Semantics Vol.3, no.4 (2005).

[17] A. Maedche, B. Motik1, N. Silva1, R. Volz :"MAFRA—A MApping FRAmework for Distributed Ontologies" (2002).

[18] A. Maedche, B. Motik1, N. Silva1, R. Volz :"MAFRA—An Ontology MApping FRAmework in the Context of the Semantic Web" (2002).

[19] A. N. Kaplan :"Towards a Consistent Logical Framework for Ontological Analysis", In Proceedings of International Conference on Formal Ontology in Information Systems (FOIS-2001), ACM Press, pp.244-255 (2001).

[20] A. N. Prior : "Modality and Quantification in $S5$", Journal of Symbolic Logic, vol.21, pp.6062 (1956).

[21] A. N. Prior :"Time and Modality", Oxford University Press, Oxford (1957).

[22] A. N. Prior :"Tense Logic for Non-permanent Existents", In Paper on Time and Tense, Clarendon Press, Oxford (1968).

[23] B. Brumitt, B. Meyers, J. Krumm, A. Kern, S. Shafer :"Easyliving: Technologies for Intelligent Environments", In Proceedings of Handheld and Ubiquitous Computing Second International Symposium (HUC 2000), Springer-Verlag, pp.12-29 (2000).

[24] B. J. Wielinga, A. T. Schreiber, J. A. Breuker :"KADS: A Modeling Approach to Knowledge Engineering", Knowledge Acquisition, Special issue, vol.4, no.1 (1992).

[25] B. I. Blum :"Beyond Programming", Oxford University Press, New York (1996).

[26] B. Johanson, A. Fox, T. Wingograd :"The Interactive Workspace Project: Experiences with Ubiquitous Computing Rooms", IEEE Pervasive Computing Magazine, vol.1, no.2 (2002).

[27] B. Miller :"Exists and Existence", Review of Metaphysics, vol.40, pp.237-270 (1987).

[28] B. Miller :"Existence", In Edward N. Zalta, ed. Stanford Encyclopedia of Philosophy (2002).

[29] B. Smith :"Ontologies and Information Systems" (2003).

[30] B. Swartout, P. Ramesh, K. Knight, T. Russ :"Toward Distributed Use of large-Scale Ontologies", In AAAI'97 Spring Symposium on Ontological Engineering, Stanford University, Calfornia, pp.138-148 (1997).

[31] C. Batini, M. Lenzerine, S. B. Navathe :"Comparison of Methodologies for Database Schema Integration", ACM Computing Surveys, vol.18, no.4, pp.323-364 (1986).

[32] C. Beierle :"An Order-sorted Logic for Knowledge Representation Systems", Artificial Intelligence, vol.55, pp.149-191 (1992).

[33] C. Menzel :"Actualism", In E. N. Zalta, editor, The Stanford Encyclopedia of Philosophy (2005).

[34] C. S. Chihara :"The Worlds of Possibility: Modal Realism and the Semantics of Modal Logic", Clarendon Press, Oxford (1998).

[35] C. Welty and W. Andersen :"Towards OntoClean 2.0: A Framework for Rigidity", Journal of Applied Ontology, vol.1, no.1 (October 2005).

[36] D. A. Waterman :"A Guide to Expert Systems", Addison-Wesley, Boston, Massachusetts (1986).

[37] D. Brickley and R. V. Guha :"RDF Vocabulary Description language 1.0: RDF Schema", W3C Working Draft (2003).

[38] D. B. Lenat :"Building Large Knowledge-base Systems: Representation and Inference in the Cyc Project", Addison-Wesley, Boston, Massachusetts (1990).

[39] D. B. Lenat :"Cyc: A Large-scale Investment in Knowledge Infrastructure", ACM, vol.38, no.11, pp.32-38 (1995).

[40] D. Calvanese, D. G. Giuseppe, M. Lenzerini :"Ontology of Integration and Integration of Ontologies", In Proceedings of the International Workshopon Description Logic (DL 2001) (2001).

[41] D. F. Belardinelli :"Quantified Modal Logic and the Ontology of Physical Objects", PhD Thesis (2006).

[42] D. McGuinness, R. Fikes, J. Rice, S. Wilder :"An Environment for Merging and Testing Large Ontologies", In Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning(KR-2000), Colorado, USA (2000).

[43] D. Raggett, A. Le Hors, I. Jacobs :"HTML 4.01 Specification", W3C Recommendation (1999).

[44] E. Downs, P. Clare, I. Coe :"Structured Analysis and Design method (SSADM), Prentice Hall, New Jersey (1998).

[45] E. Friedman-Hill :"Jess in Action: Java Rule-based system", Manning Publication Company, Greenwich, Connecticut (2003).

[46] E. Hirsch :"The Concept of Identity", Oxford University Press, New York, Oxford (1982).

[47] J. Tang, J. Li, B. Liang, X. Huang, Y. Li, K. Wang: "Using Bayesian Decision for Ontology Matching", In Journal of Web Semantics, WEBSEM-88, Elsevier (2006).

[48] E. J. Lowe :"What is a Criterion of Identity", The Philosophical Quarterly, vol.39, pp.1-21 (1989).

[49] E. J. Lowe :"Kinds of Being: A Study of Individuation, Identity, and the Logic of Sortal Terms", Oxford:Basil Blackwell (1989).

[50] E. Motta :"Reusable Components for Knowledge Modelling: Principles and Case Studies in Parametric Design", IOS Press, Amsterdam, The Netherlands (1999).

[51] F. Badder and B. Hollunder :"KRIS: Knowledge Representation and Inference System", SIGART Bulletin, vol.2, no.3, pp.8-14 (1991).

[52] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider :"The Description Logic Handbook", Cambridge University Press (2003).

[53] F. Belardinelli :"Quantified Modal Logic and Ontology of Physical Object", PhD Thesis, Scuola Normale Superiore (SNS) in Pisa (2006).

[54] F. Peter, H. Patrick, I. Horrocks :"OWL Web Ontology Language Overview", W3C Recommendation (10 February 2004).

[55] F. Peter, H. Patrick, I. Horrocks :"OWL Web Ontology Language Guide", W3C Recommendation (10 February 2004).

[56] F. Peter, H. Patrick, I. Horrocks :"OWL Web Ontology Language Reference", W3C Recommendation (10 February 2004).

[57] F. Peter, H. Patrick, I. Horrocks :"OWL Web Ontology Language Semantics and Abstract Syntax", W3C Recommendation (10 February 2004).

[58] G. Corsi :"A Unified Completeness Theorem for Quantified Modal Logics", Journal of Symbolic Logic, vol(67), pp.14831510 (2002).

[59] G. E. Hughes and M. J. Cresswell :"A New Introduction to Modal Logic", Routledge, London (2003).

