ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ONTONL: ΜΙΑ ΓΕΝΝΗΤΡΙΑ ΔΙΕΠΑΦΩΝ ΦΥΣΙΚΗΣ ΓΛΩΣΣΑΣ ΣΕ ΒΑΣΕΙΣ ΓΝΩΣΗΣ

Υπό

Αναστασία Καραναστάση

Διατριβή που υπεβλήθη για την μερική ικανοποίηση των απαιτήσεων για την απόκτηση Μεταπτυχιακού Διπλώματος Ειδίκευσης

στους Ηλεκτρονικούς Μηχανικούς και Μηχανικούς Υπολογιστών

Εργαστήριο Διανεμημένων Πληροφοριακών Συστημάτων και Εφαρμογών Πολυμέσων

ΧΑΝΙΑ 2007

Περίληψη

Οι διεπαφές φυσικής γλώσσας (Natural Language Interfaces) είναι πολύ αποτελεσματικές διεπαφές για απλούς χρήστες, για σύνθετα δομημένη πληροφορία, χρήστες με αναπηρία, κτλ. Είναι εντούτοις πολύ ακριβό να χτιστούν οι διεπαφές φυσικής γλώσσας σε βάσεις γνώσεων. Επιπλέον, οι ασάφειες στη φυσική γλώσσα μπορούν να δημιουργήσουν την ανάγκη για διαλόγους διευκρίνισης που καταστρέφουν την αποτελεσματικότητα της επικοινωνίας.

Σε αυτήν την διατριβή στοχεύουμε και τα δύο προβλήματα. Παρουσιάζουμε ένα πλαίσιο τεχνολογίας λογισμικού, το πλαίσιο OntoNL που προσφέρει επαναχρησιμοποιήσιμο κώδικα για την παραγωγή των διεπαφών φυσικής γλώσσας σε βάσεις γνώσης. Αναπτύσσουμε επίσης τις μεθοδολογίες μέσα σε αυτό το πλαίσιο για την ελαχιστοποίηση των σημασιολογικών ασαφειών της φυσικής γλώσσας και ταξινομούμε τα αποτελέσματα βασιζόμενοι σε ένα σημασιολογικό μέτρο συγγένειας που αναπτύξαμε, έτσι ώστε ο χρήστης να βλέπει μέσα στις πρώτες απαντήσεις τα επιθυμητά αποτελέσματα.

Το πλαίσιο OntoNL υλοποιεί μια πλατφόρμα λογισμικού που αυτοματοποιεί σε έναν μεγάλο βαθμό την κατασκευή των διεπαφών φυσικής γλώσσας για τις βάσεις γνώσης. Για να επιτύχει τη δυνατότητα εφαρμογής και επαναχρησιμοποίησης του πλαισίου OntoNL σε πολλές διαφορετικές εφαρμογές και περιοχές, το λογισμικό είναι ανεξάρτητο στις οντολογίες περιοχών.

Τα τμήματα λογισμικού του πλαισίου OntoNL εξετάζουν και αντιμετωπίζουν ενιαία μια σειρά προβλημάτων στην ανάλυση της πρότασης, κάθε ένα από τα οποία παραδοσιακά απαιτούσε έναν χωριστό μηχανισμό. Μια ενιαία αρχιτεκτονική χειρίζεται τη συντακτική και τη σημασιολογική ανάλυση, χειρίζεται τις ασάφειες και στο γενικό και εξαρτώμενο από το πεδίο περιβάλλον. Συγχρόνως, το πλαίσιο έχει σχεδιαστεί με τέτοιο τρόπο ώστε να αποφευχθούν οι εξαρτήσεις από την βάση πληροφοριών. Με τον τρόπο αυτό γίνεται επαναχρησιμοποιήσιμο στις διαφορετικές εφαρμογές με τη διαφορετική σημασιολογία περιοχών.

Παρουσιάζουμε επίσης μια εφαρμογή του πλαισίου OntoNL που δημιουργεί μια διεπαφή φυσικής γλώσσας σε μια σημασιολογική βάση πολυμέσων με ψηφιακό οπτικοακουστικό περιεχόμενο γεγονότων και μεταδεδομένων ποδοσφαίρου, που αναπτύχθηκε και

καταδείχτηκε στο 2ο και 3ο Annual Review του DELOS II EU Network of Excellence (IST 507618) (http://www.delos.info/).

Η έρευνα που πραγματοποιήθηκε στην παρούσα διατριβή υποστηρίχθηκε από το DELOS II, Network of Excellence on Digital Libraries NoE-G038-507618 / IST-507618.

ONTONL: AN ONTOLOGY-BASED NATURAL LANGUAGE INTERFACE GENERATOR FOR KNOWLEDGE REPOSITORIES

by

Anastasia Karanastasi

A thesis submitted in fulfilment of the

requirements for the degree of

Master of Science in Electronic and Computer Engineering

TECHNICAL UNIVERSITY OF CRETE

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

Laboratory of Distributed Multimedia

Information Systems and Applications

MUSIC

CHANIA 2007

DEDICATION

To Nikos.

The man who taught me that

The greatest thing you'll ever learn is just to Love, and be Loved in return.

Eden Ahbez

"Nature Boy" (1948)

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis.

I authorize the Technical University of Crete to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Technical University of Crete to reproduce the thesis by photocopying of by other means, in total or part, at the request of other institutions or individuals for the purpose of scholarly research.

ACKNOWLEDGMENTS

My thesis work would not have been possible without the help of my advisor, other collaborators, and fellow students. I am especially fortunate to have been advised by Prof. Stavros Christodoulakis. Firstly, I am grateful to him for teaching me almost everything I know about doing research and being part of the academic community. Secondly, I deeply appreciate his constant support and advice on many levels. The work in this thesis was profoundly shaped by numerous insightful discussions with him.

I would also like to thank Prof. Michael Lagoudakis and Prof. Alexandros Potamianos for being on my oral exam committee, for the time they devoted and their critical evaluation and comments on my thesis.

I would also like to thank my colleagues Chrisa Tsinaraki and Fotis Kazasis for their support and useful comments on my work and especially Nektarios Gioldasis. Many thanks to Alexandros Zotos for his contribution to the work described in chapter 6 of this thesis.

There have been a number of people who made the grad school process much, much easier to deal with: my best friends Maria Frantzi, George Kotopoulos and Aristotelis Kotopoulos, and my friends and office mates Themistoklis Dakanalis, Nikos Giannopoulos and Manolis Mylonakis.

Finally, and most importantly, deepest thanks to my mom and dad; they always believed, even when I didn't.

PUBLICATIONS

Part of the work that is included in this report has been published in the following Conference Proceedings:

- A. Karanastasi, S. Christodoulakis: "The OntoNL Semantic Relatedness Measure for OWL Ontologies", in the Proceedings of the Second IEEE International Conference on Digital Information Management (IEEE ICDIM '07), 28-31 October 2007, Lyon, France
- A. Karanastasi, S. Christodoulakis: "Semantic Processing of Natural Language Queries in the OntoNL Framework", in the Proceedings of the IEEE International Conference on Semantic Computing (IEEE ICSC), 17-19 September 2007, Irvine, CA
- A. Karanastasi, S. Christodoulakis: "Ontology-Driven Semantic Ranking for Natural Language Disambiguation in the OntoNL Framework", in the Proceedings of the 4th European Semantic Web Conference (ESWC), 3-7 June 2007, Innsbruck, Austria
- A. Karanastasi, A. Zwtos, S. Christodoulakis: "The OntoNL Framework for Natural Language Interface Generation and a Domain-specific Application", in Proceedings of the DELOS Conference on Digital Libraries, 13-14 February, Tirrenia, Pisa, Italy 2007
- A. Karanastasi, A. Zwtos, S. Christodoulakis: "User Interactions with Multimedia Repositories using Natural Language Interfaces: an Architectural Framework and its Implementation", in the Journal of Digital Information Management (JDIM), Volume 4 Issue 4, December 2006
- A. Karanastasi, A. Zwtos, S. Christodoulakis: "User Interactions with Multimedia Repositories using Natural Language Interfaces: an Architectural Framework and its Implementation", in Proceedings of the Fourth Special Workshop on Multimedia Semantics (WMS06), June 19-21, 2006
- S. Christodoulakis, A. Karanastasi, J. Koehler, K. Biatov, T. Catarci, S. Kimani: "Natural Language and Speech Interfaces to Knowledge Repositories", Poster on the 9th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2005), September 2005, Vienna, Austria

- A. Karanastasi, S. Christodoulakis: “OntoNL: An Ontology-based Natural Language Interface Generator for Multimedia Repositories”, in Proceedings of the Seventh International Workshop of the EU Network of Excellence DELOS on AUDIO-VISUAL CONTENT AND INFORMATION VISUALIZATION IN DIGITAL LIBRARIES (AVIVDiLib'05), May 2005
- A. Karanastasi, F. Kazasis, S. Christodoulakis: “A Natural Language Model and a System for Managing TV-Anytime Information in Mobile Environments”, ACM/Verlag Personal and Ubiquitous Computing Journal, Volume 4, 2005
- A. Karanastasi, F. Kazasis, S. Christodoulakis: "A Natural Language Model for Managing TV-Anytime Information in Mobile Environments", In Proceedings of the International Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS), Riga, Latvia, 7 - 8 June, 2004
- A. Karanastasi, F. Kazasis, S. Christodoulakis: "A Natural Language Model and a System for Managing TV-Anytime Information from Mobile Devices", In Proceedings of the 9th International Conference on Applications of Natural Language to Information Systems (NLDB), Manchester, United Kingdom, June 2004
- F.G. Kazasis, N. Moumoutzis, N. Pappas, A. Karanastasi, S.Christodoulakis: "Designing Ubiquitous Personalized TV-Anytime Services", In the proceedings of the International Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS), Klagenfurt/Velden, Austria, June 2003

Table of Contents

List of Tables

List of Figures

ABSTRACT

ONTONL: AN ONTOLOGY-BASED NATURAL LANGUAGE INTERFACE GENERATOR FOR KNOWLEDGE REPOSITORIES

Anastasia Karanastasi
Supervisor: Professor Stavros Christodoulakis

Natural Language Interfaces are very effective interfaces for naïve users, complex structured information, users with disabilities, etc. It is however very expensive to build Natural Language Interfaces to knowledge bases. In addition, ambiguities in natural language may create the need for clarification dialogues that destroy the effectiveness of communication.

In this Thesis we target both problems. We present a Software Engineering Framework, the OntoNL Framework, that offers reusable code for the generation of Natural Language Interfaces to Knowledge Repositories. We also develop methodologies within this framework for reducing the semantic ambiguities of the natural language and we rank the results based on a semantic relatedness measure that we developed so that the user sees within the first few answers the desired results.

The OntoNL Framework implements a software platform that automates to a large degree the construction of natural language interfaces for knowledge repositories. To achieve the applicability and reusability of the OntoNL Framework in many different applications and domains, the supporting software is independent on the domain ontologies.

The software components of the OntoNL Framework address uniformly a range of problems in sentence analysis each of which traditionally had required a separate mechanism. A single architecture handles both syntactic and semantic analysis, handles ambiguities at both the general and the domain specific environment. At the same time, the Framework has been designed in a way to avoid dependencies with the information repository so that it becomes reusable in different applications with different domain semantics.

We also present an application of the OntoNL Framework that creates a natural language interface to a semantic multimedia repository with digital audiovisual content of soccer events and metadata concerning soccer in general, that has been developed and demonstrated in the 2nd and 3rd Annual Review of the DELOS II EU Network of Excellence (IST 507618) (http://www.delos.info/).

This research that was carried out in this thesis was supported by the DELOS II, Network of Excellence on Digital Libraries NoE-G038-507618 / IST-507618.

Chapter 1

Introduction

The goal of the Natural Language Processing (NLP) is to design and build software that will analyze and understand languages that humans use naturally, so that eventually a user will be able to address his/her computer as though he/she were addressing another person.

This goal is not easy to reach. "Understanding" language means, among other things, knowing what concepts a word or phrase stands for and knowing how to link those concepts together in a meaningful way. It's ironic that natural language, the symbol system that is easiest for humans to learn and use, is hardest for a computer to master. Long after machines have proven capable of inverting large matrices with speed and grace, they still fail to master the basics of our spoken and written languages.

The challenges we face stem from the highly ambiguous nature of natural language. As an English speaker a person effortlessly understands a sentence like "Flying planes can be dangerous". Yet this sentence presents difficulties to a software program that lacks both human knowledge of the world and the user experience with linguistic structures. Is the more plausible interpretation that the pilot is at risk, or that the danger is to people on the ground? Should "can" be analyzed as a verb or as a noun? Which of the many possible meanings of "plane" is relevant? Depending on context, "plane" could refer to, among other things, an airplane, a geometric object, or a woodworking tool. How much and what sort of context needs to be brought to bear on these questions in order to adequately disambiguate the sentence?

In particular, the advantages of natural language interfaces for user interaction with information repositories when using modern interaction devices, like mobile phones, PDA’s etc., when the information is complex and when access for all is desired, are well understood.

Chapter 1

Among the very best-performing and most robust language processing systems have been knowledge-based natural language systems — NLP systems that understand an input text – sentence, by relying heavily on handcrafted knowledge about the domain and about the world in general. Not surprisingly, however, generating this background knowledge for new domains is time consuming, difficult and error prone, and requires the expertise of computational linguists familiar with the underlying NLP system. This is an example of the knowledge engineering bottleneck for natural language processing systems. It is one of the biggest problems in designing and building natural language systems and promises only to become worse as natural language systems attempt to understand a wider variety of natural language expressions, to produce more complex language models, and to derive knowledge structures directly from these expressions.

Until recently, the natural language interfaces (NLIs) between humans and machines were either specific to a particular application with limited expectations or linguistic-based with possibly many ambiguities that lead to lengthy disambiguation dialogues. An attempt of using a more generalized approach of the construction of an NLI, particularly in the domain of digital TV was presented by Karanastasi et. all [2003, 2004] with good results when dealing with ambiguities, without the need of using clarification dialogues for the disambiguation, because of the well-structured domain of Digital TV, the TV-Anytime standard [TV-Anytime Forum] and the TV-Anytime User Profile information. The differentiation from other NLIs was in the structures of the language model that were easily reusable and extendable. In that way the grammatical rules could be applied in a language model formed for a specific domain. Another improvement was in the algorithm for resolving ambiguities that was based on the schema of the repository and on User Profile information.

A limitation of a system like this is that using a domain grammar, with specific domain grammar rules the system searches for specific information in a repository, so the NLI is not very easily reusable. This specific information can have relative information that can disorient the user if the grammar rules that fulfill a certain utterance do not specify the correct context of interest. Also, the grammar rules are defined by the syntax of the

repository they refer to, a fact that can be limiting in searching in more than one ontologies of the same domain. The implementation framework that is presented here has as its domain of use the English language and not particular information concerning a task of interest. We are targeting to the creation of a framework, since it provides a very high degree of reuse, much more so than individual classes and it is easily extendable, which is significant in the automatic construction of natural language interfaces domain independent.

By providing a software engineering framework we face the problem of the prohibited cost of constructing natural language interfaces for particular applications and domain. The framework is an extendable subsystem for a set of related services. It is a cohesive set of abstract classes that define a natural language interface to conform to and object interactions to participate in. Using specific limited-sized lexicon that has a strong dependency on a specific domain, leads to loosing the context of each word a user uses, because of the relative clarification from the grammar and loosing the synonyms that can be very helpful in the disambiguation phase.

Ambiguity Resolution in Language Understanding

Ascertaining what is intended in a sentence when more than one interpretation is possible has always been a central issue in natural language processing: ambiguity resolution is required whenever the system must choose among two or more distinct representations of the input. Ambiguity pervades virtually all aspects of language analysis, and sentence analysis in particular exhibits a large number of syntactic, semantic, and pragmatic ambiguities that demand adequate resolution before the sentence can be understood. TheOntoNL framework is designed to acquire solutions to all lexical and structural ambiguity problems encountered during sentence analysis.

Part-of-Speech Ambiguity

Knowing the part of speech of a word (noun, verb, preposition, etc.) in a particular context, for example, often supplies important hints for determining the word's function in the sentence. Consider the word "chairs" in the sentences below:

1. Professor James Cameron chairs the workshop held by the Technical University of London.

2. IKEA chairs were used for the workshop held by the Technical University of London.

Despite nearly identical local contexts, "chairs" is a verb in sentence 1, but a noun in sentence 2 and making this distinction is crucial to determining meaning of each sentence.

Word Sense Ambiguity

Even if a word's part of speech is known, the intended meaning of the word in a particular context often requires disambiguation. The word "player", for example, is a noun in both sentences below. In each sentence, however, the word takes on a very different meaning:

1. He was the best player of Barcelona ever.

2. He was a major player in setting up the corporation.

In sentence 1, "player" refers to a person that participates or is skilled at some game while in the second sentence it is used metaphorically to mean "an important participant". Even when it seems as if a word is unquestionably unambiguous, it can be used in contexts that confer a novel meaning. In general, one would probably say that the word "Milan" is unambiguous, for example. It refers to a city in northern Italy. Consider the following sentence, however:

Milan will be looking to continue their positive run of results at Chievo when they take the pitch at the Bentegodi stadium on Saturday.

In this sentence Milan refers to a football team that participates in the Champions League.

Chapter 1

Noun Compound Analysis

Noun compounds are a frequently encountered construction in natural language processing (NLP), consisting of a sequence of two or more nouns which together function syntactically as a noun. In English, compounds consisting of two nouns are predominantly right-headed [Piera, 1995]. However, compound construction is recursive and both the modifier and the head can themselves be compounds, resulting in structural ambiguities.

Consider the following pair of noun compounds:

1. [player [shirt number]]
2. [[soccer team] shirt]

Both compounds consist of the same parts-of-speech, yet the structures differ: (1) is right-branching, while (2) is left-branching. In linguistics, branching is the general tendency towards a given order of words within sentences and smaller grammatical units within sentences (such as subordinate propositions, prepositional phrases, etc.).

Dealing with Unknown Words

The problem of ambiguity in sentence analysis is also clearly pronounced in the case of unknown words. When encountering a word for which it has no definition, a robust NLP system makes a series of decisions that together shape the meaning of the word as it functions in the current context. Each decision is essentially a separate, but related, ambiguity resolution task. What is the word's part of speech in the current context? What is its meaning? How is it related to other items in the sentence or paragraph or text? Is the word of special importance with respect to the goals of the answering mechanism?

A related problem for natural language processing systems is to know when the system's knowledge of a word is incomplete. Assume, for example, that an NLP system had a definition for the word "market" that was syntactically and semantically compatible with its use in sentence 1 below, but then encountered "market" in sentence 2:

1. The Asian beef market generally is starting to open up to exporters.
2. Two companies plan to market a new chip with ceramic circuits.

A robust system should (1) note that its current definition is inadequate, (2) infer the appropriate syntactic and semantic features of "market," (3) incorporate the new definition into the system's lexicon, and (4) determine how the system will distinguish the two uses of "market" in the future.

Natural Language and Context

"You shall know a word by the company it keeps." This remark of J. R. Firth, famed British linguist, seems as an apt reminder for the ubiquity of context. According to Crystal [Crystal, 1991], 'context' is a general term in linguistics and phonetics to refer to specific parts of an utterance (or text) near or adjacent to a unit (e.g., a sound, word) which is the focus of attention. Blackburn [Blackburn, 1994] offers a similar definition: "In linguistics, context is the parts of an utterance surrounding a unit and which may affect both its meaning and its grammatical contribution." However, he is quick to add that context also refers to "the wider situation, either of the speaker or of the surroundings that may play a part in determining the significance of a saying." It is standard nowadays to use the term 'co-text' for the narrow, purely linguistic context [Lyons, 1995]. As for the total nonlinguistic background to an utterance (including: the immediate situation in which it is used, the knowledge of speaker and hearer about the commonsense world, the knowledge of what has been said earlier, the relevant beliefs and presuppositions of speaker and hearer), the term 'situational context' has been offered. Similarly, Lyons [Lyons, 1995] uses the term 'context of situation' as a synonym for situational context. He believes that natural language meaning must be studied as a multiple phenomenon, its numerous aspects being relatable to (i) different levels of linguistic analysis, and (ii) features of the world.