[60] G. Guizzardi, G. Wagner, and M. Sinderen :"A Formal Theory of Conceptual Modeling Universals", In proceedings of the workshop on philosophy and informatics (WSPI) (2004).

[61] G. Guizzardi :"Ontological Foundations for Structural Conceptual Models", PhD Thesis, Telematics Institute, The Nethelands (2005).

[62] Google : http://google.blognewschannel.com/index.php/archives/2005/01/23/google-at-how-many-billion-9-11/ (2006)

[63] G. Steve, A, Gangemi, D. M. Pisanelli :"Integrating Medical terminologies with ONIONS Methodology", In Information Modeling and Knowledge Bases VIII, IOS Press, Amsterdam, The Netherlands (1998).

[64] G. Stumme and A. Maedche :"FCA-MERGE: Bottom-up Merging of Ontologies", In Proceedings of 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), Seattle, Washington, pp.225-234 (2001).

[65] G. Wiederhold :"An Algebra for Ontology Composition", In Proceedings of 1994 MontereyWorkshop on Formal Methods, pp.56-61, CA, USA (1994).

[66] H. Chalupsky :"Ontomorph: A Translation System for Symbolic Logic", In KR2000: Principles of Knowledge Representation and Reasoning, Morgan Kaufmann, pp.471482, San Francisco, CA (2000).

[67] H. Chalupsky :"A Translation System for Symbolic Knowledge", In Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (2000).

[68] H. D. Aumueller and E. R. Massmann :"A System for Flexible Combination of Schema Matching Approaches", In Proceedings of the Very Large Data Base conference (VLDB), pp.610-621 (2001)

[69] H. D. Aumueller and E. R. Massmann :"Schema and Ontology Matching with COMA++", In Proceedings of SIGMOD, pp.610-621 (2005)

[70] H. Hermes :"Introduction to Mathematical Logic", Springer-Verlag, Berlin, Germny (1973).

[71] H. Knublauch, O. Dameron, M. A. Musen :"Weaving the Biomedical Semantic Web with the Protg OWL Plugin", In the First International Workshop on Formal Biomedical Knowledge Representation, Whistler, Canada (2004).

[72] H. Knublauch, M. A. Musen, A. L. Rector :"Editing Description Logic Ontologies with the Protg OWL Plugin", In International Workshop on Description Logics (DL2004), Whistler, Canada (2004).

[73] H. Knublauch, R. W. Fergerson, N. F. Noy, M. A. Musen :"The Protg OWL Plugin: An Open Development Environment for Semantic Web Applications", In Third International Semantic Web Conference, Hiroshima, Japan (2004).

[74] H. Knublauch :"Ontology-Driven Software Development in the Context of the Semantic Web: An Example Scenario with Protg/OWL", In International Workshop on the Model-Driven Semantic Web, Monterey, CA (2004).

[75] H. Noonan :"Identity", Dartmouth, Aldershot, USA (1993).

[76] H. S. Leonard and N. Goodman :"The Calculus of Individuals and Its Uses", Journal of Symbolic Logic, vol.5, pp.45-55 (1940).

[77] "IEEE Standard Glossary of Software Engineering Terminology", IEEE Computer Society, new York, IEEE Std.610-121990 (1990).

[78] "IEEE Standard for Developing Software Life Cycle processes", IEEE Computer Society, new York, IEEE Std.1074-1995 (1996).

[79] I. Horrocks, D. Fensel, F. Harmelen, F. Decker, M. Erdmann, M. Klein :"OIL in a Nutshell", In 12th International Conference in Knowledge Engineering and Knowledge management (EKAW'00), France, LNAI 1937, Springer-Verlag (2000).

[80] I. Horrocks :"A Denotational Semantics for OIL-Lite and Standard OIL", Technical Report (2000).

[81] I. Horrocks and F. Harmelen :"Reference Description of DAML+OIL Ontology Markup Language", Technical Report (2001).

[82] I. Horrocks, U. Sattler and S. Tobies :"Reasoning with Individuals for the Description Logic SHIQ" (2001).

[83] I. Horrocks, P. F. Patel-Schneider, F. Harmelen :"From $SHIQ$ and RDF to OWL: The Making of a Web Ontology Language", International Semantic Web Conference (2001).

[84] I. Horrocks and U. Sattler :"Ontology Reasoning in the $SHOQ$(D) Description Logic", In Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), pp.199-204 (2001).

[85] I. Horrocks, F. P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean :"SWRL: A Semantic Web Rule Language Combining OWL and RuleML", Version 0.5 (November 2003).

[86] J. C. Arpírez, O. Corcho, M. Ferńandez-López, A. Gómez-Pérez :"WebODE: A Scalable Ontological Engineering Workbench", In First International Conference on Knowledge Capture (KCAP'01), Victoria, Canada, ACM Press, pp.6-13 (2001).

[87] J. C. Arpírez, O. Corcho, M. Ferńandez-López, A. Gómez-Pérez :"WebODE in a Nutshell", AI Magazine (2003).

[88] J. Domingue :"Tadzebao and WebOnto: Discussing, Browsing, and Editing Ontologies on the Web", In 11th International Workshop on Knowledge Acquisition, Modeling, and management (KAW'98), Banff, Canada, Knowledge Management, vol.4, pp.1-20 (1998).

[89] J. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubezy, H. Eriksson, N. F. Noy, S. W. Tu : "The Evolution of Protg: An Environment for Knowledge-Based Systems Development" (2002).

[90] J. Hintikka :"Modality and quantification", Theoria, vol.27, pp.119128 (1961).

[91] J. Hintikka :"Models for Modalities", Reidel Publishing Company, Dordrecht (1969).

[92] J. Madhavan, P. A. Bernstein, P. Domingos, A. Halevy :"Representing and Reasoning about Mappings between Domain Models, In Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence (AAAI 2002), AAAI Press, pp.8086, Edmonton, Alberta, Canada (2002).

[93] J. McCarthy :"Applications of Circumscription to Formalizing Common Sense Knowledge", Artificial Intelligence, vol.28, pp.89-116 (1986).

[94] J. W. Garson :"Quantification in Modal Logic", In D. Gabbay and F. Gunthner, editors, Handbook of Philosophical Logic, Kluwer Academic Publishers, Dordrecht, pp.267323 (2001).

[95] J. W. Lloyd :"Foundations of Logic Programming", Springer-Verlag, New York (1993).

[96] K. C. Yu :"Effective Partial Ontology Mapping in a Pervasive Computing Environment", PhD Thesis, The University of Hong Kong (2004).

[97] K. Fine :"Model Theory for Modal Logic", part i - the DE RE/DE DICTO distinction, Journal of Philosophical Logic, vol.7, pp.125156 (1978).

[98] K. Kaneiwa :"Order-Sorted Logic Programming with Predicate Hierarchy", Artificial Intelligence, vol.158, pp.155-188 (2001).