While approaches to natural language generation that ignore context are straightforward to implement, they often produce unsatisfactory linguistic output. In generating noun phrases for example, a system that ignores context must make a global choice about which

properties and relations to include in its output. Such a system may awkwardly evoke properties or relations in contexts in which the referent would be perfectly clear without them or it may produce ambiguous output in contexts where some categorically excluded property or relation would disambiguate the intended referent.

McCarthy offers no definition of context. His underlying assumption is that "[t]here are mathematical context structures of different properties, some of which are useful" [McCarthy, 1996]. He wittily remarks that asking what a context is like asking what a group element is.

All these remarks conclude to a need for a more systematic way for dealing with semantic ambiguities that concern natural language and the solution can be found by combining structural and syntactic information and domain specific information that specialize the context of the natural language expression.

The Motivation

A Common Framework for Ambiguity Resolution in Question-Answering Systems to Knowledge Repositories

The OntoNL framework for natural language processing systems is motivated by the following observations:

1. There is a need to organize the various methods that are used for syntactic and semantic analysis in a natural language interface with good results without the dependence of particular grammatical rules and domain-dependent lexicons. The framework describes a method for designing an information system in terms of a set of building blocks, and for showing how the building blocks fit together. It includes a list of recommended standards and compliant products that can be used to implement the building blocks.
2. Knowing the context in which an ambiguity occurs is crucial for resolving it. Sentence analysis in general and sense understanding in particular , requires a series of context-sensitive mappings from one representation into another – the

system often maps the words of a sentence into parts of speech, the part-of-speech sequences into low level constituents and the low-level constituents into predicate-argument relations enhanced afterwards with context information (sense in a particular domain), for example. This results to context-sensitive solutions by the framework.

3. By developing a Natural Language Interface Generator, we provide system developers the most natural way of communication with their users. . Every system that uses knowledge for a specific domain based on an OWL ontology can use the OntoNL Framework for NLI generation.

The Use of Domain Ontologies for a Domain-Specific Disambiguation

Due to the complexity of natural language, reliable word sense disambiguation is an unaccomplished goal in spite of years of work in fields like Artificial Intelligence, Computational Linguistics and other. The goal could be approached by applying methods for consulting knowledge sources such as domain ontologies. Ontologies are usually expressed in a logic-based language, so that detailed, accurate, consistent, sound, and meaningful distinctions can be made among the classes (general concepts), properties (those concepts may have), and the relations that can exist among these concepts. A module dealing with ontologies can perform automated reasoning using the ontologies, and thus provide advanced services to intelligent applications such as: conceptual/semantic search and retrieval, software agents, decision support, speech and natural language understanding and knowledge management.

We examine how consulting ontologies can help to do semantic language processing and disambiguation, not just syntactic.

Contributions

The goal of this work is to address the knowledge engineering bottleneck for natural language processing systems. To this end, it will present the OntoNL natural language

interface generator for interactions with knowledge repositories and make the following claims:

1. The OntoNL framework is able to address uniformly a range of problems in sentence analysis each of which traditionally had required a separate computational mechanism. In particular a single architecture:
    a. handles both syntactic and semantic ambiguities
    b. handles ambiguity at both a general and a domain specific environment
    c. uses semantic relatedness measures for the concepts of the ontology to provide better ranked results
2. The OntoNL framework makes use of OWL rich vocabulary by using upper and domain ontologies. The semantic search procedure proposed here is designed to satisfy different kinds and levels of ambiguity. The procedure uses information from the ontologies and by specific clusters of context inside an ontology. Given an OWL ontology, weights are assigned to links based on certain properties of the ontology, so that they measure the level of relatedness between concepts. In this way we can identify related concepts in the ontology that guide the semantic search procedure.
3. The OntoNL framework is reusable, domain independent and works with input only the OWL ontology that was used as a reference schema for constructing the repository.

To demonstrate support for these claims, we use OntoNL framework to create a Natural Language Interface for an MPEG-7 Semantic Repository with information concerning the domain of soccer. The NLI is used in a question answering system.

Report Structure

The rest of the report is organized as follows. Section 2 is about related research and work in the Natural Language Understanding area and in the application of the methods in Question Answering Systems. In Section 3 the OntoNL framework is presented and in Section 4 and 5 the analysis of the models for syntactic and semantic disambiguation of

Natural Language is presented. Section 6 is about the implementation of a Natural Language Interface for an MPEG-7 Multimedia Repository. Section 7 introduces the Evaluation Framework, measures and results and in Section 8 we discuss about the results and we conclude.

Chapter 2

Related Research and Work

Natural language processing (NLP) is a subfield of artificial intelligence and linguistics. It studies the problems of automated generation and understanding of natural human languages.

The major tasks in NLP

- Text to speech
- Speech recognition
- Natural language understanding
- Natural language generation
- Machine translation
- Question answering
- Information retrieval
- Information extraction
- Text-proofing
- Translation technology
- Automatic summarization

In the work presented in this thesis we focus in Natural Language Understanding and in Question Answering with information retrieval techniques. In this chapter we provide an overview of research in the Word Sense Disambiguation area and how semantics interfere with Natural Language Understanding. We also provide a short overview of ontologies that are going to be used in the thesis for the semantic disambiguation procedure and an overview of the WordNet that is used for the syntactic disambiguation of the natural

language expression. We then provide a short survey in Natural Language Interfaces to Databases and conclude with the state of the art in Question Answering Systems. We show what research has been carried out in the past, what research is currently being undertaken, and the shortcomings of current and past research. We conclude with an overview of the related technologies that were used for the research and development of the OntoNL system.

Natural Language Understanding

To *understand* something is to transform it from one representation into another, where this latter representation is chosen to correspond to a set of available actions that could be performed, and for which a mapping is designed so that for each event an *appropriate* action will be performed.

The steps in the process of natural language understanding are:

Morphological analysis [Ritchey, 1998]

Individual words are analyzed into their components, and non-word tokens (such as punctuation) are separated from the words. For example, in the phrase "Bill's house" the proper noun "Bill" is separated from the possessive suffix "'s."

Syntactic analysis [Allen, 1995]

The purpose of syntactic analysis is to determine the structure of the input text. This structure consists of a hierarchy of phrases, the smallest of which are the basic symbols and the largest of which is the sentence. It can be described by a tree with one node for each phrase. Basic symbols are represented by leaf nodes, and other phrases by interior nodes. The root of the tree represents the sentence. Syntactic processing interprets the difference between "John hit Mary" and "Mary hit John."

Semantic analysis [Allen, 1995]

The structures created by the syntactic analyzer are assigned meanings. In most universes, the sentence "Colourless green ideas sleep furiously" [Chomsky, 1957] would be rejected as *semantically anomalous*. This step must map individual words into appropriate objects in the knowledge base, and must create the correct structures to correspond to the way the meanings of the individual words combine with each other. Semantic processing determines the differences between such sentences as "The pig is in the pen" and "The ink is in the pen."

Discourse integration [Ledoux, et. al, 2006]

It is the process of capturing the contextual effects that individual sentences have on each other in determining their joint meaning. The meaning of an individual sentence may depend on the sentences that precede it and may influence the sentences yet to come. The entities involved in the sentence must either have been introduced explicitly or they must be related to entities that were. The overall discourse must be coherent.

Pragmatic analysis [Doyle, 1997]

The structure representing what was said is reinterpreted to determine what was actually meant. It is the process of using more general knowledge about the world/domain to modify the interpretation into it's true meaning

Word Sense Disambiguation

Word sense disambiguation is the task of assigning to each occurrence of an ambiguous word in a text one of its possible senses [Ide, N., Veronis, J.,1998]. It is a process that can use one to all the steps mentioned above. The task therefore necessarily involves two steps: (1) the determination of all the different senses for every word relevant (at least) to the text or discourse under consideration; and (2) a means to assign each occurrence of a word to the appropriate sense.

Chapter 2

The automatic disambiguation of word senses has been an interest and concern since the earliest days of computer treatment of language in the 1950's [Ide and Veronis, 1998]. Sense disambiguation is an "intermediate task" [Wilks and Stevenson, 1996] which is not an end in itself, but rather is necessary at one level or another to accomplish most natural language processing tasks. It is obviously essential for language understanding applications such as message understanding, man-machine communication, etc.; it is at least helpful, and in some instances required, for applications whose aim is not language understanding:

- machine translation: sense disambiguation is essential for the proper translation of words.
- information retrieval and hypertext navigation: when searching for specific keywords, it is desirable to eliminate occurrences in documents where the word or words are used in an inappropriate sense.
- content and thematic analysis: a common approach to content and thematic analysis is to analyze the distribution of pre-defined categories of words--i.e., words indicative of a given concept, idea, theme, etc.--across a text.
- grammatical analysis: sense disambiguation is useful for part of speech tagging.
- speech processing: sense disambiguation is required for correct phonetization of words in speech synthesis.
- text processing: sense disambiguation is necessary for spelling correction.

The problem of word sense disambiguation has been described as AI-complete, that is, a problem which can be solved only by first solving all the difficult problems in artificial intelligence (AI), such as the representation of common sense and encyclopedic knowledge. However, at about the same time considerable progress was being made in the area of knowledge representation, especially the emergence of semantic networks, which were immediately applied to sense disambiguation. In the past ten years, attempts to automatically disambiguate word senses have multiplied, due, like much other similar activity in the field of computational linguistics, to the availability of large amounts of machine readable text and the corresponding development of statistical methods to identify and apply information about regularities in this data. Now that other problems

amenable to these methods, such as part of speech disambiguation and alignment of parallel translations, have been fairly thoroughly addressed, the problem of word sense disambiguation has taken centre stage, and it is frequently cited as one of the most important problems in natural language processing research today. The following sections survey the approaches applied to date.

Early Word Sense Disambiguation (WSD) in Machine Translation

The first attempts at automated sense disambiguation were made in the context of machine translation (MT). In his famous Memorandum, Weaver [Weaver, 1949] discusses the need for WSD in machine translation and outlines the basis of an approach to WSD which underlies all subsequent work on the topic:

> *If one examines the words in a book, one at a time as through an opaque mask with a hole in it one word wide, then it is obviously impossible to determine, one at a time, the meaning of the words. [...] But if one lengthens the slit in the opaque mask, until one can see not only the central word in question but also say N words on either side, then if N is large enough one can unambiguously decide the meaning of the central word. [...]The practical question is: "What minimum value of N will, at least in a tolerable fraction of cases, lead to the correct choice of meaning for the central word?"*

A well-known early experiment by Kaplan [Kaplan, 1950] attempted to answer this question at least in part, by presenting ambiguous words in their original context and in a variant context providing one or two words on either side to seven translators. Kaplan observed that sense resolution given two words on either side of the word was not significantly better or worse than when given the entire sentence. The same phenomenon has been reported by several researchers since Kaplan's work appeared: [Masterman, 1961], [Koutsoudas and Korfhage, 1956], [Gougenheim and Michéa, 1961], [Choueka and Lusignan, 1985].

The striking fact about this early work on WSD is the degree to which the fundamental problems and approaches to the problem were foreseen and developed at that time. However, without large-scale resources most of these ideas remained untested and to large extent, forgotten until several decades later.

Chapter 2

AI methods

AI methods began to flourish in the early 1960's and began to attack the problem of language understanding. As a result, WSD in AI work was typically accomplished in the context of larger systems intended for full language understanding. In the spirit of the times, such systems were almost always grounded in some theory of human language understanding which they attempted to model and often involved the use of detailed knowledge about syntax and semantics to perform their task, which was exploited for WSD. The most common methods that follow this perspective are the symbolic methods and the connectionist methods. As far as it concerns the connectionist methods work in psycholinguistics in the 1960's and 70's established that semantic priming--a process in which the introduction of a certain concept will influence and facilitate the processing of subsequently introduced concepts that are semantically related--plays a role in disambiguation by humans. This idea is realized in spreading activation models [Collins and Loftus, 1975; Anderson, 1976, 1983], where concepts in a semantic network are activated upon use, and activation spreads to connected nodes. Activation is weakened as it spreads, but certain nodes may receive activation from several sources and be progressively reinforced.

Knowledge-based methods

The AI-based work of the 1970's and 80's was theoretically interesting but not at all practical for language understanding in any but extremely limited domains. A significant roadblock to generalizing WSD work was the difficulty and cost of hand-crafting the enormous amounts of knowledge required for WSD: the so-called "knowledge acquisition bottleneck" [Gale et al., 1993]. Work on WSD reached a turning point in the 1980's when large-scale lexical resources such as dictionaries, thesauri, and corpora became widely available. Efforts began to attempt to automatically extract knowledge from these sources and, more recently, to construct large-scale knowledge bases by hand.

Corpus-based methods

Since the end of the Nineteenth Century, the manual analysis of corpora has enabled the study of words and graphemes and the extraction of lists of words and collocations for the study of language acquisition or language teaching. Corpora have been used in linguistics since the first half of the Twentieth Century. Some of this work concerns word senses, and it is often strikingly modern. From the other hand, manual sense-tagging of a corpus is extremely costly, and at present very few sense-tagged corpora are available. Several efforts to create sense tagged corpora have or are being made: recently, the Linguistic Data Consortium distributes a corpus of approximately 200,000 sentences from the Brown Corpus and the Wall Street Journal in which all occurrences of 191 words are hand-tagged with their WordNet senses [Ng and Lee, 1996]. Also, the Cognitive Science Laboratory at Princeton has undertaken the hand-tagging of 1000 words from the Brown Corpus with WordNet senses [Miller et al., 1993] (so far, 200,000 words are available via ftp), and hand-tagging of 25 verbs a small segment of the Wall Street Journal (12,925 sentences) is also underway [Wiebe et al., 1997]. However, these corpora are far smaller than those typically used with statistical methods.

Several efforts have been made to automatically sense-tag a training corpus via bootstrapping methods. Hearst [Hearst, 1991] proposed an algorithm (CatchWord) which includes a training phase during which each occurrence of a set of nouns13 to be disambiguated is manually sense-tagged in several occurrences. Statistical information extracted from the context of these occurrences is then used to disambiguate other occurrences. If another occurrence can be disambiguated with certitude, the system automatically acquires additional statistical information from these newly disambiguated occurrences, thus improving its knowledge incrementally. Hearst indicates that an initial set of at least 10 occurrences is necessary for the procedure, and that 20 or 30 occurrences are necessary for high precision.

The problem of data sparseness, which is common for much corpus-based work, is especially severe for work in WSD. First, enormous amounts of text are required to ensure that all senses of a polysemous word are represented, given the vast disparity in frequency among senses.

Smoothing is used to get around the problem of infrequently occurring events, and in particular to ensure that non-observed events are not assumed to have a probability of zero.

Class-based models attempt to obtain the best estimates by combining observations of classes of words considered to belong to a common category. Class-based methods answer in part the problem of data sparseness, and eliminate the need for pre-tagged data. However, there is some information loss with these methods because the hypothesis that all words in the same class behave in a similar fashion is too strong.

Similarity-based methods exploit the same idea of grouping observations for similar words, but without re-grouping them into fixed classes. Each word has a potentially different set of similar words. Like many class-based methods, similarity-based methods exploit a similarity metric between patterns of co-occurrence.

Open problems

Context is the only means to identify the meaning of a polysemous word. Therefore, all work on sense disambiguation relies on the context of the target word to provide information to be used for its disambiguation. For data-driven methods, context also provides the prior knowledge with which current context is compared to achieve disambiguation. Broadly speaking, context is used in two ways:

- The bag of words approach: here, context is considered as words in some window surrounding the target word, regarded as a group without consideration for their relationships to the target in terms of distance, grammatical relations, etc.
- Relational information: context is considered in terms of some relation to the target, including distance from the target, syntactic relations, selectional preferences, orthographic properties, phrasal collocation, semantic categories, etc.

Information from micro-context, topical context, and domain contributes to sense selection, but the relative role and importance of information from the different contexts and their inter-relations are not well understood. Very few studies have used information

of all three types, and the focus in much recent work is on micro-context alone. This is another area where systematic study is needed for WSD.

Semantics

Basic Notions of Semantics

A perennial problem in semantics is the delineation of its subject matter. The term *meaning* can be used in a variety of ways, and only some of these correspond to the usual understanding of the scope of linguistic or computational semantics. We shall take the scope of semantics to be restricted to the literal interpretations of sentences in a context, ignoring phenomena like irony, metaphor, or *conversational implicature* [Grice, 1975], [Levinson 1983].

A standard assumption in computationally oriented semantics is that knowledge of the meaning of a sentence can be equated with knowledge of its truth conditions: that is, knowledge of what the world would be like if the sentence were true. This is not the same as knowing whether a sentence is true, which is (usually) an empirical matter, but knowledge of truth conditions is a prerequisite for such verification to be possible. *Meaning as truth conditions* needs to be generalized somewhat for the case of imperatives or questions, but is a common ground among all contemporary theories, in one form or another, and has an extensive philosophical justification, e.g., [Davidson 1969, 1973].

A semantic description of a language is some finitely stated mechanism that allows us to say, for each sentence of the language, what its truth conditions are. Just as for grammatical description, a semantic theory will characterize complex and novel sentences on the basis of their constituents: their meanings, and the manner in which they are put together. The basic constituents will ultimately be the meanings of words and morphemes. The modes of combination of constituents are largely determined by the syntactic structure of the language. In general, to each syntactic rule combining some sequence of child constituents into a parent constituent, there will correspond some semantic operation combining the meanings of the children to produce the meaning of the parent.

Chapter 2

Practical Applications of Semantics

Some natural language processing tasks (e.g., message routing, textual information retrieval, translation) can be carried out quite well using statistical or pattern matching techniques that do not involve semantics in the sense assumed above. However, performance on some of these tasks improves if semantic processing is involved.

Some tasks, however, cannot be carried out at all without semantic processing of some form. One important example application is that of database query, of the type chosen for the Air Travel Information Service (ATIS) task [Defense Advanced Research Projects Agency 1989]. For example, if a user asks, ``*Does every flight from London to San Francisco stop over in Reykyavik?*'' then the system needs to be able to deal with some simple semantic facts. Relational databases do not store propositions of the form *every X has property P* and so a logical inference from the meaning of the sentence is required. In this case, *every X has property P* is equivalent to *there is no X that does not have property P* and a system that knows this will also therefore know that the answer to the question is *no* if a non-stopping flight is found and *yes* otherwise.

Any kind of generation of natural language output (e.g., summaries of financial data, traces of KBS system operations) usually requires semantic processing. Generation requires the construction of an appropriate meaning representation, and then the production of a sentence or sequence of sentences which express the same content in a way that is natural for a reader to comprehend, e.g., [McKeown et. al, 1994]. To illustrate, if a database lists a 10 a.m.\ flight from London to Warsaw on the 1st--14th, and 16th--30th of November, then it is more helpful to answer the question *What days does that flight go?* by *Every day except the 15th* instead of a list of 30 days of the month. But to do this the system needs to know that the semantic representations of the two propositions are equivalent.

Chapter 2

Ontologies

In computer science, an **ontology** is a data model that represents a domain and is used to reason about the objects in that domain and the relations between them.

Ontologies are used in artificial intelligence, the semantic web, software engineering and information architecture as a form of knowledge representation about the world or some part of it. Ontologies generally describe:

- **Individuals**: the basic or "ground level" objects
- **Classes**: sets, collections, or types of objects
- **Attributes**: properties, features, characteristics, or parameters that objects can have and share
- **Relations**: ways that objects can be related to one another

Difference from philosophical ontology

The term ***ontology*** has its origin in philosophy, where it is used to name the fundamental branch of metaphysics concerned with existence. In other words, ***ontology*** is the name of a philosophical discipline in much the same way that *biology* names one of the scientific disciplines.

Its use in computer science retains little of its original philosophical meaning. In computer science ***an ontology*** is a specification of concepts and the relationships that may exist between those concepts for some Universe of discourse or community (see also taxonomy.) An ontology is therefore primarily a way for a community to agree upon the meanings of terms and relations so that they may reliably share knowledge and information. With terms and relations thus defined, automated processes that share this common definition can perform simple reasoning. There are numerous syntaxes for describing ontologies; for example, see Web Ontology Language.

It should be clear, therefore, that while it makes little sense to speak of ***an ontology*** in the context of philosophy and metaphysics, it makes perfect sense in the domain of computer

science. Computer scientists borrowed the term ***ontology*** from metaphysics because ***an ontology*** is a way of representing knowledge about the things that can exist in some Universe of discourse.

Elements of an ontology

Contemporary ontologies share many structural similarities, regardless of the language in which they are expressed. As mentioned above, most ontologies describe individuals (instances), classes (concepts), attributes, and relations. In this section each of these components is discussed in turn.

Individuals (Instances)

Individuals (instances) are the basic, "ground level" components of an ontology. The individuals in an ontology may include concrete objects such as people, animals, tables, automobiles, molecules, and planets, as well as abstract individuals such as numbers and words. Strictly speaking, an ontology need not include any individuals, but one of the general purposes of an ontology is to provide a means of classifying individuals, even if those individuals are not explicitly part of the ontology.

Classes (concepts)

Classes (Concepts) are abstract groups, sets, or collections of objects. They may contain individuals, other classes, or a combination of both. Some examples of classes:

- **Person**, the class of all people
- **Molecule**, the class of all molecules
- **Number**, the class of all numbers
- **Vehicle**, the class of all vehicles
- **Car**, the class of all cars
- **Individual**, representing the class of all individuals
- **Class**, representing the class of all classes
- **Thing**, representing the class of all things

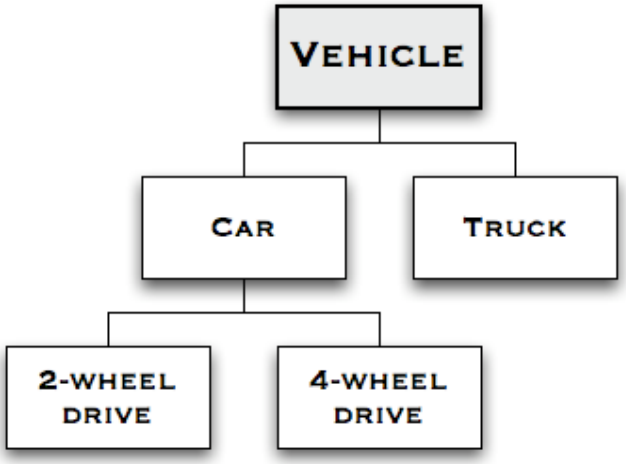
Ontologies vary on whether classes can contain other classes, whether a class can belong to itself, whether there is a universal class (that is, a class containing everything), etc.

Sometimes restrictions along these lines are made in order to avoid certain well-known paradoxes.

The classes of an ontology may be extensional or intentional in nature. A class is extensional if and only if it is characterized solely by its membership. More precisely, a class C is extensional if and only if for any class C', if C' has exactly the same members as C, then C and C' are identical. If a class does not satisfy this condition, then it is intentional. While extensional classes are more well-behaved and well-understood mathematically, as well as less problematic philosophically, they do not permit the fine grained distinctions that ontologies often need to make. For example, an ontology may want to distinguish between the class of all creatures with a kidney and the class of all creatures with a heart, even if these classes happen to have exactly the same members.

Importantly, a class can subsume or be subsumed by other classes. For example, Vehicle subsumes Car, since (necessarily) anything that is a member of the latter class is a member of the former. The subsumption relation is used to create a hierarchy of classes, typically with a maximally general class like **Thing** at the top, and very specific classes like **2002 Ford Explorer** at the bottom.

A **partition** is a set of related classes and associated rules that allow objects to be placed into the appropriate class. For example, this partial diagram of an ontology has a partition of the Car class into the classes 2-Wheel Drive and 4-Wheel Drive:



The partition rule determines if a particular car is placed in the 2-Wheel Drive or the 4-Wheel Drive class.

If the partition rule(s) guarantee that a single Car object cannot be in both classes, then the partition is called a **Disjoint Partition**. If the partition rules ensure that every concrete object in the super-class is an instance of at least one of the partition classes, then the partition is called an **Exhaustive Partition**.

Attributes

Objects in the ontology can be described by assigning attributes to them. Each attribute has at least a name and a value, and is used to store information that is specific to object it is attached to. For example the Ford Explorer object has attributes such as:

- *Name*: Ford Explorer
- *Number-of-doors*: 4
- *Engine*: {4.0L, 4.6L}
- *Transmission*: 6-speed

The value of an attribute can be a complex data type; in this example, the value of the attribute called *Engine* is a list of values, not just a single value.

If you did not define attributes for the concepts you would have either a taxonomy (if concept relationships are described) or a **Controlled Vocabulary**. These are useful, but are not considered true ontologies.

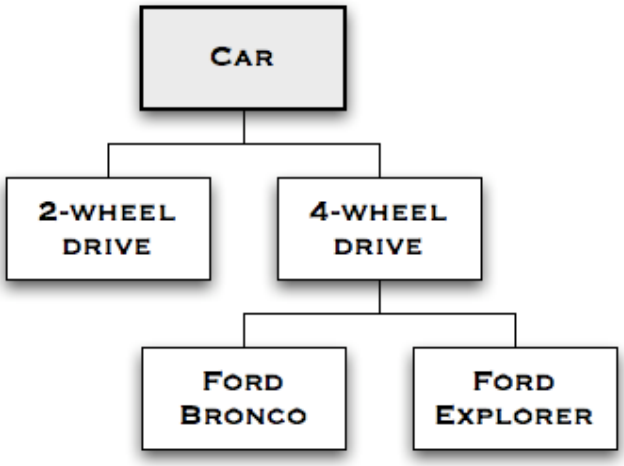
Relationships

An important use of attributes is to describe the relationships (also known as relations) between objects in the ontology. Typically a relation is an attribute whose value is another object in the ontology. For example in the ontology that contains the Ford Explorer and the Ford Bronco, the Ford Bronco object might have the following attribute:

Successor: Ford Explorer

This tells us that the Explorer is the model that replaced the Bronco. Much of the power of ontologies comes from the ability to describe these relations. Together, the set of relations describes the semantics of the domain.

The most important type of relation is the subsumption relation (written as *is-a*, *is-subtype-of* or *is-subclass-of*). This defines which objects are members of classes of objects. For example we have already seen that the Ford Explorer *is-a* 4-wheel drive, which in turn *is-a* Car:



The addition of the is-a relationships has created a hierarchical taxonomy; a tree-like structure that clearly depicts how objects relate to one another. In such a structure, each object is the 'child' of a 'parent class' (Some languages restrict the is-a relationship to one parent for all nodes, but many do not).

Another common type of relations is the Meronymy relation (written as *part-of*) that represents how objects combine together to form composite objects. For example, if we extended our example ontology to include objects like Steering Wheel, we would say that "Steering Wheel is-part-of Ford Explorer" since a steering wheel is one of the components of a Ford Explorer.

If we introduce part-of relationships to our ontology, we find that this simple and elegant tree structure quickly becomes complex and significantly more difficult to interpret manually. It is not difficult to understand why; an entity that is described as 'part of'

another entity might also be 'part of' a third entity. Consequently, entities may have more than one parent. The structure that emerges is known as a Directed Acyclic Graph (DAG).

As well as the standard is-a and part-of relations, ontologies often include additional types of relation that further refine the semantics they model. These relations are often domain-specific and are used to answer particular types of question.

For example in the domain of automobiles, we might define a *made-in* relationship which tells us where each car is built. So the Ford Explorer is *made-in* Louisville. The ontology may also know that Louisville is-in Kentucky and Kentucky is-a state of the USA. Software using this ontology could now answer a question like "which cars are made in America?"

Domain ontologies and upper ontologies

A domain ontology (or domain-specific ontology) models a specific domain, or part of the world. It represents the particular meanings of terms as they apply to that domain. For example the word *card* has many different meanings. An ontology about the domain of poker would model the "playing card" meaning of the word, while an ontology about the domain of computer hardware would model the "punch card" and "video card" meanings.

An upper ontology (or foundation ontology) is a model of the common objects that are generally applicable across a wide range of domain ontologies. It contains a core glossary in whose terms objects in a set of domains can be described. There are several standardized upper ontologies available for use, including Dublin Core, GFO, OpenCyc/ResearchCyc, SUMO, WordNet, and DOLCE.

Since domain ontologies represent concepts in very specific and often eclectic ways, they are often incompatible. As systems that rely on domain ontologies expand, they often need to merge domain ontologies into a more general representation. This presents a challenge to the ontology engineer. Different ontologies in the same domain can also arise due to different perceptions of the domain based on cultural background, education, ideology, or because a different representation language was chosen.

At present, merging ontologies is a largely manual process and therefore time-consuming and expensive. Using a foundation ontology to provide a common definition of core terms can make this process manageable. There are studies on generalized techniques for merging ontologies, but this area of research is still largely theoretical.

Ontology languages

An **ontology language** is a formal language used to encode the ontology. There are a number of such languages for ontologies, both proprietary and standards-based:

- **OWL** is a language for making ontological statements, developed as a follow-on from RDF and RDFS, as well as earlier ontology language projects including OIL, DAML and DAML+OIL. OWL is intended to be used over the World Wide Web, and all its elements (classes, properties and individuals) are defined as RDF resources, and identified by URIs.
- **KIF** is a syntax for first-order logic that is based on S-expressions.
- The **Cyc** project has its own ontology language called CycL, based on first-order predicate calculus with some higher-order extensions.

WordNet

WordNet is a semantic lexicon for the English language. It groups English words into sets of synonyms called synsets, provides short, general definitions, and records the various semantic relations between these synonym sets. The purpose is twofold: to produce a combination of dictionary and thesaurus that is more intuitively usable, and to support automatic text analysis and artificial intelligence applications. The database and software tools have been released under a BSD style license and can be downloaded and used freely. The database can also be browsed online.

As of 2005, the database contains about 150,000 words organized in over 115,000 synsets for a total of 203,000 word-sense pairs; in compressed form, it is about 12 megabytes in size.

WordNet distinguishes between nouns, verbs, adjectives and adverbs because they follow different grammatical rules. Every synset contains a group of synonymous words or collocations (a *collocation* is a sequence of words that go together to form a specific meaning, such as "car pool"); different senses of a word are in different synsets. The meaning of the synsets is further clarified with short defining *glosses*. A typical example synset with gloss is:

- good, right, ripe -- (most suitable or right for a particular purpose; "a good time to plant tomatoes"; "the right time to act"; "the time is ripe for great sociological changes")

Most synsets are connected to other synsets via a number of semantic relations. These relations vary based on the type of word, and include:

- Nouns
    - *hypernyms*: *Y* is a hypernym of *X* if every *X* is a (kind of) *Y*
    - *hyponyms*: *Y* is a hyponym of *X* if every *Y* is a (kind of) *X*
    - *coordinate terms*: *Y* is a coordinate term of *X* if *X* and *Y* share a hypernym
    - *holonym*: *Y* is a holonym of *X* if *X* is a part of *Y*
    - *meronym*: *Y* is a meronym of *X* if *Y* is a part of *X*
- Verbs
    - *hypernym*: the verb *Y* is a hypernym of the verb *X* if the activity *X* is a (kind of) *Y* (*travel* to *movement*)
    - *troponym*: the verb *Y* is a troponym of the verb *X* if the activity *Y* is doing *X* in some manner (*lisp* to *talk*)
    - *entailment*: the verb *Y* is entailed by *X* if by doing *X* you must be doing *Y* (*snoring* by *sleeping*)
    - *coordinate terms*: those verbs sharing a common hypernym
- Adjectives
    - *related nouns*

- *participle of verb*

- Adverbs
  - *root adjectives*

While semantic relations apply to all members of a synset because they share a meaning and are all mutually synonyms, words can also be connected to other words by lexical relations, including antonyms (opposites of each other) and derivationally related words.

WordNet also provides the *polysemy count* of a word: the number of synsets that contain the word. If a word participates in several synsets (i.e. has several senses), then typically some senses are much more common than others. WordNet quantifies this by the *frequency score*: in several sample texts all words were semantically tagged with the corresponding synset, and then it was counted how often a word appeared in a specific sense.

The morphology functions of the software distributed with the database try to deduce the lemma or root form of a word from the user's input; only the root form is stored in the database unless it has irregular inflected forms

Semantic Relatedness in a Semantic Network

According to Lee et. al. [1993], a "semantic network is broadly described as any representation interlinking nodes with arcs, where the nodes are concepts and the links are various kinds of relationships between concepts". The majority of the methods discussed in the present section use WordNet, a broad coverage semantic network created as an attempt "to model the lexical knowledge of a native speaker of English" [Richardson and Smeaton, 1995].

Computing Path Length

A natural way to evaluate semantic relatedness in a taxonomy, given its graphical representation, is "to evaluate the distance between the nodes corresponding to the items being compared – the shortest the path from one node to another, the similar they are.

Given multiple paths, one takes the length of the shortest one" [Resnik, 1995]. The first approach follows exactly this methodology.

Rada et al.'s Simple Edge Counting

Rada and colleagues [Rada et. al, 1989, Rada and Bicknell, 1989] describe a research effort directed towards improving quality of a bibliographic information retrieval system in a highly specific domain — biomedical literature. Unlike the other approaches below, which use WordNet, Rada et al.'s central knowledge source is MeSH (Medical Subject Headings), a hierarchical semantic network of over 15,000 terms used in indexing over five million articles in Medline, one of the world's largest bibliographic retrieval systems, maintained by the National Library of Medicine.

The principal assumption put forward by Rada and colleagues is that "the number of edges between terms in the MeSH hierarchy is a measure of conceptual distance between terms". Their distance $dist_{Retal}(ti, tj)$ between two terms is thus defined simply as

$$dist_{Retal}(ti, tj) = \text{minimal number of edges in a path from } ti \text{ to } tj \text{ .} \quad (1)$$

Even with such a simple distance function, the authors were able to obtain surprisingly good results. In part, their success can be explained by the following general observation of Lee *et al.* [1993]: "In the context of Quillian's semantic networks, shortest path lengths between two concepts are not sufficient to represent conceptual distance between those concepts. However, when the paths are restricted to IS-A links, the shortest path length does measure conceptual distance." Another component of their success is certainly the aforementioned specificity of the domain, which ensures relative homogeneity of the hierarchy.

Sussna's Depth-Relative Scaling

In Sussna's [1993, 1997] approach, each edge in the WordNet noun network is constructed as consisting of two arcs representing inverse relations. Each relation *r* has a weight or a range $[min_r;max_r]$ of weights associated with it: all *antonymy* arcs get the value of $min_r = max_r = 2.5$, *hypernymy, hyponymy, holonymy,* and *meronymy* have weights between $min_r = 1$ and $max_r = 2$. (Since *synonymy* is an intranode relation, its (non-existent) arcs get weight 0.) The point in the range for a relation *r* arc from node c_1 to node c_2 depends on the number n_r of arcs of the same type leaving c_1; namely,

$$w(c_1 \rightarrow c_2) = \max_r - \frac{\max_r - \min_r}{n_r(c_1)} \quad (2)$$

This is the *type-specific fanout* factor, which, according to Sussna, "reflects dilution of the *strength of connotation* between a source and target node" and "takes into account the possible asymmetry between the two nodes, where the strength of connotation in one direction differs from that in the other direction". The two inverse weights for an edge are averaged and scaled by depth *d* of the edge "within the overall 'tree'". The key motivation for scaling is Sussna's observation that sibling-concepts deeper in the tree appear to be more closely related to one another than those higher in the tree. The formula for the distance between adjacent nodes c_1 and c_2 then becomes

$$dist_S(c_1,c_2) = \frac{w(c_1 \rightarrow_r c_2) + w(c_2 \rightarrow_{r'} c_1)}{2d} \quad (3)$$

where *r* is the relation that holds between c_1 and c_2 and *r′* is its inverse *(i.e.,* the relation that holds between c_2 and c_1*).*

Finally, the semantic distance between two arbitrary nodes c_i and c_j is computed as the sum of the distances between the pairs of adjacent nodes along the shortest path connecting c_i and c_j .

Wu and Palmer's Conceptual Similarity

In a paper focusing on "semantic representation of verbs in computer systems and its impact on lexical selection problems in machine translation", Wu and Palmer [1994] devote a couple of paragraphs to introducing a metric that is somewhat specialized but nonetheless deserving of at least a brief mention. Very superficially, the key idea of the authors' approach to translating English verbs into Mandarin Chinese is to "project" verbs (and verb compounds) of both languages onto something they call "conceptual domains". The first immediate effect of the projection operation is that it separates different senses of verbs by placing them into different domains. Another important feature of conceptual domains — and the one that directly concerns us — is the fact that the concepts within a single domain can be organized in a strict hierarchical structure (namely, a tree) on which a measure of similarity can be defined.

Wu and Palmer define *Conceptual Similarity* between a pair of concepts $c_1$and c_2 as

$$sim_{WP}(c_1, c_2) = \frac{2 \times N3}{N1 + N2 + 2 \times N3} \quad (4)$$

where *N1* is the length (in number of nodes) of the path from c_1 to c_3, which is the least common superconcept of c_1 and c_2, *N2* is the length of the path from c_2 to c_3, and *N3* is the length of the path from c_3 to the root of the hierarchy. Note that *N3* represents the 'global' depth in the hierarchy, and to emphasize its role as a scaling factor more clearly, we can consider a translation of the above equation from the language of similarity into the language of *distance:*

$$dist_{WP}(c_1, c_2) = 1 - sim_{WP}(c_1, c_2) = \frac{N1 + N2}{N1 + N2 + 2 \times N3} \quad (5)$$

Leacock and Chodorow's Normalized Path Length

In the course of their attempt to alleviate the problem of sparseness of training data for a statistical local-context classifier, Leacock and Chodorow [1998] proposed the

following formula for computing the semantic similarity between words w_1 and w_2 (notation borrowed from [Resnik, 1995]):

$$sim_{LC}(w_1, w_2) = -\log \frac{\min_{c_1,c_2} len(c_1, c_2)}{2 \times D} \quad (6)$$

where D is the maximum depth of the taxonomy (also known as *height,* in graph theory), len(c_1,c_2) is the length of the shortest path between c_1 and c_2, and c_1,c_2, stand for "the set of concepts in the taxonomy that are senses of word w_1, w_2"[Resnik, 1995].

To avoid singularities, Leacock and Chodorow measure path lengths in nodes, rather than edges, so synonyms *(i.e.,* members of the same synset) are 1 unit of distance apart from each other.

Integrated Approaches

Like the methods in the preceding subsection, the final group of approaches that we present in this thesis attempt to counter problems inherent in a general ontology. These approaches incorporate an additional, and qualitatively different, knowledge source: all three techniques outlined below use corpus analysis to augment the information already present in the network. As a side-effect, this "provides a way of adapting a static knowledge structure to multiple contexts" [Resnik, 1995].

Resnik's Information-Based Approach

The key underlying idea of Resnik's [1995] approach is the intuition that one criterion of similarity between two concepts is "the extent to which they share information in common", which in an IS-A taxonomy can be determined by inspecting the relative position of a most specific concept that subsumes them both. This intuition seems to be indirectly captured by edge-counting methods (such as that

of Rada and colleagues) in that "if the minimal path of IS-A links between two nodes is long, that means it is necessary to go high in the taxonomy, to more abstract concepts, in order to find a least upper bound". An example given in [Resnik, 1995] is the difference in the relative positions of the most specific subsumer of **nickel** and **dime** — **coin** — and that of **nickel** and **credit card** — **medium of exchange** (Figure 2).

In mathematical terms, let us augment our taxonomy (whose set of concepts is denoted by *C)* with a function $p : C \rightarrow [0,1]$, such that for any $c \in C$, *p(c)* is the probability of encountering an *instance* of concept c. Following the standard definition from Information Theory, the *information content* of c is then — *log p(c)*. Finally, for a pair of concepts c_1 and c_2, we can define their semantic similarity as

$$sim_R(c_1, c_2) = \max_{c \in S(c_1, c_2)} [-\log p(c)] = -\log p(lso(c_1, c_2)) \quad (7)$$

where *S(*c_1, c_2) stands for the set of concepts that subsume both c_1 and c_2, and *lso(*c_1, c_2) stands for the most specific common subsumer *(lowest super-ordinate)* of c_1 and c_2.