[99] K. Sparck-Jones :"What is New about the Semantic Web? Some Questions", SIGIR Forum, vol.38, no.2 (2004).

[100] K. Wong :"Rigid designation, Existence and Semantics for Quantified Modal Logic", Department of Philosophy, The Chinese University of Hong Kong (2003).

[101] L. Goble :"The Blackwell Guide to Philosophical Logic", Blackwell (2001).

[102] M. A. Musen, S. W. Tu, H. Eriksson, J. H. Gennari, A. R. Puerta : " PROTEGE-II: An Environment for Reusable Problem-Solving Methods and Domain Ontologies", In International Joint Conference on Artificial Intelligence, Chambery, Savoie, France (1993).

[103] M. A. Musen, J. H. Gennari, H. Eriksson, S. W. Tu, A. R. Puerta : "PROTEGE II: Computer Support For Development Of Intelligent Systems From Libraries Of Components", The Eighth World Congress on Medical Informatics, Vancouver, B.C., Canada, pp.766-770 (1995).

[104] M. A. Musen :"Domain Ontologies in Software Engineering: Use of Protege with the EON Architecture", Methods of Information in Medicine, vol.37, pp.540-550 (1998).

[105] M. Blázquez, M. Ferńandez-López, J. M. García-Pinar, A. Gómez-Pérez :"Building Ontologies at the Knowledge level using the Ontology Design Environment", In 11th International Workshop on Knowledge Acquisition, Modeling and Management (KAW'98), Canada, pp.1-15 (1998).

[106] M. Dean and G. Schreiber :"OWL Web Ontology Language Reference", W3C Recommendation (2003).

[107] M. Dertouzos :"The Oxygen Project", Scientific American, vol.282, no.3, pp.52-63 (1999).

[108] M. Ehrig and Y. Sure :"Ontology Mapping - An Integrated Approach", In Proceedings of European Semantic Web Symposium (ESWS), pp.76-91 (2004).

[109] M. Ferńandez-López, A. Gómez-Pérez, N. Juristo :"METHONTOLOGY: From Ontological Art Towards Ontological Engineering", In Spring Symposium on Ontological Engineering of AAAI, Stanford University, pp.33-40 (1997).

[110] M. Ferńandez-López, A. Gómez-Pérez, A. Pazos, J. Pazos :"Building a Chemical Ontology using METHONTOLOGY and the Ontology Design Environment", IEEE Intelligent Systems & their Applications, vol.4, no.1, pp.37-46 (1999).

[111] M. Grüninger and M. S. Fox :"Methodology for the Design and Evaluation of Ontologies", In IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing, pp.1-10 (1995).

[112] M. Denny :"Ontology Building: A Survey of Editing Tools", Technical Report, XML.com (2002).

[113] M. Ehrig and S. Stabb :"QOM—Quick Ontology Mapping", Technical report, University of Karlsruhe, Institute AIFB (2004).

[114] M. J. Cresswell':"A Blackwell Guide to Philosophical Logic", Blackwell (2001).

[115] M. J. Loux :"The Possible and the Actual", Cornell University Press, Ithaca, NY (1979).

[116] M. Kifer, G. Lausen, J. Wu :"Logical Foundations of Object-oriented and Frame-based Languages", Journal of ACM, vol.42, no.4, pp.741-843 (1995).

[117] M. Klein :"Combining and Relating Ontologies: An Analysis of Problems and Solutions", In Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01), Workshop: Ontologies and Information Sharing, Seattle, USA (2001).

[118] M. Musen :"Annotated support for Building and Extending Expert Models", Machine Learning, vol.4, pp.347-376 (1989).

[119] M. R. Genesereth and R. E. Fikes :"Knowledge Interchange Format, Version 3.0, Reference Manual", Technical Report Logic 92-1, Computer Science Department, Stanford University, California (1992).

[120] M. Satyanarayanan :"Pervasive Computing: Vision and Challenges", IEEE Personal Communications, pp.10-17 (2001).

[121] M. Schmidt-Schau and G. Smolka: "Attributive Concept Descriptions with Complements", Artificial Intelligence, vol.48, no.1, pp.1-26 (1991).

[122] M. Shaw and B. Gaines :"Comparing Conceptual Structures: Consensus, Conflict, Correspondence and Contrast", In the journal of Knowledge Acquisition, vol.1, pp.341-363 (1989).

[123] M. Uschold and M. King :"Towards A Methodology for Building Ontologies", In IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing, Canada, vol.6, pp.1-10 (1995).

[124] M. Uschold and M. Grüninger :"Ontologies: Principles, Methods, and Applications", Knowledge Engineering Review, vol.11, no.2, pp.93-155 (1996).

[125] M. Uschold and R. Jasper :"A Framework for Understanding and Classifying Ontology Applications", In IJCAI'99 Workshop on Ontology and Problem Solving Methods, Amsterdam, The Netherlands, vol.18, no.11, pp.1-12 (1999).

[126] M. Vermeer :"Semantic Interoperability for Legacy Databases", PhD Thesis, University of Twente, The Netherlands (1997).

[127] NCITS :"Draft proposed American National Standard for Knowledge Interchange Format", National Committee for Information Technology Standards, Technical Committee T2 (Information Interchange and Interpretation) (1998).

[128] N. F. Noy and M. A. Musen :"An Algorithm for Merging and Aligning Ontologies: Automation and Tool Support", In Sixteenth National Conference on Artificial Intelligence (AAAI-99), Workshop on Ontology Management, Orlando, FL (1999).

[129] N. F. Noy, W. Grosso, M. A. Musen : "Knowledge-Acquisition Interfaces for Domain Experts: An Empirical Evaluation of Protege-2000", In Twelfth International Conference on Software Engineering and Knowledge Engineering (SEKE2000), Chicago (2000).

[130] N. F. Noy, R. W. Fergerson, M. A. Musen: "The knowledge model of Protege-2000: Combining Interoperability and Flexibility", In 2th International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000), Juan-les-Pins, France (2000).

[131] N. F. Noy and M. A. Musen :"PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment", In Seventeenth National Conference on Artificial Intelligence (AAAI-2000), Austin, TX (2000).

[132] N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, M. A. Musen : "Creating Semantic Web Contents with Protege-2000", IEEE Intelligent Systems, vol.16, no.2, pp.60-71 (2001).

[133] N. F. Noy and D. L. McGuinness :"Ontology Development 101: A Guide to Creating Your First Ontology" (2001).

[134] N. F. Noy and M. A. Musen :"Anchor-PROMPT: Using Non-Local Context for Semantic Matching", In Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001), Seattle, WA (2001).

[135] N. F. Noy and M. A. Musen :"Evaluating Ontology-Mapping Tools: Requirements and Experience" (2002).

[136] N. F. Noy and M. A. Musen :"PROMPTDIFF: A Fixed-Point Algorithm for Comparing Ontology Versions", In Proceedings of the 18th National Conference on Artificial Intelligence (AAAI'02) (2002).