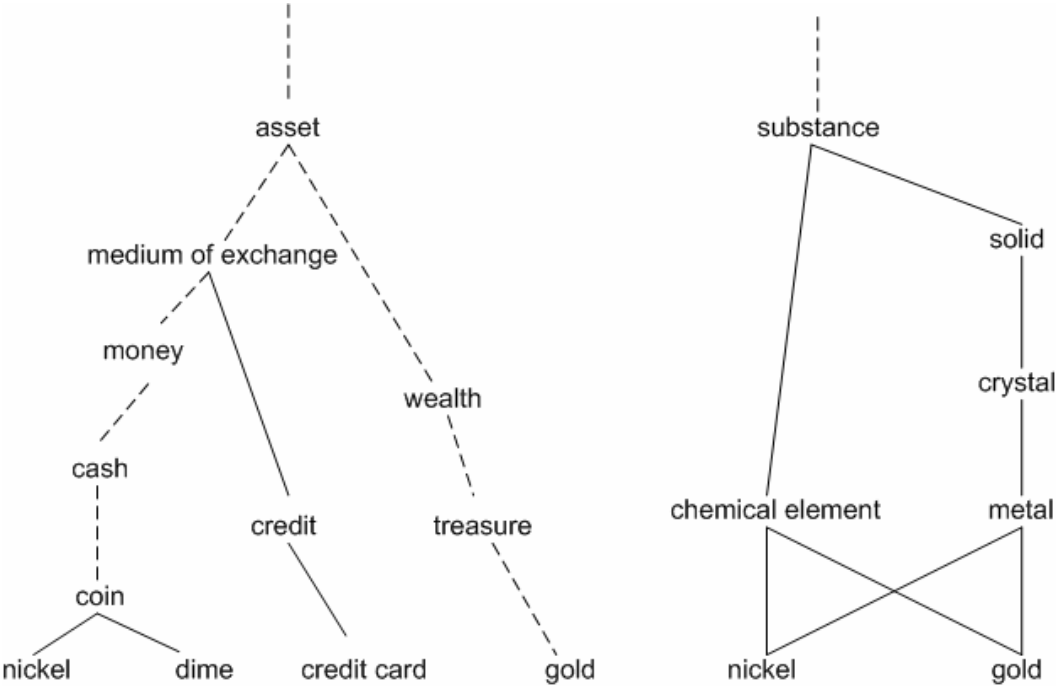


Figure 1: Fragment of the WordNet taxonomy. Solid lines represent IS-A links; dashed lines indicate that some intervening nodes have been omitted. Adapted from [Resnik, 1995]

One thing to note about the definition of *p* is that it is monotonic as one moves up the taxonomy: c_1 IS-A c_2 implies $p(c_1) < p(c_2)$ (whenever we encounter a **nickel**, we have encountered a **coin** (Figure 2), so *p*(nickel) ≤ *p*(coin)). In particular, if the taxonomy has a unique top node (such as *um-thing* in PENMAN Upper Model), its *p* is 1. As a consequence, the higher the position of the most specific subsumer for given two concepts in the taxonomy *(i.e.,* the more abstract it is), the lower the similarity. In particular, if the most specific subsumer of a pair of concepts is the top node, their similarity is 0.

Given the formula for similarity between two concepts, the similarity between two words w_1 and w_2 can be calculated as

$$sim_R(w_1, w_2) = \max_{c_1 \in s(w_1), c_2 \in s(w_2)} [sim(c_1, c_2)], \quad (8)$$

with *s(wi) (i* = 1,2).

In Resnik's experiments, frequencies of concepts in the taxonomy were estimated through noun frequencies gathered from the Brown Corpus of American English [Francis and Kucera, 1982], a 1-million-word "collection of text across genres ranging from news articles to science fiction". The key characteristic of his counting method is that an individual occurrence of any noun in the corpus "was counted as an occurrence of each taxonomic class containing it" (see below). For example, an occurrence of the noun *nickel* was, in accordance with Figure 2, counted towards the frequency of nickel, coin, and so forth. Note that, as a consequence of using raw (non-disambiguated) data, encountering a word will contribute to the counts of all its senses (if it is polysemous) and those of any of its homographs. So in case of *nickel,* the counts of nickel', chemical element, metal, etc., will also be increased.

Formally,

$$freq(c) = \sum_{n \in words(c)} count(n) \quad (9)$$

where words(c) is the set of words whose senses are subsumed by concept c (provided that subsumption is reflexive), and, adopting the maximum likelihood estimate (MLE) rule,

$$p(c) = \frac{freq(c)}{N} \quad (10)$$

where N is the total number of nouns in the corpus which are also present in WordNet.

Jiang and Conrath's Combined Approach

Resnik's approach described above attempts to deal with the problem of "varying link distances" [Resnik, 1995] by generally downplaying the role of network edges in the determination of the degree of semantic proximity: edges are used solely for locating super-ordinates of a pair of concepts; in particular, the number of links does not figure in any of the formulas pertaining to the method; numerical evidence comes from corpus statistics, which are associated with nodes.

Such a selective use of the structure of the taxonomy, however, has its drawbacks, one of which is the indistinguishability, in terms of semantic distance, of any two pairs of concepts having the same most-specific subsumer. Going back to Figure 2, sim_R(money, credit)=sim_R(dime, credit card) = - log p(medium of exchange), whereas, for a typical *edge-based* method such as Leacock and Chodorow's, clearly sim_{LC}(money, credit) ≠ sim_{LC}(dime, credit card).

Jiang and Conrath's [1997] idea was to synthesize edge- and node-based techniques (hence it is a *combined* approach) by effectively restoring the dominant function of network edges in similarity computations and using corpus statistics as a corrective factor. They hypothesized that the general formula for the weight of a link between a child-concept c_c and its parent-concept c_p in a hierarchy should be of the form

$$wt(c_c, c_p) = \left(\beta + (1-\beta)\frac{\overline{E}}{E(c_p)} \right) \left(\frac{d(c_p)+1}{d(c_p)} \right)^{\alpha} LS(c_c, c_p) T(c_c, c_p) \text{, (11)}$$

where E(cp) denotes the number of children of cp ("local density"), $\overline{E}$ denotes the average local density over the entire hierarchy, d(cp) the depth of the node cp in the hierarchy, LS(cc,cp) the strength of the link between cc and cp, T(cc,cp) the link-type coefficient, and the parameters $\alpha \in [0,\infty)$ and $\beta \in [0,1]$ control the degree of contribution of the node depth and the density factor, respectively. A careful reader may notice a parallel between the local density, node depth, and link-type factors in previous equation and type-specific fanout, edge depth, and relation weight of Sussna's approach. The emphases of the two research programs, however, have been different. Unlike Sussna, Jiang and Conrath to date have experimented only with a single link-type, IS-A (personal communication), which was assigned T of 1. Their investigation into the roles of the density and depth components have demonstrated that "they are not the major determinants of the overall edge weight": setting a=0.5 and β=0.3 resulted in "a small performance improvement" over the simplest case of a=0 and β=1 (i.e., giving no consideration to density or depth). The main focus of Jiang and Conrath's effort has thus been the link-strength factor, with the previous equation reduced to the special case

$$wt(cc,cp) = LS(cc,cp) \text{ (12)}$$

In the framework of the IS-A hierarchy, Jiang and Conrath postulated the strength LS(cc,,cp) of the link connecting a child-concept cc to its parent-concept cp to be proportionate to the conditional probability p(cc|cp) of encountering an instance of cc given an instance of cp. More specifically,

$$LS(cc,cp) = -\log p(cc|cp) \text{ (13)}$$

By definition,

$$p(c_c \mid c_p) = \frac{p(c_c \,\&\, c_p)}{p(c_p)} \quad \text{(14)}$$

If we adopt Resnik's scheme for assigning probabilities to concepts, then p(cc&cp) = p(cc), since any instance of a child is automatically an instance of its parent. Then,

$$p(c_c \mid c_p) = \frac{p(c_c)}{p(c_p)}, \quad (15)$$

and

$$LS(c_c, c_p) = IC(c_c) - IC(c_p), \quad (16)$$

if we let IC(c) stand for the information content of concept c.

As per common practice, the semantic distance between an arbitrary pair of nodes was taken to be the sum of the weights of the edges along the shortest path that connects the nodes:

$$dist_{JC}(c_1, c_2) = \sum_{c \in path(c_1, c_2) \setminus lso(c_1, c_2)} wt(c, par(c)) \quad (17)$$

Here, path(c1, c2) is the set of all the nodes in the shortest path from c1 to c2, and par(c) returns the parent of the node c. One of the elements of path(c1, c2) in an IS-A hierarchy will always be the most specific common subsumer of the two concepts, lso(c1,c2) the most specific common subsumer (lowest super-ordinate) of c1 and c2. Furthermore (and this explains its removal from path(c1, c2) in), it will be the only element without a parent in the same set.

Expanding the sum in the right-hand side of Equation 17, plugging in the expression for the edge weight from Equation 12, and performing necessary eliminations will result in the following final formulas for the semantic distance between concepts c1 and c2:

$$dist_{JC}(c_1, c_2) = IC(c_1) + IC(c_2) - 2 \times IC(lso(c_1, c_2)), \quad (18)$$

or

$$dist_{JC}(c_1, c_2) = 2 \log p(lso(c_1, c_2) - (\log p(c_1) + \log p(c_2)) \quad (19)$$

Lin's Universal Similarity Measure

Having noticed that all of the similarity measures known to him are tied to a particular application, domain, or resource, Lin [1998] undertook an attempt to define a measure of similarity that is both universal (applicable to arbitrary objects and "not presuming any form of knowledge representation") and theoretically justified ("derived from a set of assumptions" — instead of "directly by a formula" — so that "if the assumptions are deemed reasonable, the similarity measure necessarily follows"). In arriving at such a definition, he used the following three intuitions as a basis:

1. The similarity between *A* and *B* (throughout this subsection, *A* and *B* will denote *arbitrary objects*) is related to their commonality. The more commonality they share, the more similar they are.
2. The similarity between *A* and *B* is related to the differences between them. The more differences they have, the less similar they are.
3. The maximum similarity between *A* and *B* is reached when *A* and *B* are identical, no matter how much commonality they share.

Lin also found it necessary to introduce a few additional assumptions (and definitions), notably that the *commonality* between *A* and *B* is measured by the amount of information contained in "the proposition that states the commonalities" between them, formally

$$IC(common(A,B)), \quad (20)$$

and that the *difference* between *A* and *B* is measured by

$$IC(description(A,B)) - IC(common(A,B)), \quad (21)$$

where description(A, *B)* is a proposition describing what *A* and *B* are.

Given the above setting and the apparatus of Information Theory, Lin was able to prove the following

Similarity Theorem: The similarity between *A* and *B* is measured by the ratio between the amount of information needed to state their commonality and the information needed to fully describe what they are:

$$sim_L(A,B) = \frac{\log P(common(A,B))}{\log P(description(A,B))}, \quad (22)$$

His measure of similarity between two concepts in a taxonomy ensued as a corollary:

$$sim_L(c_1,c_2) = \frac{2 \times \log p(lso(c_1,c_2))}{\log p(c_1) + \log p(c_2)} \quad (23)$$

where the notation is consistent with Equations 7 and 19. (The probabilities *p(c)* are determined in a manner analogous to Resnik's $p_B(c)$ (Equation 10))

As Lin points out, Resnik's similarity measure (Equation 7) is "quite close" to sim_L. In fact, it can be shown that $sim_R(c_1,c_2) = 1/2IC(common(c_1, c_2))$. What may be a little more unexpected, Lin demonstrates that, under certain conditions, his similarity measure coincides with Wu and Palmer's $sim_{WP}(c_1,c_2)$ (Equation 4).

Natural Language Interfaces to Databases

Prototype Nlidbs had already appeared in the late sixties and early seventies. The best-known Nlidb of that period is Lunar [Woods et. al., 1972], a natural language interface to a database containing chemical analyses of moon rocks. Lunar and other early natural language interfaces were each built having a particular database in mind, and thus could not be easily modified to be used with different databases. (Although the internal representation methods used in Lunar were argued to facilitate independence between the database and other modules [Woods, 1968], the way that these were used was somewhat specific to that project's needs.

By the late seventies several more Nlidbs had appeared. Rendezvous [Codd, 1974] engaged the user in dialogues to help him/her formulate his/her queries. Ladder [Hendrix et. al., 1978] could be used with large databases, and it could be configured to interface to different underlying database management systems (Dbmss). Ladder used semantic grammars, a technique that interleaves syntactic and semantic processing. Although semantic grammars helped to implement systems with impressive characteristics, the resulting systems proved difficult to port to different application domains. Indeed, a different grammar had to be developed whenever Ladder was configured for a new application. As researchers started to focus on portable Nlidbs, semantic grammars were gradually abandoned. Planes [Waltz, 1978] and Philiqa1 [Scha, 1977] were some of the other Nlidbs that appeared in the late seventies.

Chat-80 [Warren and Pereira, 1982] is one of the best-known Nlidbs of the early eighties. Chat-80 was implemented entirely in Prolog. It transformed English questions into Prolog expressions, which were evaluated against the Prolog database. The code of Chat-80 was circulated widely, and formed the basis of several other experimental Nlidbs.

In the mid-eighties Nlidbs were a very popular area of research, and numerous prototype systems were being implemented. A large part of the research of that time was devoted to portability issues. For example, Team [Grosz, 1983] was designed to be easily configurable by database administrators with no knowledge of Nlidbs.

Ask [Thompson and Thompson, 1983] allowed end-users to teach the system new words and concepts at any point during the interaction. Ask was actually a complete information management system, providing its own built-in database, and the ability to interact with multiple external databases, electronic mail programs, and other computer applications. All the applications connected to Ask were accessible to the end-user through natural language requests. The user stated his/her requests in English, and Ask transparently generated suitable requests to the appropriate underlying systems.

Janus [Resnik, 1989] had similar abilities to interface to multiple underlying systems (databases, expert systems, graphics devices, etc). All the underlying systems could participate in the evaluation of a natural language request, without the user ever becoming

aware of the heterogeneity of the overall system. Janus is also one of the few systems to support temporal questions.

Although some of the numerous NLIDBs developed in the mid-eighties demonstrated impressive characteristics in certain application areas, NLIDBs did not gain the expected rapid and wide commercial acceptance. For example, in 1985 Ovum Ltd. [Johnson, 1985] was foreseeing that "By 1987 a natural language interface should be a standard option for users of DBMs and 'Information Centre' type software, and there will be a reasonable choice of alternatives." Since then, several commercially available NLIDBs have appeared, and some of them are claimed to be commercially successful. However, NLIDBs are still treated as research or exotic systems, rather than a standard option for interfacing to databases, and their use is certainly not wide-spread. The development of successful alternatives to NLIDBs, like graphical and form-based interfaces, and the intrinsic problems of NLIDBs (both discussed in the following section) are probably the main reasons for the lack of acceptance of NLIDBs.

In recent years there has been a significant decrease in the number of papers on NLIDBs published per year. Still, NLIDBs continue to evolve, adopting advances in the general natural language processing field, exploring architectures that transform NLIDBs into reasoning agents, and integrating language and graphics to exploit the advantages of both modalities, to name some of the lines of current research. Generic linguistic front-ends have also appeared. These are general-purpose systems that map natural language input to expressions of a logical language (e.g. the Cle system [Alshawi, 1992] – see also [Alshawi, 1992]). These generic front-ends can be turned into NLIDBs, by attaching additional modules that evaluate the logic expressions against a database.

The database is structured according to some model of data, and the NLIDB is designed to work with that data model. Database systems have also evolved a lot during the last decades. The term "database system" now denotes (at least in computer science) much more complex and principled systems than it used to denote in the past. Many of the underlying "database systems" of early NLIDBs would not deserve to be called database systems with today's standards.

In the early days of database systems, there was no concept of naive end-users accessing the data directly; this was done by an expert programmer writing a special computer program.

The reason for this was the 'navigational' nature of the data model used by these early database systems. Not only did the user need to know about the structure of the data in the application. He/she also needed to know many programming tricks to get at the data. The development of the relational model of data in the 1970's [Codd, 1970] had a major impact on database systems. In the relational model, the only storage structure is the table, and this was something that even naive users could understand. Relatively simple declarative query languages, such as SQL, were developed for this class of user.

Currently, there are two major developments in database technology that will have an impact on NLIDBs. The first is the growing importance of object-oriented database systems, and the second is the trend in relational database technology towards more complex storage structures to facilitate advanced data modelling. We note that both of these trends could make it harder to produce an NLIDB. They both reflect a tendency to concentrate on new complex database application areas, such as network management and computer-aided design, where the user is anything but naive, and the immediate access to the database will often be carried out by a layer of application software.

Previous work on Natural Language Interfaces for Question Answering

The early years: databases, cognitive science and limited domains

Research in Natural Language Interfaces for user query answering is not new: a number of systems attempting to understand and answer natural language questions have been developed since the early sixties [Simmons 1965] (Simmons reviewed 15 different systems which had been implemented to that date). These early systems, such as the BASEBALL program [Green et al., 1961] were based on retrieving information in a very limited domain (in this case baseball games played over one season in the American

league) from a database. Another early experiment in this direction was the SHRDLU system [Winograd 1972], which answered simple questions about a world constituted of moveable blocks.

Simmons [Simmons, 1973] presents one of the earliest generic question answering algorithms, which proceeds as follows:

1. Accumulate a database of semantic structures representing sentence meanings.
2. Select a set of structures that appears relevant to the question. Relevance is measured by the number of lexical concepts in common between the proposed answer and the question. This is done by ordering the candidates according to the number of Token values they have in common with the questions.
3. Match the question structure against each candidate. This is done by:
    - Determining if the head verb of the question matches the head verb of the candidate. If there is no direct match, a paraphrase rule is applied to see if the question structure can be transformed into the structure of the answer. Paraphrase rules are stored as part of the lexicon and an examination of the lexical structures of two words will be able to determine if there is a rule (path) connecting the two. If there is not, the set of words that the first transforms into is recursively examined to see if can be transformed into the second word. If this fails, the transformation rules are recursively applied to the second word to see if a match can be found. This procedure continues until either a match is found or an arbitrarily set depth is reached.
    - Applying the same procedure to the other words in the question and the candidate answer in order to transform the question structure into the form of the candidate answer.
    - Examining quantifiers and modalities to see if quantificational, tense and negation relationships are matched.
    - Examining the question's semantic structure to determine if the question word type (the wh-word) is present and satisfied in the answer.

The key to this algorithm is the notion of semantic structure, which became a key theme in natural language processing research in the seventies, with the emergence of systems

based on research in cognitive psychology, attempting to model human intelligence. Lehnert [Lehnert, 1978], for example, sought to understand the nature of questions, in particular their classification, based on ideas from dependency theory set out for example in Schank and Abelson [Schank and Abelson, 1977] . Related to this is the work of Dyer [Dyer, 1983], who built the BORIS system, which attempted to understand short narratives (in a restricted domain) and answer questions related to the stories. A similar approach was also taken by Bobrow, [Bobrow et al., 1977], who built the GUS system for modeling human dialogue.

The common characteristic of all these systems was their limited scope: the domain which the systems attempted to answer questions about was very limited and questions were restricted to that limited domain; linked to this was the fact that there was no attempt to find "real" user questions with systems only being able to answer the "toy" questions prepared by the researchers.

Beyond cognitive psychology: open-domain Natural Language Query Answering

A first attempt to move beyond the limited domain systems for natural language processing of user queries was the FAQFinder system of Burke [Burke et al., 1997] which tried to link users' questions to a set of previously stored "question and answer" files (taken from the "Frequently Asked Questions" posts of a number of newsgroups) by locating the most similar question in the document collection and therefore the most probable answer: questions could therefore be phrased at will on a very large number of different topics, the topics being limited by the previously stored question and answer files. The task performed by the FAQFinder system however is more accurately described as answer finding rather than question answering [Berger et al. 2000], who describe similar work trying to find a statistical relationship between questions and answers, rather than the semantic relationship described by Burke), i.e. the search for an answer to a question in a collection of ready-made answers, as opposed to a collection of generic documents: in other words, the system is not required to actively seek and construct an answer from unrestricted text or a knowledge base.

Semantic Processing to Question Answering

The extended use of the web has created a need for services that will help users to find information they need fast and without cost. As far as it concerns keyword queries, an interesting semantic searcher is SCORE [Sheth et al., 2002]. It uses automatic classification and information-extraction techniques together with metadata and ontology information to enable contextual multi-domain searches that try to understand the exact user information need expressed in a keyword query.

Combining semantic searching with natural language processing, leads to Jeeves [Askjeeves, 2000]. The system looks up the user's question in its own database and returns the list of matching questions which it knows how to answer. Then, the user selects the most appropriate entry in the list. By this way the system stalls the interaction. Also, the system does not have any result processing mechanisms and answers the users by a set of documents, which is not so acceptable in natural language question-answering systems.

Recent work on Natural Language Interfaces

Ontologies and Natural Language Interfaces

Ontologies have shown to be the right answer to knowledge structuring and modeling by providing a formal conceptualization of a particular domain that is shared by a group of people in an organization [O'Realy, 1998]. The rich, ontology-based semantic markup information in a knowledge repository opens the way to novel, sophisticated forms of question answering, which not only can potentially provide increased precision and recall compared to today's search engines, but are also capable of offering additional functionalities, such as i) proactively offering additional information about an answer, ii) providing measures of reliability and trust and/or iii) explaining how the answer was derived.

Most closely related to this philosophy is ONTOSEEK. ONTOSEEK is an information retrieval system coupled with an ontology [Guarino, 1999]. ONTOSEEK performs retrieval based on content instead of string-based retrieval. Queries are translated to conceptual graphs, but the problem in this step is according to the authors "in reducing to ontology-driven graph matching where individual nodes and arcs match if the ontology indicates that a subsumption relation holds between them". These graphs are semi-automatically constructed and users have to verify the links between different nodes in the graph via the designated user interface.

Another natural language interaction system which amalgamates Natural Language Processing (NLP), Logic, Ontologies and Information Retrieval techniques to provide answers to queries in a specific domain in real time is AQUA [Vargas-Vera and Motta, 2004]. AQUA translates English questions into logical queries that are then used to generate of proofs. AQUA is coupled with the AKT reference ontology for the academic domain. This ontology (written in OCML) currently contains people, organizations, research areas, projects, publications, technologies and events, and works as a pattern-matching, which means that it tries to find exact match with names in the ontology. The drawback is that the tests that have been carried out concern only a specific ontology with a specific grammar and the system does not take into account the semantics of the ontology. The evaluation tests were quite preliminary and limited.

A work for question answering on top of the British Telecom Digital Library is described by Cimiano et. al. [Cimiano et. al., 2006]. It is an approach to query answering over knowledge resources that makes use of different ontology management components within an application scenario of the BT Digital Library. The novelty of the approach lies in the combination of different semantic technologies providing a clear benefit for the application scenario considered. The drawback concerns the natural language interface were the translation of the natural language queries to structured queries relies on a limited and partially automatically generated lexicon for the underlying ontology. The lexicon specifies the possible lexical representations of the ontology elements in the user query. So, the disambiguation procedure is limited to the specific application and it is not automatic.

Chapter 2

Conclusions

Natural Language Interfaces for human-system interactions try to solve the problem of determining an answer to a question by searching for a response in a collection of documents or data or metadata or generally in an information repository. While research in this area spans almost three decades [Winograd 1972 and Dyer 1983 for early systems; Moldovan et al. 2003 for the most successful recent example], progress has been slow, and even the more successful recent systems are very limited [Moldovan et al. 2003 for example find a correct answer for just over 83% of the questions examined; the questions were however limited to closed-type questions requiring a single concept as an answer and avoided the more complicated queries asking "why" or "how"]. Moreover, research in natural language interfaces has been characterized by a lack of theoretical underpinnings, and a consequent confusion regarding the aim of such systems: while there had been an initial attempt to characterize in more detail the problem, either by developing generic natural language query answering algorithms [Simmons 1973] or by proposing a generic framework within which to work [e.g. Lehnert 1978 and 1986, working within the conceptual dependency model developed in cognitive science], there has been little in this direction since, with work such as Graesser and Franklin [Graesser and Franklin, 1990] being focused on developing a cognitive model of human question answering (and trying to partially implement this model) rather than providing a model for the problem of *automated natural language interaction systems*. Thus, it is often unclear what even Natural Language Interactions systems that are a typical application of natural language interaction systems are trying to achieve. At the same time, linguists and philosophers have been examining the problem of determining the general nature of questions and answers [Gadamer 1960, Eco Hiz 1978, Eco 1990, Ginzburg 1995a and 1995b] and what determines a "relevant" answer to a question [Grice 1967, Brown and Yule 1983, Wilson and Sperber 1986, Sperber and Wilson 1995]. Nevertheless there is little, if any, evidence of interaction between the theoretical and practical strands of question answering research.

A new theory therefore needs to be developed specifically for the practical problem of automated question answering, given that current systems, aiming at an ambitious

application of automated systems to an "open domain" with "open" questions, lack a solid theoretical underpinning and research in this area is limited by the ambiguity of what is being evaluated and the uncertainty of the direction research should take. The following chapters will address this problem by examining the what and how of question answering, as opposed to most current research, which is solely concerned with the how, presenting a clear theoretical foundation and a software framework which can provide an unambiguous model for work in this area. Before proceeding to the next chapter we will introduce the related technologies that were guide the developed system.

Related Technologies

The Stanford Log-linear Part-Of-Speech Tagger

The task of POS-tagging is to assign part of speech tags to words reflecting their syntactic category. Often words can belong to different syntactic categories in different contexts. Essentially then POS-tagging is a first attempt to disambiguate the sense of every word that constitutes the user's request. For example, for the sentence "Flies like a flower", there are four words and several possible tags, giving many sequences depicted below.

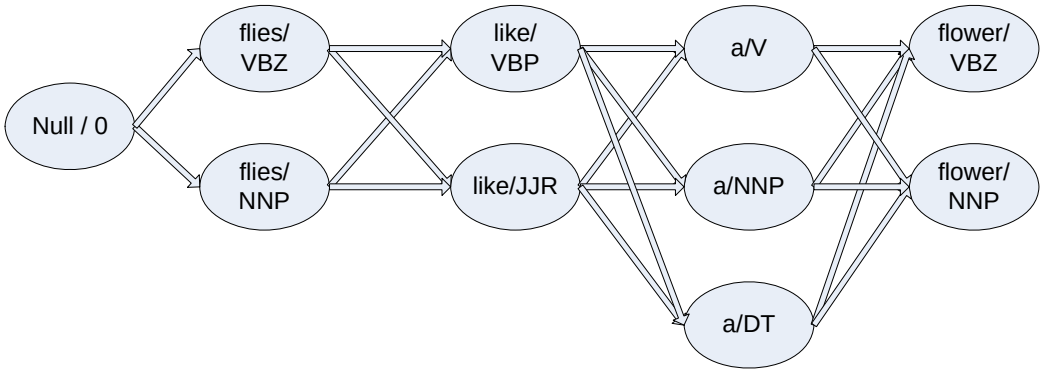


Figure 2: An example of the common problem of assigning a tag to a word in a sentence

Numerous approaches exist for automatic assignment of parts of speech ("tagging"), that use top performing methods, such as Hidden Markov Models [Brants, 2000], maximum entropy approaches [Ratnaparkhi, 1996] and transformation-based learning. The Stanford

Log-Linear POS Tagger adopted a maximum entropy approach, because it allows the inclusion of diverse sources of information without causing fragmentation and without necessarily assuming independence between the predictors [Toutanova et. all, 2000, 2003].

The part-of-speech tagger demonstrates the following ideas: (i) explicit use of both preceding and following tag contexts (past and future tag identity) via a dependency network representation, (ii) broad use of lexical features, including jointly conditioning on multiple consecutive words, (iii) effective use of priors in conditional log-linear models, and (iv) finegrained modeling of unknown word features. By implementing these ideas, the resulting Stanford tagger (http://nlp.stanford.edu/software/tagger.shtml) gives 96.86% accuracy on the Penn Treebank (http://www.cis.upenn.edu/~treebank/), an error reduction of 4.4% on the best previous single automatically learned tagging result and 86.91% on previously unseen words.

The procedure for the development of the tagger started with a maximum entropy based tagger that uses features very similar to the ones proposed in Ratnaparkhi [Ratnaparkhi, 1996]. The Stanford pos-tagger learns a log-linear conditional probability model from tagged text, using a maximum entropy method. This model assigns a probability for every tag *t* in the set *T* of possible tags given a word and its context *h*, which is usually defined as the sequence of several words and tags preceding the word. This model can be used for estimating the probability of a tag sequence t1…tn given a sentence w1…wn

The idea of maximum entropy modeling is to choose the probability distribution *p* that has the highest entropy out of those distributions that satisfy a certain set of constraints. The constraints restrict the model to behave in accordance with a set of statistics collected from the training data. The statistics are expressed as the expected values of appropriate functions defined on the contexts h and tags t. In particular, the constraints demand that the expectations of the features for the model match the empirical expectations of the features over the training data.

Some commonly used statistics for part of speech tagging are: how often a certain word was tagged in a certain way; how often two tags appeared in sequence or how often three

tags appeared in sequence. These look a lot like the statistics a Markov Model would use. However, in the maximum entropy framework it is possible to easily define and incorporate much more complex statistics, not restricted to n-gram sequences.

Having defined a set of constraints that the model should accord with, the target is to find the model satisfying the constraints that maximizes the conditional entropy of p . The intuition is that such a model assumes nothing apart from that it should satisfy the given constraints.

The approximation of *p(h,t)* (joint distribution of contexts and tags), is calculated by the product of $\tilde{p}(h)$, the empirical distribution of histories *h* and the conditional distribution $p(h,t) \approx \tilde{p}(h) \cdot p(t \mid h)$ [Toutanova et. al, 2000].

Ontology Description Standard

In this section we present an overview of the and ontology description standard on which the ontology disambiguation of the OntoNL relies; the OWL ontology definition language.

The Web Ontology Language (OWL)

The *Web Ontology Language (OWL)* [McGuinness and Van Harmelen, 2004] is the dominant standard in ontology definition. OWL has been developed according to the description logics paradigm and uses RDF(S) [Brickley and Guha, 2004], [Manola and Milles, 2004] syntax.

The Semantic Web is a vision for the future of the Web in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web. The Semantic Web will build on XML's ability to define customized tagging schemes and RDF's flexible approach to representing data. The first level above RDF required for the Semantic Web is an ontology language what can formally describe the meaning of terminology used in Web documents. If machines are

expected to perform useful reasoning tasks on these documents, the language must go beyond the basic semantics of RDF Schema. The OWL Use Cases and Requirements Document provides more details on ontologies, motivates the need for a Web Ontology Language in terms of six use cases, and formulates design goals, requirements and objectives for OWL.

- OWL has been designed to meet this need for a Web Ontology Language. OWL is part of the growing stack of W3C recommendations related to the Semantic Web.
- XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.
- XML Schema is a language for restricting the structure of XML documents and also extends XML with datatypes.
- RDF is a datamodel for objects ("resources") and relations between them, provides a simple semantics for this datamodel, and these datamodels can be represented in an XML syntax.
- RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.
- OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

Three OWL species of increasing descriptive power have been specified: OWL Lite, OWL DL and OWL Full. The basic functionality provided by OWL is following:

- Import of XML Schema Datatypes, through the `rdfs:Datatype` construct, for the representation of simple types that extend or restrict the basic datatypes (e.g. ranges etc.).

- Definition of OWL Classes, using the `owl:class` construct, for the representation of sets of individuals sharing some properties. Class hierarchies may be defined using the `rdfs:subClassOf` construct.
- Definition of OWL properties, for the representation of the features of the OWL class individuals. Two kinds of properties are provided by OWL: (a) *Object Properties*, which relate individuals of one OWL class (the property domain) with individuals of another OWL class (the property range). Object properties are defined using the `owl:objectProperty` construct; and (b) *Datatype Properties*, which relate individuals belonging to one OWL class (the domain of the property) with values of a given datatype (the range of the property). Datatype properties are defined using the `owl:datatypeProperty` construct.

Property hierarchies may be defined using the `rdfs:subPropertyOf` construct.

- Definition of class individuals.
- Definition of restrictions, using the `owl:Restriction` construct, including type restrictions, cardinality restrictions and value restrictions.

The following OWL Lite features related to RDF Schema are included.

Class: A class defines a group of individuals that belong together because they share some properties. For example, Deborah and Frank are both members of the class Person. Classes can be organized in a specialization hierarchy using subClassOf. There is a built-in most general class named Thing that is the class of all individuals and is a superclass of all OWL classes. There is also a built-in most specific class named Nothing that is the class that has no instances and a subclass of all OWL classes.

rdfs:subClassOf: Class hierarchies may be created by making one or more statements that a class is a subclass of another class. For example, the class Person could be stated to be a subclass of the class Mammal. From this a reasoner can deduce that if an individual is a Person, then it is also a Mammal.

rdf:Property: Properties can be used to state relationships between individuals or from individuals to data values. Examples of properties include hasChild, hasRelative, hasSibling, and hasAge. The first three can be used to relate an instance of a class Person to another instance of the class Person (and are thus occurences of ObjectProperty), and the last (hasAge) can be used to relate an instance of the class Person to an instance of the datatype Integer (and is thus an occurence of DatatypeProperty). Both owl:ObjectProperty and owl:DatatypeProperty are subclasses of the RDF class rdf:Property.

rdfs:subPropertyOf: Property hierarchies may be created by making one or more statements that a property is a subproperty of one or more other properties. For example, hasSibling may be stated to be a subproperty of hasRelative. From this a reasoner can deduce that if an individual is related to another by the hasSibling property, then it is also related to the other by the hasRelative property.

rdfs:domain: A domain of a property limits the individuals to which the property can be applied. If a property relates an individual to another individual, and the property has a class as one of its domains, then the individual must belong to the class. For example, the property hasChild may be stated to have the domain of Mammal. From this a reasoner can deduce that if Frank hasChild Anna, then Frank must be a Mammal. Note that rdfs:domain is called a global restriction since the restriction is stated on the property and not just on the property when it is associated with a particular class. See the discussion below on property restrictions for more information.

rdfs:range: The range of a property limits the individuals that the property may have as its value. If a property relates an individual to another individual, and the property has a class as its range, then the other individual must belong to the range class. For example, the property hasChild may be stated to have the range of Mammal. From this a reasoner can deduce that if Louise is related to Deborah by the hasChild property, (i.e., Deborah is the child of Louise), then Deborah is a Mammal. Range is also a global restriction as is domain above. Again, see the discussion below on local restrictions (e.g. AllValuesFrom) for more information.

Individual : Individuals are instances of classes, and properties may be used to relate one individual to another. For example, an individual named Deborah may be described as an instance of the class Person and the property hasEmployer may be used to relate the individual Deborah to the individual StanfordUniversity.

The following OWL Lite features are related to equality or inequality.

equivalentClass : Two classes may be stated to be equivalent. Equivalent classes have the same instances. Class equivalence can be used to create synonymous classes. For example, Car can be stated to be equivalentClass to Automobile. From this a reasoner can deduce that any individual that is an instance of Car is also an instance of Automobile and vice versa.

equivalentProperty: Two properties may be stated to be equivalent. Equivalent properties relate one individual to the same set of other individuals. Property equivalenve may be used to create synonymous properties. For example, hasLeader may be stated to be the equivalentProperty to hasHead. From this a reasoner can deduce that if X is related to Y by the property hasLeader, X is also related to Y by the property hasHead and vice versa. A reasoner can also deduce that hasLeader is a subproperty of hasHead and hasHead is a subProperty of hasLeader.

sameAs: Two individuals may be stated to be the same. These constructs may be used to create a number of different names that refer to the same individual. For example, the individual Deborah may be stated to be the same individual as DeborahMcGuinness.

differentFrom: An individual may be stated to be different from other individuals. For example, the individual Frank may be stated to be different from the individuals Deborah and Jim. Thus, if the individuals Frank and Deborah are both values for a property that is stated to be functional (thus the property has at most one value), then there is a contradiction. Explicitly stating that individuals are different can be important in when using languages such as OWL (and RDF) that do not assume that individuals have one and only one name. For example, with no additional information, a reasoner will not deduce that Frank and Deborah refer to distinct individuals.

AllDifferent: A number of individuals may be stated to be mutually distinct in one AllDifferent statement. For example, Frank, Deborah, and Jim could be stated to be mutually distinct using the AllDifferent construct. Unlike the differentFrom statement above, this would also enforce that Jim and Deborah are distinct (not just that Frank is distinct from Deborah and Frank is distinct from Jim). The AllDifferent construct is particularly useful when there are sets of distinct objects and when modelers are interested in enforcing the unique names assumption within those sets of objects. It is used in conjunction with distinctMembers to state that all members of a list are distinct and pairwise disjoint.

There are special identifiers in OWL Lite that are used to provide information concerning properties and their values. The distinction between ObjectProperty and DatatypeProperty is mentioned above in the property description.

inverseOf: One property may be stated to be the inverse of another property. If the property P1 is stated to be the inverse of the property P2, then if X is related to Y by the P2 property, then Y is related to X by the P1 property. For example, if hasChild is the inverse of hasParent and Deborah hasParent Louise, then a reasoner can deduce that Louise hasChild Deborah.

TransitiveProperty: Properties may be stated to be transitive. If a property is transitive, then if the pair (x,y) is an instance of the transitive property P, and the pair (y,z) is an instance of P, then the pair (x,z) is also an instance of P. For example, if ancestor is stated to be transitive, and if Sara is an ancestor of Louise (i.e., (Sara,Louise) is an instance of the property ancestor) and Louise is an ancestor of Deborah (i.e., (Louise,Deborah) is an instance of the property ancestor), then a reasoner can deduce that Sara is an ancestor of Deborah (i.e., (Sara,Deborah) is an instance of the property ancestor). OWL Lite (and OWL DL) impose the side condition that transitive properties (and their superproperties) cannot have a maxCardinality 1 restriction. Without this side-condition, OWL Lite and OWL DL would become undecidable languages. See the property axiom section of the OWL Semantics and Abstract Syntax document for more information.

SymmetricProperty: Properties may be stated to be symmetric. If a property is symmetric, then if the pair (x,y) is an instance of the symmetric property P, then the pair (y,x) is also an instance of P. For example, friend may be stated to be a symmetric property. Then a reasoner that is given that Frank is a friend of Deborah can deduce that Deborah is a friend of Frank.

FunctionalProperty : Properties may be stated to have a unique value. If a property is a FunctionalProperty, then it has no more than one value for each individual (it may have no values for an individual). This characteristic has been referred to as having a unique property. FunctionalProperty is shorthand for stating that the property's minimum cardinality is zero and its maximum cardinality is 1. For example, hasPrimaryEmployer may be stated to be a FunctionalProperty. From this a reasoner may deduce that no individual may have more than one primary employer. This does not imply that every Person must have at least one primary employer however.

InverseFunctionalProperty: Properties may be stated to be inverse functional. If a property is inverse functional then the inverse of the property is functional. Thus the inverse of the property has at most one value for each individual. This characteristic has also been referred to as an unambiguous property. For example, hasUSSocialSecurityNumber (a unique identifier for United States residents) may be stated to be inverse functional (or unambiguous). The inverse of this property (which may be referred to as isTheSocialSecurityNumberFor) has at most one value for any individual in the class of social security numbers. Thus any one person's social security number is the only value for their isTheSocialSecurityNumberFor property. From this a reasoner can deduce that no two different individual instances of Person have the identical US Social Security Number. Also, a reasoner can deduce that if two instances of Person have the same social security number, then those two instances refer to the same individual.