[137] N. F. Noy and M. A. Musen: "The PROMPT Suite: Interactive Tools For Ontology Merging And Mapping", In International Journal of Human-Computer Studies, vol.59, pp.983-1024 (2003).

[138] N. F. Noy: "Ontology Mapping and Alignment", SSSW2005 (2005).

[139] N. Guarino, M. Carrara, P. Giaretta :"Formalizing Ontological Commitments", In Proceedings of AAAI'94 (1994).

[140] N. Guarino, M. Carrara, P. Giaretta :"An Ontology of Meta-Level Categories", In D. J., E. Sandewall and P. Torasso (eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR94). Morgan Kaufmann, San Mateo, CA: 270-280 (1994).

[141] N. Guarino and P. Giaretta :"Ontologies and Knowledge Bases: Towards a Terminological Clarification", In Knowledge Building and Knowledge Sharing (KBKS'95), University of Twente, Enschede, The Netherlands, IOS Press, pp. 25-32 (1995).

[142] N. Guarino :"Formal Ontologies in Information Systems", In International Conference on Formal Ontology in Information Systems (FOIS'98), Trento, Italy, IOS Press, pp.3-15 (1998).

[143] N. Guarino :"Some Ontological Principles for Designing Upper Level Lexical Resources", In Proceedings of the First International Conference on Lexical Resources and Evaluation, Granada, Spain (28-30 May 1998).

[144] N. Guarino :"The role of Identity Conditions in Ontology Design", In Proceedings of the IJCAI-99 Workshop on Ontology and Problem Solving Methods (KRRS), Stockholm, Sweden (August 2, 1999).

[145] N. Guarino and C. Welty :"Towards a Methodology for Ontology-based Model Engineering", In, Bezivin, J. and Ernst, J., eds, Proceedings of the ECOOP-2000 Workshop on Model Engineering (June, 2000).

[146] N. Guarino and C. Welty :"Identity, Unity, and Individuality: Towards a Formal Toolkit for Ontological Analysis", In W. Horn, ed., Proceedings of ECAI-2000: The European Conference on Artificial Intelligence. IOS Press, Amsterdam (August, 2000).

[147] N. Guarino and C. Welty :"Ontological Analysis of Taxonomic Relationships", In, Laender, A. and Storey, V., eds, Proceedings of ER-2000: The 19th International Conference on Conceptual Modeling, Springer-Verlag (October, 2000).

[148] N. Guarino, and C. Welty :"A Formal Ontology of Properties", In, Dieng, R., and Corby, O., eds, Proceedings of EKAW-2000: The 12th International Conference on Knowledge Engineering and Knowledge Management", AAAI Press, Menlo Park (October, 2000).

[149] N. Guarino and C. Welty :"An Overview of OntoClean", In Steffen Staab and Rudi Studer, eds., The Handbook on Ontologies, Berlin:Springer-Verlag, pp.151-172 (2004).

[150] N. Hanssens, A. Kulkarni, R. Tuchida, T. Horton :"Building Agent-based Intelligent Workspaces", MIT Laboratory for Computer Science and MIT Artificial Intelligence Laboratory, MIT Oxygen Project (2002)

[151] N. N. Tun and S. Tojo :"Consistency Maintenance for Ontological Knowledge Updating", In proceedings of IEEE International Conference on information Reuse and Integration (IRI-2004), pp.388-393 (2004).

[152] N. N. Tun and S. Tojo :"Inheritance of Multiple Identity Conditions in Order-Sorted Logic", In Proceedings of 17th Australian Joint International Conference on Artificial Intelligence (AI-2004), Springer LNAI 3339: Advances in Artificial Intelligence, Springer-Verlag LNAI.3339, pp.1187-1193 (2004).

[153] N. N. Tun and S. Tojo :"Consistent Hybrid Knowledge Representation in Order-Sorted Logic", In Proceedings of International Conference on Computational Intelligence (ICCI 2004), IJCI Press, pp.349-353 (2004).

[154] N. N. Tun and S. Tojo :"Unification of Sorts Among Local Ontologies for Semantic Web Applications", In Proceedings of 4th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and databases (AIKED05), WSEAS Transaction of Computers Issue.2, Vol.4, ISSN.1109-2750, pp.123-129 (2005).

[155] N. N. Tun and S. Tojo :"IC-based Ontology Expansion in Devouring Accessibility", Australian Ontology Workshop (AOW 2005), Australian Computer Society, vol.58, pp.99-106 (2005).

[156] N. N. Tun and S. Tojo :"Identity Conditions for Ontological Analysis", In Proceedings of Knowledge Science, Engineering, and management, Springer verlag, LNAI 4092, pp.418-430 (2006).

[157] N. N. Tun and S. Tojo : "Semantic Enrichment in Ontologies for Matching", International Journal of Semantic Web Information Systems (IJSWIS), IDEA Group Ltd., volume 2, issue 4, pp.33-67 (2006).

[158] N. N. Tun and S. Tojo: "EnOntoModel: A Philosophy-based Conceptual Model for Semantically-Enriched Ontologies", In International Journal of Intelligent Information Technologies (IJIIT) (2007).

[159] O. Corcho, M. Ferńandez-López, A. Gómez-Pérez, O. Vicente :"WEBODE: An Integrated Workbench for Ontology Representation, Reasoning, and Exchange", In 13th International Conference on Knowledge Engineering and Management (EKAW'02), Spain, LNAI 2473, Springer-Verlag, pp.138-153 (2002).

[160] O. Corcho and A. Gómez-Pérez :"A Roadmap to Ontology Specification Languages", In Proceedings of 12th International Conference on Knowledge Engineering and Management (2002).

[161] O. Lassila and R. Swick :"Resource Description Framework (RDF) Model and Syntax Specification", W3C Recommendation (1999).

[162] O. Lassila and D. McGuinness :"The Role of Frame-based Representation on the Semantic Web", Technical Report, Knowledge System Laboratory, Stanford University, California (2001).

[163] P. D. Karp, V. Chaudheri, J. Thomere :"XOL: An XML-based Ontology Exchange Language, Version 3.0", Technical Report (1999).

[164] P. Mitra and G. Wiederhold :"Resolving Terminology Heterogeneity in Ontologies", In Proceedings of ECAI'02 Workshop on Ontologies and Semantic Interoperability, Lyon, France (2002).

[165] P. Mitra, N. F. Noy, A. R. Jaiswal :"OMEN: A Probabilistic Ontology Mapping Tool", In Workshop on Meaning coordination and negotiation at the Third International Conference on the Semantic Web (ISWC-2004) (2004).

[166] P. Shvaiko and J. Euzenat :"A Survey of Schema-based Matching", `http://www.ontologymatching.org` (2005).