OWL allows restrictions to be placed on how properties can be used by instances of a class. These type are used within the context of an **owl:Restriction.** The **owl:onProperty** element indicates the restricted property. The following two restrictions limit which values can be used while the next section's restrictions limit how many values can be used.

allValuesFrom: The restriction allValuesFrom is stated on a property with respect to a class. It means that this property on this particular class has a local range restriction associated with it. Thus if an instance of the class is related by the property to a second individual, then the second individual can be inferred to be an instance of the local range restriction class. For example, the class Person may have a property called hasDaughter restricted to have allValuesFrom the class Woman. This means that if an individual person Louise is related by the property hasDaughter to the individual Deborah, then from this a reasoner can deduce that Deborah is an instance of the class Woman. This restriction allows the property hasDaughter to be used with other classes, such as the class Cat, and have an appropriate value restriction associated with the use of the property on that class. In this case, hasDaughter would have the local range restriction of Cat when associated with the class Cat and would have the local range restriction Person when associated with the class Person. Note that a reasoner can not deduce from an allValuesFrom restriction alone that there actually is at least one value for the property.

someValuesFrom: The restriction someValuesFrom is stated on a property with respect to a class. A particular class may have a restriction on a property that at least one value for that property is of a certain type. For example, the class SemanticWebPaper may have a someValuesFrom restriction on the hasKeyword property that states that some value for the hasKeyword property should be an instance of the class SemanticWebTopic. This allows for the option of having multiple keywords and as long as one or more is an instance of the class SemanticWebTopic, then the paper would be consistent with the someValuesFrom restriction. Unlike allValuesFrom, someValuesFrom does not restrict all the values of the property to be instances of the same class. If myPaper is an instance of the SemanticWebPaper class, then myPaper is related by the hasKeyword property to at least one instance of the SemanticWebTopic class. Note that a reasoner can not deduce (as it could with allValuesFrom restrictions) that all values of hasKeyword are instances of the SemanticWebTopic class

OWL includes a limited form of cardinality restrictions. OWL cardinality restrictions are referred to as local restrictions, since they are stated on properties with respect to a particular class. That is, the restrictions constrain the cardinality of that property on

instances of that class. OWL Lite cardinality restrictions are limited because they only allow statements concerning cardinalities of value 0 or 1 (they do not allow arbitrary values for cardinality, as is the case in OWL DL and OWL Full).

minCardinality: Cardinality is stated on a property with respect to a particular class. If a minCardinality of 1 is stated on a property with respect to a class, then any instance of that class will be related to at least one individual by that property. This restriction is another way of saying that the property is required to have a value for all instances of the class. For example, the class Person would not have any minimum cardinality restrictions stated on a hasOffspring property since not all persons have offspring. The class Parent, however would have a minimum cardinality of 1 on the hasOffspring property. If a reasoner knows that Louise is a Person, then nothing can be deduced about a minimum cardinality for her hasOffspring property. Once it is discovered that Louise is an instance of Parent, then a reasoner can deduce that Louise is related to at least one individual by the hasOffspring property. From this information alone, a reasoner can not deduce any maximum number of offspring for individual instances of the class parent. In OWL Lite the only minimum cardinalities allowed are 0 or 1. A minimum cardinality of zero on a property just states (in the absence of any more specific information) that the property is optional with respect to a class. For example, the property hasOffspring may have a minimum cardinality of zero on the class Person (while it is stated to have the more specific information of minimum cardinality of one on the class Parent).

maxCardinality: Cardinality is stated on a property with respect to a particular class. If a maxCardinality of 1 is stated on a property with respect to a class, then any instance of that class will be related to at most one individual by that property. A maxCardinality 1 restriction is sometimes called a functional or unique property. For example, the property hasRegisteredVotingState on the class UnitedStatesCitizens may have a maximum cardinality of one (because people are only allowed to vote in only one state). From this a reasoner can deduce that individual instances of the class USCitizens may not be related to two or more distinct individuals through the hasRegisteredVotingState property. From a maximum cardinality one restriction alone, a reasoner can not deduce a minimum cardinality of 1. It may be useful to state that certain classes have no values for a

particular property. For example, instances of the class UnmarriedPerson should not be related to any individuals by the property hasSpouse. This situation is represented by a maximum cardinality of zero on the hasSpouse property on the class UnmarriedPerson.

cardinality: Cardinality is provided as a convenience when it is useful to state that a property on a class has both minCardinality 0 and maxCardinality 0 or both minCardinality 1 and maxCardinality 1. For example, the class Person has exactly one value for the property hasBirthMother. From this a reasoner can deduce that no two distinct individual instances of the class Mother may be values for the hasBirthMother property of the same person.

Alternate namings for these restricted forms of cardinality were discussed. Current recommendations are to include any such names in a front end system.

OWL Lite contains an intersection constructor but limits its usage.

intersectionOf: OWL Lite allows intersections of named classes and restrictions. For example, the class EmployedPerson can be described as the intersectionOf Person and EmployedThings (which could be defined as things that have a minimum cardinality of 1 on the hasEmployer property). From this a reasoner may deduce that any particular EmployedPerson has at least one employer.

Both OWL DL and OWL Full use the same vocabulary although OWL DL is subject to some restrictions. Roughly, OWL DL requires type separation (a class can not also be an individual or property, a property can not also be an individual or class). This implies that restrictions cannot be applied to the language elements of OWL itself (something that is allowed in OWL Full). Furthermore, OWL DL requires that properties are either ObjectProperties or DatatypeProperties: DatatypeProperties are relations between instances of classes and RDF literals and XML Schema datatypes, while ObjectProperties are relations between instances of two classes. The OWL Semantics and Abstract Syntax document explains the distinctions and limitations. We describe the OWL DL and OWL Full vocabulary that extends the constructions of OWL Lite below.

oneOf: (enumerated classes): Classes can be described by enumeration of the individuals that make up the class. The members of the class are exactly the set of enumerated individuals; no more, no less. For example, the class of daysOfTheWeek can be described by simply enumerating the individuals Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday. From this a reasoner can deduce the maximum cardinality (7) of any property that has daysOfTheWeek as its allValuesFrom restriction.

hasValue: (property values): A property can be required to have a certain individual as a value (also sometimes referred to as property values). For example, instances of the class of dutchCitizens can be characterized as those people that have theNetherlands as a value of their nationality. (The nationality value, theNetherlands, is an instance of the class of Nationalities).

disjointWith: Classes may be stated to be disjoint from each other. For example, Man and Woman can be stated to be disjoint classes. From this disjointWith statement, a reasoner can deduce an inconsistency when an individual is stated to be an instance of both and similarly a reasoner can deduce that if A is an instance of Man, then A is not an instance of Woman.

unionOf, complementOf, intersectionOf (Boolean combinations): OWL DL and OWL Full allow arbitrary Boolean combinations of classes and restrictions: unionOf, complementOf, and intersectionOf. For example, using unionOf, we can state that a class contains things that are either USCitizens or DutchCitizens. Using complementOf, we could state that children are not SeniorCitizens. (i.e. the class Children is a subclass of the complement of SeniorCitizens). Citizenship of the European Union could be described as the union of the citizenship of all member states.

minCardinality, maxCardinality, cardinality (full cardinality): While in OWL Lite, cardinalities are restricted to at least, at most or exactly 1 or 0, full OWL allows cardinality statements for arbitrary non-negative integers. For example the class of DINKs ("Dual Income, No Kids") would restrict the cardinality of the property hasIncome to a minimum cardinality of two (while the property hasChild would have to be restricted to cardinality 0).

complex classes : In many constructs, OWL Lite restricts the syntax to single class names (e.g. in subClassOf or equivalentClass statements). OWL Full extends this restriction to allow arbitrarily complex class descriptions, consisting of enumerated classes, property restrictions, and Boolean combinations. Also, OWL Full allows classes to be used as instances (and OWL DL and OWL Lite do not).

RDF Query Languages

Someone may ask why not SQL or XML. Query languages are typically designed to be applied to data corresponding to a particular data model. For example, SQL is used to retrieve, create, modify, and delete data represented in (a variation of) the relational model of data. Similarly, XQuery is used to locate and retrieve (but not yet update) data that is represented in the XPath data model, XDM [XDM]. It is sometimes possible to use one language to query data represented in a data model other than that for which the language was designed. This may be accomplished by *mapping* the data from its native data model into the query language's data model, but the question then would be at what cost.

Each of these data models and corresponding query languages has advantages and disadvantages. SQL and the relational model are well designed to represent highly structured data such as that used by many business processes. Such data usually includes a value for every column of every table. SQL, but not the pure relational model supports a special value—called the *null value*—to represent data that is missing, unknown, or inapplicable. The possibility of null values complicates the definition of and queries written in the SQL language, which makes SQL a bit awkward for use in dealing with less structured information. The syntax of the SQL language focuses on identifying data for which most or all components are available and combining data based on the values of those components. In particular, combining data from two or more tables is specified by explicit SQL operators such as JOIN or UNION.

XPath, XQuery, and the XPath Data Model are all directed at support of less regular data. These languages and their data model function well when presented with data in which

most or all of the values are present, but they also function well when applied to data in which many values are not represented at all. Such data, often called "semi-structured data", is quite common in the XML tree-structured world in which elements and attributes may be optional and omitted entirely from instance data. It has been argued that the XPath Data Model represents a superset of, and could thus supercede, the relational model. Based on [Melton, 2006] that conclusion is rejected because of the inherent overhead required by the XPath Data Model to self-identify each piece of data versus SQL's regular structures. That is, each datum in an SQL table takes its "name" from the name of the column in which it appears, while the XPath Data Model requires that the name of each datum be given explicitly—on every instance—as the name of the element or attribute of which it is the value. XQuery syntax is optimized for building new XML documents from one or more inherently semi-structured XML documents, easily accommodating the complete absence of some data. Combining data from two or more (source) XML documents is specified through the use of explicit operators such as a comma in a for expression or the keyword union.

To which of these camps does RDF belong? Every RDF triple comprises a subject, a predicate (or *property*), and an object, which—even though the subject or the object may not be explicit (that is, they may be represented by "blank nodes") in some triples—implies that RDF is structured data. However, RDF defines a graph-based or network data model, which—like tree-based data models such as XML—readily manages the concept of optional data. In fact, [RDF] states that "it is not assumed that complete information about any resource is available". The conclusion based on [Melton, 2006] is that the RDF model is structured in the same sense that the relational model is: every provided assertion—SQL row or RDF triple—is complete (with the occasional missing datum represented by an SQL null value or an RDF blank node), but there may be assertions missing from the table or graph.

Surely, they say, those "reasoning engines" that operate on RDF and OWL constructs could just as easily operate on them in the context of a relational database as in a new kind of collection manager. In that case, why wouldn't SQL be the language of choice for answering questions before and after those engines have done their jobs?

In the question why RDF isn't, or shouldn't be, stored in a relational database and then queried using SQL, a good answer is this: SPARQL syntax makes virtually all join operations implicit, while SQL syntax usually makes them explicit. A consequence of this design decision is that the SQL expressions to answer typical questions that will be asked against RDF collections tend to be much larger and somewhat more difficult to create because of the need to write explicit join operations and the requisite explicit join conditions. Because typical questions asked of RDF involve several, sometimes many, join operations, SPARQL provides a more compact notation that is perhaps easier to get right with less debugging time spent.

In the question why XQuery isn't more appropriate, since RDF is typically serialized as XML, the response is similar: the number of explicit join operations in for clauses and the number of join conditions required in the where clauses probably makes the XQuery expressions more tedious to write and to debug than the corresponding SPARQL queries.

SPARQL follows this well-trodden path, offering a simple, reasonably familiar (to SQL users) SELECT query form andbuilds on previous RDF query languages such as rdfDB (http://www.guha.com/rdfdb/), RDQL, and SeRQL (http://www.openrdf.org/doc/sesame/users/ch06.html), and has several valuable new features of its own. A key aspect of the Semantic Web idea is the ability to extract and query information held across many different ad hoc, third-party apps, services, or repositories. That ability to move in and among various data sources is key to the Semantic Web idea of the mash up. SPARQL, which is both a query language and a data access protocol, has the ability to become a key component in the Semantic Web applications: as a standard backed by a flexible data model, it can provide a common query mechanism for all the Semantic Web applications. We choose SPARQL as the query language to represent the natural language queries after the syntactic and semantic disambiguation since SPARQL is defined in terms of the W3C's RDF data model and will work for any data source that can be mapped into RDF.

The 14 June 2007 draft, along with the other working drafts for SPARQL, are a Candidate Recommendation in W3C; it been widely reviewed and satisfies the requirements documented in *RDF Data Access Use Cases and Requirements*.

For the above reasons as well as because our work emphasizes on semantic access to information systems and gets input OWL domain ontologies we have chosen SPARQL to be the query language of the OntoNL Framework.

SPARQL Syntax

The SPARQL is a query language for getting information from such RDF graphs. It provides facilities to:

- extract information in the form of URIs, blank nodes, plain and typed literals.
- extract RDF subgraphs.
- construct new RDF graphs based on information in the queried graphs.

A **SPARQL query** is a tuple (GP, DS, SM, R) where:

- GP is a **graph pattern**
- DS is an **RDF Dataset**
- SM is a set of **solution modifiers**
- R is a **result form**

The graph pattern of a query is called the query pattern.

A **Graph Pattern** is one of:

- **Basic Graph Pattern**, where a set of triple patterns must match
- **Group Graph Pattern**, where a set of graph patterns must all match using the same variable substitution
- **Value Constraints**, which restrict RDF terms in a solution
- **Optional Graph Pattern**, where additional patterns may extend the solution
- **Alternative**, where two or more possible patterns are tried
- **Patterns on Named Graphs**, where patterns are matched against named graphs

An **RDF Dataset** is a set:

{ G, (<u1>, G1), (<u2>, G2), . . . (<un>, Gn) }.

where G and each Gi are graphs, and each <ui> is an IRI. Each <ui> is distinct. G is called the default graph. (<ui>, Gi) are called named graphs. There may be no named graphs.

A **Solution Sequence Modifier** is one of:

- Projection modifier
- Distinct modifier
- Order modifier
- Limit modifier
- Offset modifier