[167] P. Visser, D. M. Jones, T. Bench-Capon, M. Shave :"An Analysis of Ontological Mismatches: Heterogeneity versus Interoperability", In AAAI 1997 Spring Symposium on Ontological Engineering, Stanford, USA (1997).

[168] P. Visser, D. M. Jones, T. Bench-Capon, M. Shave :"Assessing Heterogeneity by Classifying Ontology Mismatches", In Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS98), Trento, Italy (1998).

[169] R. Corazzon : "Ontology: A Resource Guide for Philosophers", Christian Wolff's "Philosophia prima sive Ontologia" (1729).

[170] R. C. Barcan :"The Deduction Theorem in a Functional Calculus of First Order based on Strict Implication", Journal of Symbolic Logic, vol.11, pp.115118 (1946).

[171] R. C. Barcan :"A Functional Calculus of First Order based on Strict Implication", Journal of Symbolic Logic, vol.11, pp.116 (1946).

[172] R. C. Barcan :"The Identity of Individuals in a Strict Functional Calculus of Second Order", Journal of Symbolic Logic, vol.12, pp.1215 (1947).

[173] R. de Hoog :"Methodologies for Building Knowledge Based systems: Achievements and Prospects", In Handbook of Expert Systems, CRC Press, Chapter 1 (1998).

[174] R. J. Brachman, R. E. Fikes, H. J. Levesque :"Krypton: A Functional Approach to Knowledge Acquisition", IEEE Computer, vol.16, no.10, pp.67-73 (1983).

[175] R. J. Beachman and J. G. Schmolze :"An Overview of the KL-ONE Knowledge Representation System", Cognitive Science, vol.9, no.2, pp.171-216 (1985).

[176] R. J. E. Gamma, H. Richard, J. Vlissides :"Design Patterns: Elements of Reusable Object Oriented Software", Addison-Wesley.

[177] R. Kent :"The Information Flow Foundation for Conceptual Knowledge Organization", In Proceedings of the 6th International Conference of the International Society for Knowledge Organization (ISKO), Toronto, Canada (August 2000).

[178] R. M. MacGregor :"Using a Description Classifier to Enhance Deductive Inference", In Proceedings Seventh IEEE Conference on AI Applications, pp.141-147 (1991).

[179] R. M. MacGregor :"Restrospective on LOOM", Technical Report, Information Science Institute, University of Southern California (1999).

[180] R. Mizoguchi, J. Vanwelkenhuysen, M. Ikeda :"Task Ontology for Reuse of problem Solving Knowledge", In Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing (KBKS'95), University of Twente, Enschede, The Netherlands, IOS Press, pp.46-57 (1995).

[181] R. Neches, R. E, Fikes, T. Finin, T. R. Gruber, T. Senator, W. R. Swartout :"Enabling technology for Knowledge Sharing", AI Magazine, vol.12, no.3, pp.36-56 (1991).

[182] R. Studer, V. R. Benjermins, D. Fensel :"Knowledge Engineering: Principles and Methods", IEEE Transactions on Data and Knowledge Engineering, vol.25, no.1, pp.161-197 (1998).

[183] R. S. Pressmann :"Software Engineering: A Practioner's Approach, 5th Edition, McGraw-Hill, New York (2000).

[184] S. Bechhofer, I. Horrocks, C. Goble, R. Stevens :"OILED: A Reasonable Ontology Editor for the Semantic Web", In Joint German/Austrian Conference on Artificial Intelligence, pp.396-408 (2001).

[185] S. Bowers and L. Delcambre :"Representing and Transforming Model-based Information", In Proceedings of the FirstWorkshop on the Semantic Web at the Fourth European Conference on Digital Libraries (2000).

[186] S. Ceri and J. Widom :"Managing Semantic Heterogeneity with Production Rules and Persistent Queues", In Proceedings of the 19th VLDB Conference, pp.108-119, Dublin, Ireland (1993).

[187] S. Hakkarainen :"Dynamic Aspect and Semantic Enrichment in Schema Comparison", PhD Thesis, Stockkhol University (1999).

[188] S. Hakkarainen, L. Hella, S. Tuxen, G. Sindre :"Evaluating the Quality of Web-based Ontology Building Methods: A Framework and a Case Study", In Proceedings of the 6th International Baltic Conference on Database and Information Systems, Latvia, Springer-Verlag (2004).

[189] S. Kripke :"A Completeness Theorem in Modal Logic", Journal of Symbolic Logic, vol.24, pp.114 (1959).

[190] S. Kripke :"Semantical Considerations on Modal Logic", In L. Linsky (ed.), Reference and Modality, Oxford, Oxford University Press, pp.63-72 (1963).

[191] S. Kripke :"Semantical Analysis of Modal Logic I, Normal Propositional Calculi", Zeitschrift fr Mathematische Logik und Grundlagen der Mathematik 9, pp.67-96 (1963).

[192] S. Kripke :"Semantical Analysis of Modal Logic II, Non-normal Modal Propositional Calculi", in J. W. Addison et al. (eds.), The Theory of Models, Amsterdam, North-Holland, pp.206-220 (1965).

[193] S. Kripke :"Identity and Necessity", In M. Munitz (ed.), Identity and Individuation, New York, New York University Press, pp.135-164 (1971).

[194] S. Kripke :"Naming and Necessity", In D. Davidson and G. Harman, (eds.), Semantics of Natural Language, Dordrecht, D. Reidel, pp.253-355, (1972).

[195] S. Lesniewski :"Foundations of the General Theory of Sets", ed. S. J. Surma, J. Srzednicki, D. I. Barnett, and F. V. Rickey, Dordrecht: Kluwer, Vol.1, pp.129-173, original from 1916 (1992).

[196] S. Luke and J. D. Heflin :"SHOE 1.01 Proposed Specification", Technical Report, Parallel Understanding Systems Group, Department of Computer Science, University of Maryland (2000).

[197] S. Melnik, H. Garcia-Molina, E. Rahm :"Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching", In Proceedings of the International Conference on Data Engineering (ICDE) (2002).

[198] S. T. March :"Special Issue on Heterogeneous Databases", ACM Computing Surveys, vol.22, no.3 (1990).

[199] S. W. Tu, H. Eriksson, J. Gennari, Y. Shahar, M. A. Musen: "Ontology-Based Configuration of Problem-Solving Methods and Generation of Knowledge-Acquisition Tools: Application of PROTEGE-II to Protocol-Based Decision Support", Artificial Intelligence in Medicine, issue.7, pp.257-289 (1995).

[200] T. Berners-Lee, J. Hendler, O. Lassila :"The Semantic Web", Scientific American, vol. 284, no.5, pp.35-43 (2001).

[201] T. Berners-Lee and E. Miller :"The Semantic Web Lifts Off", In ERCIM News, no.51, (October 2002).

[202] T. Berners-Lee, R. T. Fielding, L. Masinter :"Uniform Resource Identifier (URI): Generic Syntax", IETF REP 3986 (standards track), Internet Eng. Task Force (January 2005).