The **Result Form** of a query is one of

- SELECT - Returns all, or a subset of, the variables bound in a query pattern match
- CONSTRUCT - Returns an RDF graph constructed by substituting variables in a set of triple templates
- DESCRIBE - Returns an RDF graph that describes the resources found
- ASK - Returns a boolean indicating whether a query pattern matches or not

```
PREFIX  dc: <http://purl.org/dc/elements/1.1/>

SELECT  ?title

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

Starting from the top we encounter the `PREFIX` keyword. `PREFIX` is essentially the SPARQL equivalent of declaring an XML namespace: it associates a short label with a specific URI. And, just like a namespace declaration, the label applied carries no particular meaning. It's just a label. A query can include any number of `PREFIX` statements. The label assigned to a URI can be used anywhere in a query in place of the URI itself; for example, within a triple pattern. Prefixes are syntactic: the prefix name does not affect the query, nor do prefix names in queries need to be the same prefixes as used in a serialization of the data.

In addition to the SELECT queries used in this article, SPARQL supports three other query types. ASK simply returns "yes" if the query's graph pattern has any matches in the dataset, and "no" if it does not. DESCRIBE returns a graph containing information related to the nodes matched in the graph pattern. For instance, DESCRIBE ?person WHERE { ?person foaf:name "Jon Foobar" } would return a graph containing triples from the model about Jon Foobar. Finally, CONSTRUCT is used to output a graph pattern for each query solution. This allows a new RDF graph to be created directly from the results of the query. You can think of a CONSTRUCT query on an RDF graph as somewhat analogous to an XSL transformation of XML data.

SPARQL FILTERs restrict the set of solutions according to a given expression. Specifically, FILTERs eliminate any solutions that, when substituted into the expression, result in either an effective boolean value of false or produce an error.

SPARQL provides a subset of the functions and operators defined by XQuery Operator Mapping. The following rules accommodate the differences in the data and execution models between XQuery and SPARQL:

- Unlike XPath/XQuery, SPARQL functions do not process node sequences. When interpreting the semantics of XPath functions, assume that each argument is a sequence of a single node.
- Functions invoked with an argument of the wrong type (except xsd:boolean) will produce a type error.
- Any expression other than logical or (||) or logical and (&&) that encounters an error will produce that error.
- A logical or that encounters an error on only one branch will return TRUE if the other branch is TRUE and an error if the other branch is FALSE.
- A logical and that encounters an error on only one branch will return an error if the other branch is TRUE and FALSE if the other branch is FALSE.

- A logical or or logical and that encounters errors on both branches will produce *either* of the errors.

Chapter 3

The OntoNL Framework

In this thesis we describe a software engineering framework that aims to automate as much as possible the construction of natural language interfaces to knowledge repositories. Our purpose is to provide reusable generic methodologies and software that will facilitate the generation of natural language interfaces for the interaction of the users with the knowledge repository. Since natural language interactions may involve ambiguities, our emphasis is in trying to reduce the ambiguities as much as possible. We are building a system that supports this architectural framework.

The visual representation of the framework is shown in the following figure.

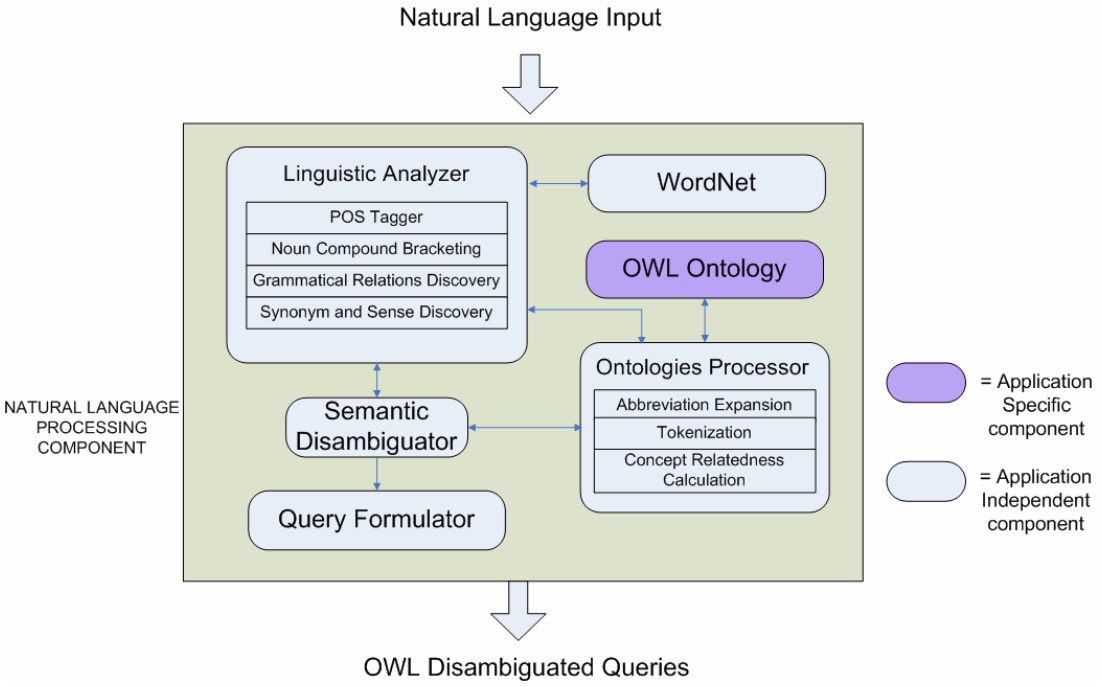


Figure 3: The OntoNL Framework

In figure 3 we see the two steps of interaction using the OntoNL system,described by a model architecture with additional information, the components of software that comprise the framework modules. We are now going to provide short descriptions of the components and then their interaction.

Natural Language Processing Component

The Natural Language Processing Component provides the mechanisms for inserting a user query either from different front-ends either from a system that received in some way natural language etc., and concludes to disambiguated queries based on the OWL ontological structures it used for the disambiguation. It consists of five sub-components that are responsible for parsing the natural language expression and after that to syntactically and semantically disambiguate it in a particular domain. Each time, the domain is defined by a particular ontology, as shown in figure 1.

The Natural Language Processing Component is comprised by a component responsible for the linguistic analysis of a user expression in English (Linguistic Analyzer), a component for the semantic disambiguation based on the application's domain (Semantic Disambiguator), a component for the processing of the ontological structures and semantics (Ontologies Processor) and a component for the reformulation of the disambiguated user expression to a query language (Query Reformulator).

The linguistic analysis component is responsible for the production of a language model with information concerning the syntax and the semantics of the components of an utterance in the domain of English language. Name semantics are captured by part-of-speech tagging, meaningless word filtering, synonym and sense discovery by using a thesaurus, noun compound bracketing mechanisms enhanced with information that comes from the ontologies and grammatical relationship annotation for capturing the role of each word grammatically inside the user utterance.

Chapter 3

The semantic disambiguation architecture is based on the information retrieved from the ontology that defines the domain. It follows a procedure consisting of different manipulation of the ontology based on different levels of ambiguity.

The semantic similarity clustering of the ontologies, which is a responsibility of the Ontologies Processor, that describe a domain is based on a methodology for weighting. The methodology takes advantage of the rich semantic information that can be extracted from OWL ontologies, and classic measures for measuring the strength of the relation between concepts in a graph. The Ontologies Processor has also the role to bridge the gap between the terminology used by programmers of the ontology and the natural language, by using mechanisms as word tokenization and abbreviation expansion.

The procedure for dealing with ambiguities has different mechanisms that refer to the degree and type of disambiguation. Three examples of disambiguation follow:

- Request Type 1
    - "players of soccer team Barcelona"
- Request Type 2
    - "players of Barcelona" (Barcelona = city, soccer team, etc.)
- Request Type 3
    - "mvp of Barcelona versus Real" (mvp = ?, Barcelona = city?, soccer team? .., Real = soccer team?, person? ..)

In the request type 1, ambiguities can be resolved by using knowledge from the reference domain ontology. Words like "players" and "soccer team" can be found as concepts in the ontology.

In the request type 2 there is a greater value of ambiguity since we cannot assign the keyword "Barcelona" to any concept. The way of dealing with such an ambiguity, requires a more systematic way of searching for specific and strongly related clusters of context inside the ontology to make more domain specific the disambiguation.

In the request type 3 we are dealing with the fact that the response contains words that are either concept instances or simple words without knowing the concept they refer to (information that was not retrieved from the the ontology). In this difficult case we cannot help the disambiguation processs.

To summarize the interactions concerning the OntoNL framework:

The NLP Component receives a natural language expression. It communicates with the WordNet ontology that helps the syntactic disambiguation. After completing the syntactic disambiguation it consults the ontological structures and the relatedness weight values extracted from the processing of the ontology for the semantic disambiguation. Finally, it constructs queries in an ontology query language with the disambiguated information.

Summary

In this chapter we introduced the OntoNL Framework by showing the basic software components and how they are interacting. In the two following chapters we are going to present the two basic models of the framework; the model for the syntactic and the model for the semantic disambiguation of the natural language user input.

Chapter 4

Syntactic Disambiguation in OntoNL

Ascertaining what is intended in a text when more than one interpretation is possible has always been a central issue in natural language processing: ambiguity resolution is required whenever the system must choose among two or more distinct representations of the input. Ambiguity pervades virtually all aspects of language analysis, and sentence analysis in particular exhibits a large number of syntactic, semantic, and pragmatic ambiguities that demand adequate resolution before the sentence can be understood. The OntoNL framework is designed to acquire solutions to all lexical and structural ambiguity problems encountered during sentence analysis.

For the needs of the sentence analysis, we have created a software component, the linguistic analyzer that follows a procedure with mechanisms for the syntactic disambiguation on the user's utterance. The mechanisms concern the part-of-speech tagging procedure, the noun compound bracketing procedure, the grammatical relation annotation, the sense and synonyms discovery procedure that conclude to a language model that describes the structure of the utterance after the syntactic analysis.

In the special occasion where the OntoNL is applied in a question answering system the sentences that a user uses are requests or questions (Wh-questions; questions that start with the words What, Where, Why, Which, When or How). Requests do not contain the actual information to address the knowledge repository in the subject of the sentence, but in one or more dependent clauses that complement the independent clause to a complex sentence. For example, in a domain concerning football, a user query could be:

- I want you to find me the players that scored for Barcelona in the last two football games that used to play for Milan.

In this case, the actual 'subject' of the request is not the one in the independent clause 'I want you to find me the players…', but the object of the independent clause is actually the 'subject' of interest of the user's request ('players'). We need, therefore to identify what the user asks the system and the complements-constraints that (s)he gives for this 'subject'.

In the next sub-sections we describe the mechanisms that the parser uses to disambiguate syntactically and semantically English language interactions with the knowledge repositories.

The OntoNL Request Conversion Mechanism

For the request conversion mechanism, there is an effort to eliminate the first words that a request or a question may have because, the words that do not contain semantics for the retrieval of information from a repository. We distinguish 3 different types of requests and questions in which we have a different approach when dealing with the information of the utterance:

a. Requests for metadata (ex. Show me the goals scored in the game between Italy and France)

b. WH-questions (ex. What was the score in the game between Italy and France?)

In the literature we find that the wh-questions testify the subject of the request according to the type of input. We present in what follows the semantics of request of each different type of wh-question.

When?	Time
Where?	Place
Who?	Person
Why?	Reason
How?	Manner
What?	Object/Idea/Action
Which (one)?	Choice of alternatives
Whose?	Possession
Whom?	Person (objective formal)

How much?	Price, amount (non-count)
How many?	Quantity (count)
How long?	Duration
How often?	Frequency
How far?	Distance
What kind (of)?	Description

c. Yes/No questions (ex. Were there any goals in the game between Italy and France?)

In the input conversion mechanism we identify the type of the input and we use an indicator to distinguish the three different types. After the conversion, the input becomes:

1. the goals scored in the game between Italy and France
2. the score in the game between Italy and France
3. any goals in the game between Italy and France

The grammatical dependencies (subject, object, verb, complements) of those converted sentences have the semantics that we retrieve by interacting with the application-domain ontology. This information enhances the OntoNL Language Model.

The OntoNL Part-Of-Speech Tagger

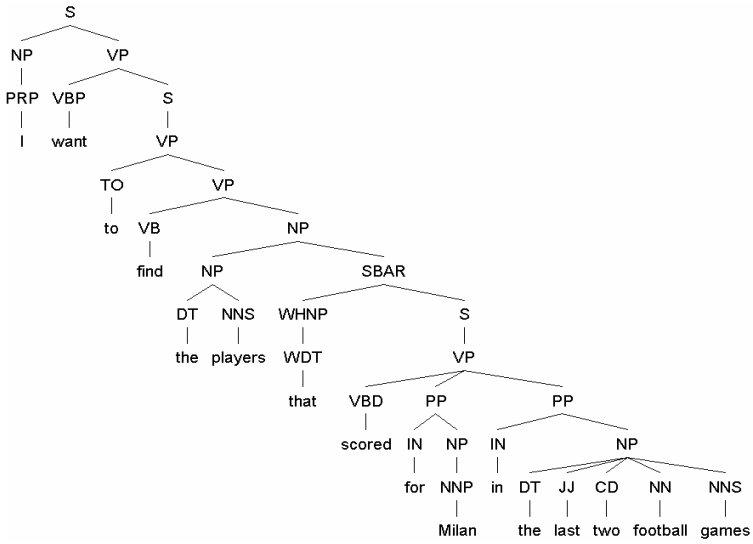
The tagset that OntoNL used for the representation of the part-of-speech tags was a subset of the Penn Treebank Tagset (table 1).

1	Coordinating conjunction	CC
2	Cardinal number	CD
3	Determiner	DT
4	Existential there	EX
5	Foreign word	FW
6	Preposition or subordinating conjunction	IN
7	Adjective	JJ
8	Adjective, comparative	JJR
9	Adjective, superlative	JJS
10	List item marker	LS
11	Modal	MD

12	Noun, singular or mass	NN
13	Noun, plural	NNS
14	Proper noun, singular	NNP
15	Proper noun, plural	NPS
16	Predeterminer	PDT
17	Possessive ending	POS
18	Personal pronoun	PP
19	Possessive pronoun	PP$
20	Adverb	RB
21	Adverb, comparative	RBR
22	Adverb, superlative	RBS
23	Particle	RP
24	Symbol	SYM
25	to	TO
26	Interjection	UH
27	Verb, base form	VB
28	Verb, phrase	VP
29	Verb, past tense	VBD
30	Verb, gerund or present participle	VBG
31	Verb, past participle	VBN
32	Verb, non-3rd person singular present	VBP
33	Verb, 3rd person singular present	VBZ
34	Wh-determiner	WDT
35	Wh-pronoun	WP
36	Possessive wh-pronoun	WP$
37	Wh-adverb	WRB
38	Wh-Noun Phrase	WHNP

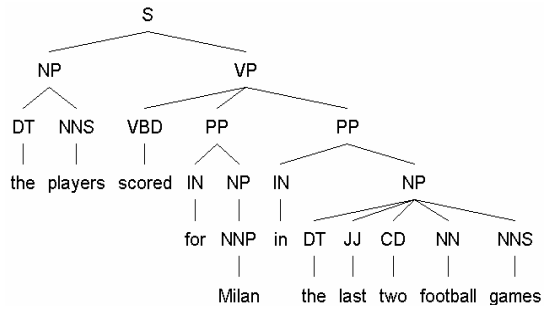
Table 1: The Penn Treebank Tagset

For the OntoNL part of speech tagging procedure we used the Stanford Log-Linear Part-Of-Speech Tagger (http://nlp.stanford.edu/software/tagger.shtml) with the tagset defined in table 1. An example of the results coming from the POS-tagger for a request of a user is shown in Figure 4. In Figure 5 there is an interpretation for the system of the core information coming from user request. We isolate the object part of the user request and transformed it in a new sentence with a subject and a verb phrase. The examples show that for better results we must isolate and translate the user request to a simple sentence and then try to find any grammatical relations in it.



NP	Noun Phrase
VP	Verb Phrase
PRP	Personal Pronoun
VBP	Non-3rd person singular present tense verb
DT	Determiner
NNS	Plural Noun
VB	Verb Base form
WHNP	Wh-Noun Phrase
WDT	Wh-Determiner
VBD	Verb, Past Tense
IN	Preposition
JJ	Adjective
CD	Cardinal Number
NN	Singular Noun
NNP	Singular Proper Noun

Figure 4: The result of the POS – Tagging [http://nlp.stanford.edu/software/tagger.shtml] procedure for the sentence 'I want to find the players that scored for Milan in the last two football games'



NP	Noun Phrase
VP	Verb Phrase
PRP	Personal Pronoun
VBP	Non-3rd person singular present tense verb
DT	Determiner
NNS	Plural Noun
VB	Verb Base form
VBD	Verb, Past Tense
IN	Preposition
JJ	Adjective
CD	Cardinal Number
NN	Singular Noun
PP	Prepositional Phrase

Figure 5: The result of the POS – Tagging [http://nlp.stanford.edu/software/tagger.shtml] procedure for the sentence the players scored for Milan in the last two football games'

The OntoNL Noun Compound Analyzer

During the process of parsing natural language expressions, there are many linguistic matters that must be taken into account. For instance, parsing noun compounds (NCs) requires detailed world knowledge that is unavailable outside a limited domain [Sparck

Jones, 1983]. There are many corpus-based algorithms (at least four) proposed for syntactically analyzing noun compounds. Three of the algorithms use the adjacency model and the fourth algorithm uses the dependency model [Lauer, 1995]. The procedure that the adjacency model follows it is reproduced here for reference [Marcus, 1980]:

Given three nouns n_1, n_2 and n_3:

- if either [n_1 n_2] or [n_2 n_3] is not semantically acceptable then build the alternative structure;
- otherwise, if [n_2 n_3] is semantically preferable to [n_1 n_2] then build [n_2 n_3].
- otherwise, build [n_1 n_2].

The dependency model utilizes the following procedure when given three nouns n_1, n_2 and n_3:

- determine how acceptable the structures [n_1 n_2] and [n_1 n_3] are;
- if the latter is more acceptable, build [n_1 n_3] first;
- otherwise, build [n_1 n_2] first.

Adjacency and dependency models examine the left or right bracketing in a 3-word noun compound, as for example '[[soccer team] shirt]'. A well known work on automated unsupervised NC bracketing is that of Lauer [Lauer, 1995] who introduces the probabilistic dependency model for the syntactic disambiguation of NCs and argues against the adjacency model [Marcus, 1980], [Pustejovsky et al., 1993] [Resnik, 1993]. Recent experimental results along with experiments done in the OntoNL environment show that the dependency model performs better (see chapter 7).

Recent research on the field of noun compound bracketing [Keller and Lapata, 2003], [Nakov and Hearst, 2005] uses of the Web search engines for measuring page hits for more accurate results. However, the use of Web search engines can impose limitations, because of the lack of linguistic annotation, such as the use of a word as a particular part-of-speech (noun compound refers to nouns, but there are nouns that can be found as verbs in documents, also) and the ignorance of punctuation characters, hyphens and possessive markers.

The problem with applying lexical association to noun compounds is the enormous number of parameters required one for every possible pair of nouns. This leads to the need of a vast amount of memory space and to the severe data sparseness problem. Resnik [Resnik, 1993] introduced the term conceptual association to refer to association values computed between groups of words. So, by having words organized in groups with similar behavior, the parameter space can be built in terms of the groups. Resnik computed the groups of words by calculating a measure of association using the classes to which the direct object and object of the preposition belong, and by selecting the attachment site for which the evidence of association was strongest. The use of classes introduced two sources of ambiguity. The first was word sense ambiguity: just as lexically based methods conflate multiple senses of a word into the count of a single token, each word may be mapped to many different classes in the WordNet taxonomy. Second, even for a single sense, a word may be classified at many levels of abstraction -- for example, even interpreted solely as a physical object (ratber than a monetary *unit), penny* may be categorized as a (coin, 3566679), (cash,3566144), (money, 3565439), and so forth on up to (possession, 11572) [Resnik and Hearst, 1993].

In OntoNL when dealing with noun compounds we first use a method to expand n-grams into all morphological forms by using morphological tools [Minnen, et. al., 2001]. For example, if we have a bigram 'player scores', then we create a list of all possible forms: 'player scores', 'player score', 'players score', etc. Also, based on the successful performance of the dependency model over the adjacency [Lauer, 1995] we adopt the use of it.

In OntoNL, conceptual association is used with groups consisting of all categories from the *Roget's II: The New Thesaurus* (http://www.bartleby.com/62/) (254 categories). Given two categories t_1 and t_2 there is a parameter $P(t_1 \rightarrow t_2)$ that represents the degree of acceptability of the structure $[n_1\ n_2]$ where n_1 is a noun appearing in t_1 and n_2 appears in t_2. The event $t_1 \rightarrow t_2$ declares the modification of a noun in t_2 by a noun in t_1.

In OntoNL we used a window to collect training instances by observing how often a pair of nouns co-occurs within some fixed number of words. For window size $n \geq 2$, let

$count_n(n_1, n_2)$ be the number of times a sequence $n_1w_1 \ldots w_in_2$ occurs in the training corpus where $i \le n-2$. The estimates are:

$$P(t_1 \to t_2) = \frac{1}{\sum_{w_1 \in N, w_2 \in t_2} \frac{count_n(w_1, w_2)}{amb(w_1, w_2)}} \sum_{w_1 \in t_1, w_2 \in t_2} \frac{count_n(w_1, w_2)}{amb(w_1) amb(w_2)}$$

where *amb*(*w*) counts the number of categories w appears and N is a set of words that can only be used as nouns. The *amb*(*w*) has the effect of dividing the evidence from a training instance across all possible categories for the words. The first parameter of the multiplication is used to ensure that the parameters for a head noun sum to unity. After the calculation of the estimates, we continue by trying to make a right choice of all possible analyses for three word compounds, which are the counting of a right or a left branching analysis. If the ratio is >1 then we conclude to a left-branching analysis. If it is <1 then a right branching analysis is chosen. If it is =1, the OntoNL analyzer, based on Lauer (Lauer, 1995) guesses left-branching, a rare case for conceptual association based on experimental results.

So, for the dependency model and a given compound of w_1, w_2, w_3 the estimation of the ratio is done by applying the equation

$$R_{dep} = \frac{\sum_{t_i \in cats(w_i)} P(t_1 \to t_2) P(t_2 \to t_3)}{\sum_{t_i \in cats(w_i)} P(t_1 \to t_3) P(t_2 \to t_3)}$$

where t_1, t_2 and t_3 are conceptual categories in a taxonomy or thesaurus, and the nouns $w_1, \ldots, w_n$ are members of these categories. For a correct result we must sum over all possible categories for the words in the compound. In any case, the estimation of probabilities over concepts reduces the number of model parameters.

The innovation in this work is the training set of the noun compound bracketing procedure. The corpus is provided by the domain ontologies the OntoNL uses for disambiguation and it consists of the concept names that are compound nouns, after the abbreviation expansion and the tokenization process that will be described in Chapter 5,

their synonyms from the WordNet, the <owl:label> content and the <owl:comment> content that are the tags were the ontology developer can specify in more details the semantics of the concept that is described. This may lead to the conclusion that the training set is very limited in comparison to a linguistic corpus, but it is more specific to the needs of the application. We are interested in the particular needs of the user based on a specific domain. By combining the use of domain ontologies as the training corpus we maintain all the information. Since we use the noun compound bracketing methodology to be more accurate when dealing with the user request's ambiguities we use as a test set the noun compounds that are may appear in the exact user request and as a training corpus the total of domain ontologies used. The results of the proposed methodology for the noun compound bracketing will be presented in Chapter 7.