[203] T. Berners-Lee, N. Shadbolt, W. Hall :"The Semantic Web Revisited", IEEE Intelligent Systems published by the IEEE Computer Society (2006).

[204] T. Brauner and S. Ghilardi: "First-order Modal Logic", In P. Blackburn, J. van Benthem, and F. Wolter, editors, Handbook of Modal Logic, Elsevier (2000).

[205] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler :"Extensible Markup Language (XML) 1.0", W3C Recommendation (2000).

[206] T. E. Rothenfluh, J. H. Gennari, H. Eriksson, A. R. Puerta, S. W. Tu, M. A. Musen: "Reusable Ontologies, Knowledge-Acquisition Tools, and Performance Systems: PROTEGE-II Solutions to Sisyphus-2", In the 8th Banff Knowledge Acquisition for Knowledge-Based Workshop, Banff, Alberta, vol.43, pp.1-30 (1994).

[207] T. R. Gruber :"Ontolingua: A Mechanism to support Portable Ontologies", Technical Report KSL-91-66, Knowledge System Laboratory, Stanford University, California (1992).

[208] T. R. Gruber :"Towards Principles for the Design of Ontologies used for Knowledge Acquisition", In International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation, Kluwer Academic Publishers (1993).

[209] T. R. Gruber :"Towards Principles for the Design of Ontologies used for Knowledge Sharing", In International Workshop on Formal Ontology on Conceptual Analysis and Knowledge Representation, Italy, Kluwer Academic Publishers (1993).

[210] T. R. Gruber :"Translation Approach to Portable Ontology Specification", Knowledge Acquisition, vol.5, no.2, pp.199-220 (1993).

[211] T. R. Gruber and G. Olsen:"An Ontology for Engineering Mathematics", In Fourth International Conference on Principles of Knowledge Representation and Reasoning, Bonn, Germany, Morgan Kaufmann Publishers, pp.258-269 (1994).

[212] U. Sattler: "A Concept Language Extended with Different Kinds of Transitive Roles", LNAI vol.1137, Springer-Verlag (1996).

[213] V. K. Chaudhri, A. Farquhar, R. Fikes, P. D. Karp, J. P. Rice :"Open Knowledge-base Connectivity 2.0.3", Techinical Report (1998).

[214] W. Andersen and C. Menzel :"Modal Rigidity in the OntoClean Methodology", In Vieu and Varzi (eds.), Formal Ontology and Information Systems: Collected Papers from the Fifth International Conference, ISO Press (2004).

[215] W. N. Borst :"Construction of Engineering Ontologies", Center for Telematica and Information Technology, University of Tweenty, Enschede, The Netherlands (1997).

[216] W. V. O. Quine :"Ontological Relativity and Other Essays", Columbia University Press (1969).

[217] X. Su and L. Ilebrekke :"A Comparative Study of Ontology Languages and Tools", In Proceedings of the Seventh IFIP-WG8.1 International Workshop on Evaluating of Modeling Methods in Systems Analysis and Design (EMMSAD'02 (2002).

[218] X. Su :"Semantic Enrichment for Ontology Matching", PhD Thesis, Norwegian University of Science and Technology (2004)

[219] X. Su and L. Ilebrekke :"Using a Semiotic Framework for a Comparative Study of Ontology Languages and Tools", Information Modeling Methods and Methodologies, IDEA Group Publishing (2004).

[220] Y. Kalfoglou and M. Schorlemmer :"IF-Map: An Ontology Mapping Method based on Information-Flow Theory", In the 1st International Conference on Ontologies, Databases and Applications of Semantics (ODBASE'02) (2002).

[221] Y. Kalfoglou and M. Scholemmer :"Ontology Mapping: The State of the Art", In the jounal of Semantic Interoperability and Integration (2005).

[222] Y. Sure, M. Erdmann, J. Angele, S. Stabb, R. Studer, D. Wenke :"OntoEdit: Collaborative Ontology Engineering for the Semantic Web", In the First International Semantic Web Conference (ISWC'02), Italy, LNCS.2342, Springer-Verlag, pp.221-235 (2002).

[223] Y. Sure, S. Stabb, J. Angele :"OntoEdit: Guiding Ontology Development by Methodology and Inferencing", Confederated International Conferences CoopIS, DOA, ODBASE 2002, University of Carlifornia, Springer-Verlag, pp.1205-1222 (2002).

Publications

Journal Publications:

[1] N. N. Tun and S. Tojo: "EnOntoModel: A Semantically-Enriched Model for Ontologies", In International Journal of Intelligent Information Technologies (IJIIT), Oakland University (2007).

[2] N. N. Tun and S. Tojo: "Semantic Enrichment in Ontologies for Matching", International Journal of Semantic Web Information Systems (IJSWIS), IDEA Group Ltd., volume 2, issue 4, pp.33-67 (2006).

[3] N. N. Tun and S. Tojo: "Unification of Sorts Among Local Ontologies for Semantic Web Applications", In WSEAS Transaction of Computers Issue.2, Vol.4, ISSN.1109-2750, pp.123-129 (2005).

Conference Publications:

[4] N. N. Tun and S. Tojo: "Identity Conditions for Ontological Analysis", In Proceedings of the First International Conference in Knowledge Science, Engineering and Management (KSEM06), Guilin, China, Springer-Verlag LNAI.4092, pp.418-430 (2006).

[5] N. N. Tun and S. Tojo: "IC-based Ontology Expansion in Devouring Accessibility", In Proceedings of Australasian Ontology Workshop (AOW 2005), Vol.58, pp.99-106, Sydney, Australia (2005).

[6] N. N. Tun and S. Tojo: "Consistent Hybrid Knowledge Representation in Order-Sorted Logic", In Proceedings of International Conference on Computational Intelligence (ICCI 2004), IJCI Press, pp.349-353 (2004).

[7] N. N. Tun and S. Tojo: "Inheritance of Multiple Identity Conditions in Order-Sorted Logic", In Proceedings of 17th Australian Joint International Conference on Artificial Intelligence (AI-2004), Cairns, Australia, Springer-Verlag LNAI 3339: Advances in Artificial Intelligence, pp.1187-1193 (2004).

[8] N. N. Tun and S. Tojo: "Consistency Maintenance for Ontological Knowledge Updating", In proceedings of IEEE International Conference on information Reuse and Integration (IRI-2004), Las Vegas, USA, IEEE Cat.no.04EX974, pp.388-393 (2004).

Appendix A

Sortal Meta-class Ontology: `sort.owl`

```
⟨rdf:RDF xml:base="http://www.owl-ontologies.com/sort.owl"⟩
  ⟨owl:Ontology rdf:about=" "⟩
    ⟨owl:imports rdf:resource="http://protege.stanford.edu/plugins/owl/protege"/⟩
  ⟨/owl:Ontology⟩
  ⟨owl:Class rdf:ID="PhaseSort"⟩
    ⟨rdfs:subClassOf⟩
      ⟨owl:Restriction⟩
        ⟨owl:onProperty⟩
          ⟨owl:DatatypeProperty rdf:ID="nameOfCC"/⟩
        ⟨/owl:onProperty⟩
        ⟨owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema
          ♯int"⟩1 ⟨/owl:minCardinality⟩
      ⟨/owl:Restriction⟩
    ⟨/rdfs:subClassOf⟩
    ⟨rdfs:subClassOf⟩
      ⟨owl:Restriction⟩
        ⟨owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema
          ♯string"⟩phase ⟨/owl:hasValue⟩
        ⟨owl:onProperty⟩
          ⟨owl:DatatypeProperty rdf:ID="sort-category"/⟩
        ⟨/owl:onProperty⟩
      ⟨/owl:Restriction⟩
    ⟨/rdfs:subClassOf⟩
    ⟨rdfs:subClassOf⟩ ⟨owl:Class rdf:ID="Sort"/⟩ ⟨/rdfs:subClassOf⟩
    ⟨owl:disjointWith⟩ ⟨owl:Class rdf:ID="RoleSort"/⟩ ⟨/owl:disjointWith⟩
    ⟨owl:disjointWith⟩ ⟨owl:Class rdf:ID="Quasi-typeSort"/⟩ ⟨/owl:disjointWith⟩
    ⟨owl:disjointWith⟩ ⟨owl:Class rdf:ID="TypeSort"/⟩ ⟨/owl:disjointWith⟩
  ⟨/owl:Class⟩
  ⟨owl:Class rdf:about="♯TypeSort"⟩
    ⟨rdfs:subClassOf⟩
      ⟨owl:Restriction⟩
        ⟨owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema
```

```
        ♯int”⟩1 ⟨/owl:minCardinality⟩
      ⟨owl:onProperty⟩
        ⟨owl:DatatypeProperty rdf:ID=“nameOfOwnIC”/⟩
      ⟨/owl:onProperty⟩
    ⟨/owl:Restriction⟩
  ⟨/rdfs:subClassOf⟩
    ⟨owl:disjointWith rdf:resource=“♯PhaseSort”/⟩
    ⟨owl:disjointWith⟩ ⟨owl:Class rdf:about=“♯Quasi-typeSort”/⟩
⟨/owl:disjointWith⟩
  ⟨rdfs:subClassOf⟩
    ⟨owl:Restriction⟩
      ⟨owl:hasValue rdf:datatype=“http://www.w3.org/2001/XMLSchema
        ♯string”⟩type ⟨/owl:hasValue⟩
        ⟨owl:onProperty⟩
          ⟨owl:DatatypeProperty rdf:about=“♯sort-category”/⟩
        ⟨/owl:onProperty⟩
    ⟨/owl:Restriction⟩
  ⟨/rdfs:subClassOf⟩
  ⟨rdfs:subClassOf⟩
    ⟨owl:Class rdf:about=“♯Sort”/⟩
  ⟨/rdfs:subClassOf⟩
    ⟨owl:disjointWith⟩
      ⟨owl:Class rdf:about=“♯RoleSort”/⟩
    ⟨/owl:disjointWith⟩
  ⟨protege:SLOT-CONSTRAINTS⟩
    ⟨protege:PAL-CONSTRAINT rdf:ID=“PAL-CONSTRAINT_3”⟩
      ⟨protege:PAL-NAME rdf:datatype=“http://www.w3.org/2001/XMLSchema
        ♯string”⟩notPhaseToType⟨/protege:PAL-NAME⟩
          ⟨protege:PAL-STATEMENT rdf:datatype=“http://www.w3.org/2001/
            XMLSchema♯string”⟩
            (forall ?sub
              (forall ?super
                (=⟩ (and (’sort-category’ ?super ”phase”
                  (subclass-of ?sub ?super)
                  (own-slot-not-null ’sort-category’ ?sub))
                  (not (’sort-category’ ?sub ”type”)))))
      ⟨/protege:PAL-STATEMENT⟩
    ⟨/protege:PAL-CONSTRAINT⟩
  ⟨/protege:SLOT-CONSTRAINTS⟩
⟨/owl:Class⟩
⟨owl:Class rdf:about=“♯RoleSort”⟩
  ⟨owl:disjointWith rdf:resource=“♯TypeSort”/⟩
  ⟨owl:disjointWith⟩
    ⟨owl:Class rdf:about=“♯Quasi-typeSort”/⟩
  ⟨/owl:disjointWith⟩
    ⟨rdfs:subClassOf⟩
      ⟨owl:Restriction⟩
```

```
        ⟨owl:onProperty⟩
          ⟨owl:DatatypeProperty rdf:ID="nameOfEDR"/⟩
        ⟨/owl:onProperty⟩
        ⟨owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema
          ♯int"⟩1 ⟨/owl:minCardinality⟩
      ⟨/owl:Restriction⟩
    ⟨/rdfs:subClassOf⟩
    ⟨owl:disjointWith rdf:resource="♯PhaseSort"/⟩
      ⟨rdfs:subClassOf⟩
        ⟨owl:Class rdf:about="♯Sort"/⟩
      ⟨/rdfs:subClassOf⟩
      ⟨rdfs:subClassOf⟩
        ⟨owl:Restriction⟩
          ⟨owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema
            ♯string"⟩role ⟨/owl:hasValue⟩
              ⟨owl:onProperty⟩
                ⟨owl:DatatypeProperty rdf:about="♯sort-category"/⟩
              ⟨/owl:onProperty⟩
            ⟨/owl:Restriction⟩
          ⟨/rdfs:subClassOf⟩
        ⟨/owl:Class⟩
        ⟨owl:Class rdf:about="♯Sort"⟩
        ⟨rdfs:subClassOf⟩
                ⟨owl:Restriction⟩
              ⟨owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema
              ♯string"⟩type, quasi-type, role, phase ⟨/owl:hasValue⟩
                  ⟨owl:onProperty⟩
                        ⟨owl:DatatypeProperty rdf:about="♯sort-category"/⟩
                ⟨/owl:onProperty⟩
                ⟨/owl:Restriction⟩
          ⟨/rdfs:subClassOf⟩
        ⟨rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/
          owl♯Class"/⟩
        ⟨/owl:Class⟩
⟨owl:Class rdf:about="♯Quasi-typeSort"⟩
  ⟨rdfs:subClassOf rdf:resource="♯Sort"/⟩
  ⟨owl:disjointWith rdf:resource="♯TypeSort"/⟩
  ⟨owl:disjointWith rdf:resource="♯PhaseSort"/⟩
  ⟨rdfs:subClassOf⟩
    ⟨owl:Restriction⟩
    ⟨owl:onProperty⟩
      ⟨owl:DatatypeProperty rdf:about="♯sort-category"/⟩
    ⟨/owl:onProperty⟩
    ⟨owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema
      ♯string"⟩quasi-type ⟨/owl:hasValue⟩
  ⟨/owl:Restriction⟩
⟨/rdfs:subClassOf⟩
```

```
⟨rdfs:subClassOf⟩
  ⟨owl:Restriction⟩
    ⟨owl:minCardinality rdf:datatype="http://www.w3.org/2001/
      XMLSchema♯int"⟩1  ⟨/owl:minCardinality⟩
    ⟨owl:onProperty⟩
      ⟨owl:DatatypeProperty rdf:ID="nameOfCVA"/⟩
    ⟨/owl:onProperty⟩
  ⟨/owl:Restriction⟩
⟨/rdfs:subClassOf⟩
⟨protege:SLOT-CONSTRAINTS⟩
  ⟨protege:PAL-CONSTRAINT rdf:ID="PAL-CONSTRAINT_2"⟩
    ⟨protege:PAL-STATEMENT rdf:datatype="http://www.w3.org/2001/
      XMLSchema
    ♯string"⟩
        (forall ?sub
          (forall ?super
            (=⟩ (and ('sort-category' ?super "role"
              (subclass-of ?sub ?super)
                (own-slot-not-null 'sort-category' ?sub))
                  (not ('sort-category' ?sub "quasi-type")))))
      ⟨/protege:PAL-STATEMENT⟩
    ⟨protege:PAL-NAME rdf:datatype="http://www.w3.org/2001/XMLSchema
      ♯string"⟩notRoleToQuasi-type⟨/protege:PAL-NAME⟩
  ⟨/protege:PAL-CONSTRAINT⟩
⟨/protege:SLOT-CONSTRAINTS⟩
⟨owl:disjointWith rdf:resource="♯RoleSort"/⟩⟨/owl:Class⟩
  ⟨owl:DatatypeProperty rdf:about="♯nameOfCC"⟩
    ⟨rdfs:domain rdf:resource="♯PhaseSort"/⟩
    ⟨rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
      ♯string"/⟩
    ⟨/owl:DatatypeProperty⟩
  ⟨owl:DatatypeProperty rdf:about="♯sort-category"⟩
    ⟨rdfs:range⟩
    ⟨owl:DataRange⟩
      ⟨owl:oneOf rdf:parseType="Resource"⟩
      ⟨rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema
      ♯string"⟩type⟨/rdf:first⟩
      ⟨rdf:rest rdf:parseType="Resource"⟩
        ⟨rdf:rest rdf:parseType="Resource"⟩
          ⟨rdf:rest rdf:parseType="Resource"⟩
            ⟨rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema
              ♯string"⟩phase⟨/rdf:first⟩
            ⟨rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns
              ♯nil"/⟩
          ⟨/rdf:rest⟩
          ⟨rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema
        ♯string"⟩role⟨/rdf:first⟩
      ⟨/rdf:rest⟩
```

```
      ⟨rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema
        ♯string"⟩quasi-type⟨/rdf:first⟩
      ⟨/rdf:rest⟩
    ⟨/owl:oneOf⟩
  ⟨/owl:DataRange⟩
⟨/rdfs:range⟩
  ⟨rdfs:domain rdf:resource="♯Sort"/⟩
  ⟨rdf:type rdf:resource="http://www.w3.org/2002/07/
    owl♯FunctionalProperty"/⟩
  ⟨/owl:DatatypeProperty⟩
    ⟨owl:DatatypeProperty rdf:about="♯nameOfOwnIC"⟩
    ⟨rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
      ♯string"/⟩
    ⟨rdfs:domain rdf:resource="♯TypeSort"/⟩
  ⟨/owl:DatatypeProperty⟩
  ⟨owl:DatatypeProperty rdf:about="♯nameOfEDR"⟩
    ⟨rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
      ♯string"/⟩
    ⟨rdfs:domain rdf:resource="♯RoleSort"/⟩
  ⟨/owl:DatatypeProperty⟩
  ⟨owl:DatatypeProperty rdf:about="♯nameOfCVA"⟩
    ⟨rdfs:domain rdf:resource="♯Quasi-typeSort"/⟩
    ⟨rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
      ♯string"/⟩
  ⟨/owl:DatatypeProperty⟩
⟨protege:PAL-CONSTRAINT rdf:ID="PAL-CONSTRAINT_5"⟩
  ⟨protege:PAL-NAME rdf:datatype="http://www.w3.org/2001/XMLSchema
    ♯string"⟩notQuasi-typeToType⟨/protege:PAL-NAME⟩
    ⟨protege:PAL-STATEMENT rdf:datatype="http://www.w3.org/2001/
      XMLSchema
    ♯string"⟩
    (forall ?sub
      (forall ?super
        (=⟩ (and ('sort-category' ?super "quasi-type"
          (subclass-of ?sub ?super)
            (own-slot-not-null 'sort-category' ?sub))
            (not ('sort-category' ?sub "type")))))
  ⟨/protege:PAL-STATEMENT⟩
⟨/protege:PAL-CONSTRAINT⟩
⟨protege:PAL-CONSTRAINT rdf:ID="PAL-CONSTRAINT_4"⟩
  ⟨protege:PAL-NAME rdf:datatype="http://www.w3.org/2001/
    XMLSchema♯string"⟩notPhaseToQuasi-type⟨/protege:PAL-NAME⟩
    ⟨protege:PAL-STATEMENT rdf:datatype="http://www.w3.org/2001/
      XMLSchema♯string"⟩
      (forall ?sub
        (forall ?super
          (=⟩ (and ('sort-category' ?super "phase"
```

```
                (subclass-of ?sub ?super)
                  (own-slot-not-null 'sort-category' ?sub))
                    (not ('sort-category' ?sub "quasi-type")))))
      ⟨/protege:PAL-STATEMENT⟩
    ⟨/protege:PAL-CONSTRAINT⟩
    ⟨protege:PAL-CONSTRAINT rdf:ID="PAL-CONSTRAINT_1"⟩
    ⟨protege:PAL-STATEMENT rdf:datatype="http://www.w3.org/2001/
      XMLSchema
        ♯string"⟩
        (forall ?sub
          (forall ?super
            (=⟩ (and ('sort-category' ?super "role"
              (subclass-of ?sub ?super)
                (own-slot-not-null 'sort-category" ?sub))
                (not ('sort-category' ?sub "type")))))
      ⟨/protege:PAL-STATEMENT⟩
      ⟨protege:PAL-NAME rdf:datatype="http://www.w3.org/2001/XMLSchema
        ♯string"⟩notRoleToType⟨/protege:PAL-NAME⟩
    ⟨/protege:PAL-CONSTRAINT⟩
⟨/rdf:RDF⟩
```