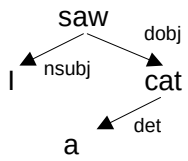
In the case were the domain ontologies do not include descriptions we cannot train the OntoNL noun compound bracketing module and what we do is to search into the domain ontological structures for all the expanded n-grams to find a match.

The procedure of the noun compound bracketing is useful in determining correctly the grammatical relationships that structure the language model. We tested the correctness of the OntoNL Syntactic Disambiguation without dealing compound nouns and the annotation was wrong especially when the system tried to locate grammatical dependencies of the object to the subject.

The OntoNL Grammatical Relations Analyzer

Grammatical relationships are an important aspect of natural language processing. One of the main goals of an interpreter is to map the syntactic descriptions found in the sentence into the correct roles that the elements, described by the nominals (a word or a group of words that functions as a noun, i.e. a word or a group of words that can stand at the head of a noun phrase), play in the situation at hand (described by the verb). We describe a model for automatically extracting typed dependency parses of English sentences from phrase structure parses. Typed dependencies and phrase structures are different ways of

representing the structure of sentences: while a phrase structure parse represents nesting of multi-word constituents, a typed dependency parse represents dependencies between individual words. A typed dependency parse additionally labels dependencies with grammatical relations, such as subject or indirect object. For example the sentence *I saw a cat* becomes:



There has been much linguistic discussion of the two formalisms. There are formal isomorphisms between certain structures, such as between dependency grammars and one bar-level, headed phrase structure grammars [Miller, 2000].

In more complex theories there is significant debate: dominant Chomskyan theories [Chomsky, 1981] have defined grammatical relations as configurations at phrase structure, while other theories such as Lexical-Functional Grammar has rejected the adequacy of such an approach [Bresnan, 2001].

Recent years have seen the introduction of a number of treebank-trained (treebanks are language resources that provide annotations of natural languages - the main point with treebanks are that that they consist of tree representations of entities in language) statistical parsers (Collins [Collins, 1999], Charniak [Charniak, 2000], Stanford [Klein and Manning, 2003]) capable of generating parses with high accuracy.

The original treebanks, in particular the Penn Treebank (The Penn Treebank Project annotates naturally-occuring text for linguistic structure) (http://www.cis.upenn.edu/~treebank/), were for English, and provided only phrase structure trees, and hence this is the native output format of these parsers.

At the same time, there has been increasing interest in using dependency parses for a range of NLP tasks, from machine translation to question answering. Such applications benefit particularly from having access to dependencies between words typed with grammatical relations, since these provide information about predicate-argument structure

which are not readily available from phrase structure parses. Perhaps partly as a consequence of this, several more recent treebanks have adopted dependency representation as their primary annotation format, even if a conversion to a phrase structure tree form is also provided.

Grammatical relationships will help to create an appropriate language model that will lead the system to retrieve the right data from the repository. They are the semantic basis for the information extraction role of the system. It can further help in conducting proper answers filled with repository data to present to the user.

Concentrating on those elements that are normally obligatory and based on the book *A University Grammar of English* [Quirk et. al, 1973], we can usefully distinguish eight general clause types:

- **S – V** Subject – Verb
- **S – V – SC** Subject – Verb – Subject Complement
- **S – V – DO** Subject – Verb – Direct Object
- **S – V – O – Adv** Subject – Verb – Object – Adverb
- **S – V – DO – OC** Subject – Verb – Direct Object – Object Complement
- **S – V – IO – DO** Subject – Verb – Indirect Object – Direct Object
- **S – V – Adj** Subject – Verb – Adjective
- **S – V – Adv** Subject – Verb – Adverb

where:

Subject of a sentence is that noun, pronoun, or phrase or clause about which the sentence makes a statement.

Verb phrase or main verb of a sentence is a word or words that express an action, event, or a state of existence. It sets up a relationship between the subject and the rest of the sentence.

Subject complement is that noun, pronoun, adjective, phrase, or clause that comes after a linking verb (some form of the *be* verb)

Direct object is a noun, pronoun, phrase or clause acting as a noun and takes the action of the main verb. A direct object can be identified by putting *what?*, *which?* or *whom?* in its place.

Indirect object is a noun, pronoun, phrase or clause acting as a noun and receives the action expressed in the sentence. It can be identified by inserting *to* or *for*.
Object complement is a noun or a adjective coming after a direct object and adds detail to the direct object. It can be identified by inserting [*to be*] between the direct and the object complement.

The grammar of English is comprised by a large set of rules that is difficult to model. For the needs of the OntoNL Framework we modeled and completely defined a subset of those rules that concern specific grammatical types that are presented in table 2. Our main goal was to create an annotation scheme for locating grammatical relations from scratch and not use a scheme from literature. The reason for that was that the grammatical relation definition in OntoNL has the goal to help the disambiguation of the user expression and not to consume the design phase in orded to provide a complicated model for computational linguistics that would give us extra complex and in the end useless information for our cause.

For the definition of grammatical relations, the goal of the OntoNL Framework is more practical than following a Chomskyan theory or a Lexical-Functional Grammar as they were presented earlier, though in essence we are following an approach where structural configurations are used to define grammatical roles.

We used as a starting point the set of grammatical relations defined in (Carroll et al., 1999). The motivation for the creation of a new annotation scheme for locating grammatical relations was that while the backbone of the hierarchy is quite similar to that in (Carroll et al., 1999), over time we have introduced a number of extensions and refinements to facilitate use in applications. Many NP(noun phrases)-internal relations play a very minor role in theoretically motivated frameworks, but are an inherent part of corpus texts and can be critical in real-world applications. Therefore, Carroll distinguishes the three modifiers, cmod, xmod and ncmod that stand for the adjuncts and non clausal modifiers respectively, but we define 5 new relations useful for natural language expressions used by plain users. Also, besides the commonest grammatical relations for NPs, our hierarchy includes the following grammatical relations: conj (conjunction) nn (noun compound), num (numeric modifier), agent (agent) and empty (empty).

Name	Description	Examples *[source]→[target]* in text
head	head ▪ Head of the parsed tree ▪ Head determines the type of the clause (noun phrase, verb phrase...)	[lives] in "Marisa lives in Rome" [gift] in "the gift of a book" [Picasso] in "Picasso the painter"
subj	subject ▪ subject of a verb ▪ link a copula subject and object ▪ link a state with the item in that state ▪ link a place with the item moving to or from that place	[I]→[promised] in "I promised to help" [I]→[to help] in "I promised to help" [the cat] → [ran] in "the cat that ran" [You] → [happy] in "You are happy" [You] → [a runner] in "You are a runner" [you] → [happy] in "They made you happy" [I] → [home] in "I went home"
dep	dependent	The remaining parts of the sentence without the head
arg	argument	Subject and complements
predicate	A predicate is the portion of a clause, excluding the subject, that expresses something about the subject	[is on the table] → [the book] in "The book is on the table"
obj	object ▪ object of a verb ▪ object of an adjective ▪ surface subject in passives ▪ object of preposition not for partitives or subsets ▪ object of an adverbial clause complementizer	[saw] → [the cat] in "I saw the cat" [promised] [to help] in "I promised to help you" [happy] → [to help] in "I was happy to help" [I] → [was seen] in "I was seen by a cat" [by] → [the tree] in "I was by the tree" [After] → [left] in "After I left, I ate"
dobj	direct object ▪ a noun or pronoun that receives the action of a transitive verb in an active sentence	[burnt] → [the toast] in "Terry burnt the toast" [visited] → [Kara] in "Serena visited Kara"
iobj	indirect object ▪ verbs that involve giving something to someone or making something for someone. ▪ are usually placed directly before the direct object. They	[gave] → [you] in "I gave you a cake" [offered] → Jim in "He offered him a job"

	usually answer the questions "to what/whom?" or "for what/whom?"	
obj2	second object ▪ a noun or pronoun that comes after the direct object in di-transitive constructions	[give] → [present] in "Give Mary a present." [mailed] → [contract] in "He mailed John the contract"
objcomp	object complement ▪ with some transitive verbs, the direct object can be followed by another noun or modifying phrase called an object complement**.**	[elected] → [president] in "The students elected him president"
empty	use instead of "subj" relation when subject is an expletive (existential) "it" or "there"	[There] → [trees] in "There are trees"
clausal	subordinating conjunctions (as, since, because...) or relative pronouns (who, which, that) usually introduce dependent clauses	[name] → [of] in "name of the building" [was seen] → [by] in "I was seen by a cat"
comp	complement ▪ between a subject and a clausal	[age] → [of] in "age of 12" [the attack] → [on] in "the attack on the base"
agent	agent ▪ the agent performs the action in passive voice	[the dog] → [the boy] in "The boy was bitten by the dog"
mod	generic modifier (use when modifier does not fit in a case below)	[the cat]→[ran] in "the cat that ran" [ran]→[with] in "I ran with new shoes"
mod-nn	noun compound modifier	[cost-of-living] → [adjustment] "The cost of living went up, but he didn't receive a cost-of-living adjustment"
mod-det	determiner	[ate]→[at] in "I ate at home"
mod-tmod	temporal modifier	[yesterday] in "Yesterday, Hillary told him to leave the house"
mod-prep	prepositional modifier	[on] → [cafeteria] in "on the cafeteria"
mod-num	numeric modifier	[hundreds]→[people] in "hundreds of people"
conj	conjunction ▪ used to annotate the type of conjunction and the heads of the conjuncts	[and] → [player, coach, member] in "the players, the coaches and the other members of the team" [or] → [smile, laugh] in "John smiled or Susan laughed"

Table 2: Definition of the OntoNL Grammatical Types

The grammatical relations are arranged in a hierarchy (figure 6), rooted with the most generic relation, **dep** (dependent). We use a rich set of grammatical relations that provide detailed information about syntactic relationships. When the relation between a head and its dependent can be identified more precisely, relations further down in the hierarchy can be used. For example, the dependent relation can be specialized to arg (argument), mod (modifier) or conj (conjuction). The arg relation is further divided into the subj (subject) relation and the comp (complement) relation, and so on. The definition of the OntoNL hierarchy follows.

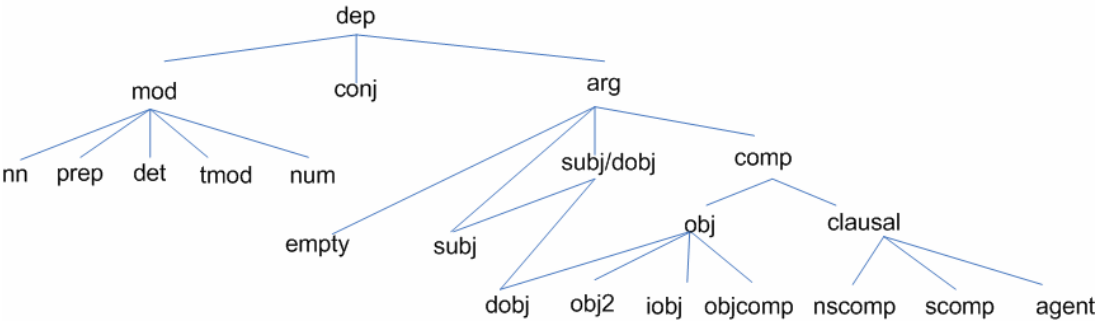


figure 6: The OntoNL grammatical relation hierarchy

-dep(introducer, head, dependent)

The most generic relation

-arg(head, dependent)

The relation between a head and an argument

-subj/dobj(head, dependent)

A specialization of the relation **arg,** which can instantiate either subjects or direct objects.

-subj (head, dependent)

The relation between a predicate and its subject; where appropriate. The **initial_gr** indicates the syntactic link between the predicate and subject before any GR-changing process.

subj(play, Vazeha) Vazeha plays in Panathinaikos

subj(employ, Paul) Paul was employed by IBM

-comp(head, dependent)

The most generic relation between a head and complement.

-obj(head, dependent)

The most generic relation between a head and object.

-dobj(head, dependent, initial_gr)

The relation between a predicate and its direct object – the first non-clausal complement following the predicate which is not introduced by a preposition.

dobj(play, ball) plays ball

dobj(give, George) give George the ball

-iobj(type, head, dependent)

The relation between a predicate and a non-clausal complement introduced by a preposition. The type indicates the preposition introducing the dependent.

iobj(in, play, Barcelona) play in Barcelona

-obj2(head, dependent)

The relation between a predicate and the second non-clausal complement in ditransitive constructions.

obj2(give, ball) give George the ball

-objcomp(head, dependent)

The relation between a predicate and a non-clausal complement introduced by a direct object

objcomp(dyed, blonde) dyed his hair blonde.

-clausal(head, dependent)

The most generic relation between a head and a clausal complement.

-nscomp(type, head, dependent)

The relation between a predicate and a clausal complement which has no overt subject. The type slot indicates the complementiser/preposition, if any, introducing the dependent.

nscomp(in, be, Paris) Mary is in Paris

-scomp(type, head, dependent)

The relation between a predicate and a clausal complement which has an overt subject. The type slot indicates the complementiser/preposition, if any, introducing the dependent.

scomp(that, say, leave) I said that he left

-agent(head, dependent)

The thematic relation of a situation that carries out the action in this situation.

agent(submitted, by, Harry) It was submitted by Harry

-conj

The conjunction between two predicates.

-mod(head, dependent)

Modifier.

-mod-nn(head, dependent)

The noun compound modifier.

-mod-det(head, dependent)

Determiner.

-mod-prep(head, dependent)

The prepositional modifier

-tmod(head, dependent)

The temporal modifier.

-mod-num (head, dependent)

The numeric modifier. Qualifies a number that serves to modify the meaning of a NP

num(sheep, 3) Sam ate 3 sheep

The technique for producing typed dependencies is essentially based on rules – or patterns – applied on phrase structure trees. The method is general, but requires appropriate rules for each language and treebank representation. Here we present details only for Penn Treebank English. The method for generating typed dependencies has two phases: **dependency extraction** and **dependency typing** [MacCartney et. al, 2006]. In the dependency extraction phase first, a sentence is parsed with a pos-tagger. Any Penn

Treebank tagger (trained on the Penn Treebank) could be used for the process described here, but in practice we are using the Stanford Tagger, trained on the Penn Wall Street Journal Treebank. The head of each constituent of the sentence is then identified, using rules according to the Collins head rules [Collins, 1999], but modified to retrieve the semantic head of the constituent rather than the syntactic head.

While heads chosen for phrase structure parsing do not really matter, retrieving sensible heads is crucial for extracting semantically appropriate dependencies. For example, in relative clauses, the Collins rule will choose as head the pronoun introducing the relative clause. As all the other words in the relative clause will depend on the head, it makes more sense to choose the verb as head when determining dependencies. In general, we prefer content words as heads, and have auxiliaries, complementizers, etc. be dependents of them. Another example concerns NPs with ambiguous structure or multiple heads which are annotated with a flat structure in the Penn Treebank:

(NP the new phone book and tour guide)

Using the Collins rule, the head for this example is the word “guide”, and all the words in the NP depend on it. In order to find semantically relevant dependencies, we need to identify two heads, “book” and “guide”. We will then get the right dependencies (the noun “book” still has primacy as a governing verb will link to it, but this seems reasonable):

nn(book, phone)

nn(guide, tour)

CC and(book, guide)

mod(book, new)

det(book, the)

It is essential in such cases to determine heads that will enable us to find the correct dependencies.

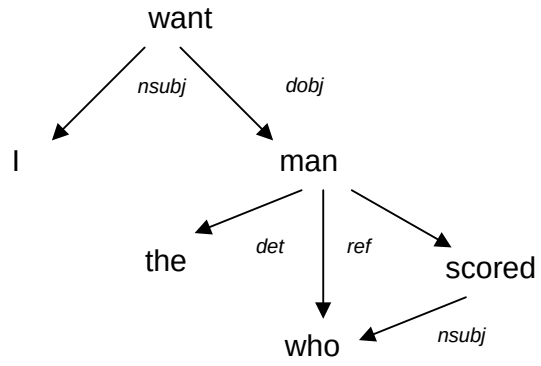


Figure 7: An example of a typed dependency parse for the sentence "I want the man who scored"

In the second phase, we label each of the dependencies extracted with a grammatical relation which is as specific as possible. For each grammatical relation, we define one or more patterns over the phrase structure parse tree (using the tree-expression syntax defined by tregex [Levy and Andrew, 2006]. Conceptually, each pattern is matched against every tree node, and the matching pattern with the most specific grammatical relation is taken as the type of the dependency (in practice, some optimizations are used to prune the search).

The OntoNL Synonyms and Sense Discoverer

An implementation framework for constructing and using natural language interfaces cannot have previous knowledge about the structure and the information in a knowledge repository. The repository has a specific way, based on a schema, to represent its contents. This fact leads to the need of generalizing the words that a user may include in a request to the set of synonyms that describe the sense of the word. The synonyms that correspond to each sense of a word can be extracted by a thesaurus or a word ontology.

Many researchers have used WordNet [Miller, 1990] in information retrieval as a tool for query expansion [Voorhees, 1994], [Smeaton and Berrut, 1995], computing lexical cohesion [Stairmand, 1997], word sense disambiguation [Voorhees, 1993], and so on. In

WordNet, words are organized into taxonomies where each node is a set of synonyms (a synset) representing a single sense. The logical structure of WordNet is shown in figure 7.

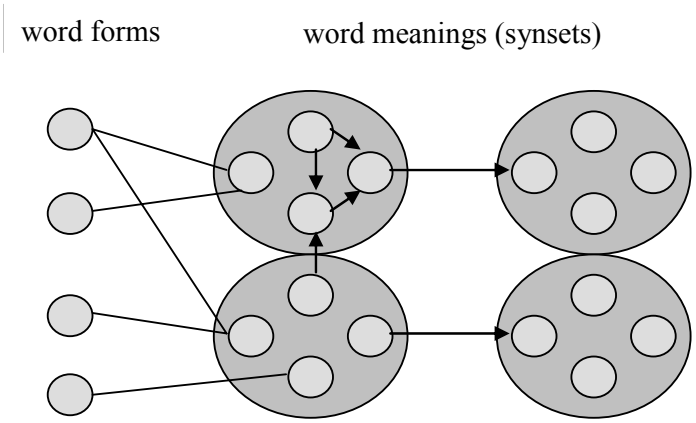


Figure 8: Logical Structure of WordNet.

Word meanings are associated with word forms that can express them. We can see on the figure that the relation between word forms and word meanings is m to n—word form can have many meanings, and many word forms can refer to the same meaning. The former phenomenon is called polysemy, the latter is called synonymy. Dealing with such an ambiguity of natural language is the key challenge in automated natural language processing. Each word meaning entry (also called synonym set, or synset), is accompanied with on (called gloss), and list of word forms that can represent the synset in spoken or written language.

There are 4 different taxonomies based on distinct parts of speech (nouns, verbs, adjectives, adverbs) and many relationships defined within each. For the synonym and sense discovery we use only noun taxonomy with hyponymy/hypernymy (or is-a) relations, which relates more general and more specific senses.

The similarity between word w_1 and w_2 in the WordNet taxonomy is defined as the shortest path from each sense of w_1 to each sense of w_2, as below [Resnik, 1995]

$$sim_{path}(w1, w2) = \max[-\log\left(\frac{N_p}{2D}\right)]$$

where N_p is the number of nodes in path p from w_1 to w_2 and D is the maximum depth of the taxonomy.

We use in the OntoNL the WordNet for synonym and sense discovery for the linguistic analysis and we use the taxonomy of nouns. It is one of our research issues to find ways to efficiently use other taxonomies, and especially verbs and adjectives.

The OntoNL Language Model

Following the methodology and the algorithms described in the syntactic analysis procedure the natural language parser concludes to a language model described in Figure 9 and Figure 10. The language model is described using a UML class diagram. Figure 9 shows the structure that contains the words that constitute the subject, the action verb and the object. The object from the user request contains the actual information to be retrieved from the repository. In conjunction with complement information from dependent clauses there can be a transformation to a new sentence without loosing information, since there is a specific grammar for response expressions, where the object will play the part of the subject. Then, a second structure is produced, using a model diagram seen in Figure 10 with information coming from all steps during the linguistic disambiguation.

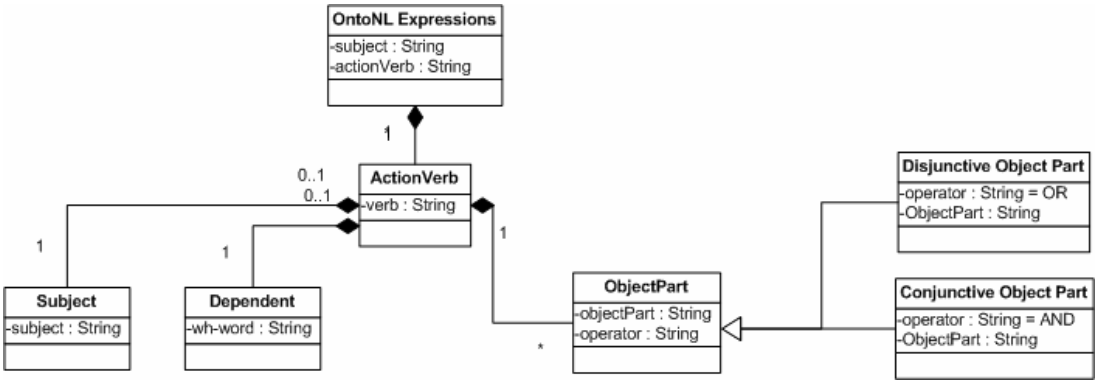


Figure 9: The syntactic structure of the independent clause

This structure contains information that helps the process of information retrieval with the subject and its complements, the object and its complements and for each word, its synsets and hyponyms from the WordNet.

In this model diagram there are classes representing the grammatical relations that are connected with associations. What we see is that there are lists of words that constitute the basic sentence structures, like the subject and the object and there are complements and special cases of objects that predicate them. The OntoNL Expressions is the general class that summarizes the cases of possible grammatical dependencies inside an utterance. It consists of a Subject Part and possibly of a Verb Phrase Group. The Subject Part consists of one or more Simple Subjects that are connected with a BooleanOr operator and the Subject Complement Class. Certain intransitive verbs, called linking verbs, connect a **subject complement** to the subject. These complements complete the meaning of the subject. Some **linking verbs** are: (appear, become, seem, feel, grow, act, look, taste, smell, sound, get, be (in all its forms). The Verb Phrase Group is an abstract class and has an IS-A relation with the Conjunctive and the Disjunctive Verb Phrase Group. By this way we distinguish the cases where the Subject Part can be described by more than one verb phrases that are separated by Boolean Operators. The Verb Phrase Group consists of one or more Verbs. The Verb consists of a unique Object Part, an abstract class of one or more Objects. Again the Object Part has an IS-A relation with the Conjunctive and the Disjunctive Object Part. That means that after the verb we can meet more than one object parts joint with a Boolean AND or OR operator. The Object can be a Direct Object, an Indirect Object, an Object Complement or a combination of these.

An example of the information contained in the language model for the sentence “The player with shirt number 9 gave Milan the victory” is shown in figure 11.

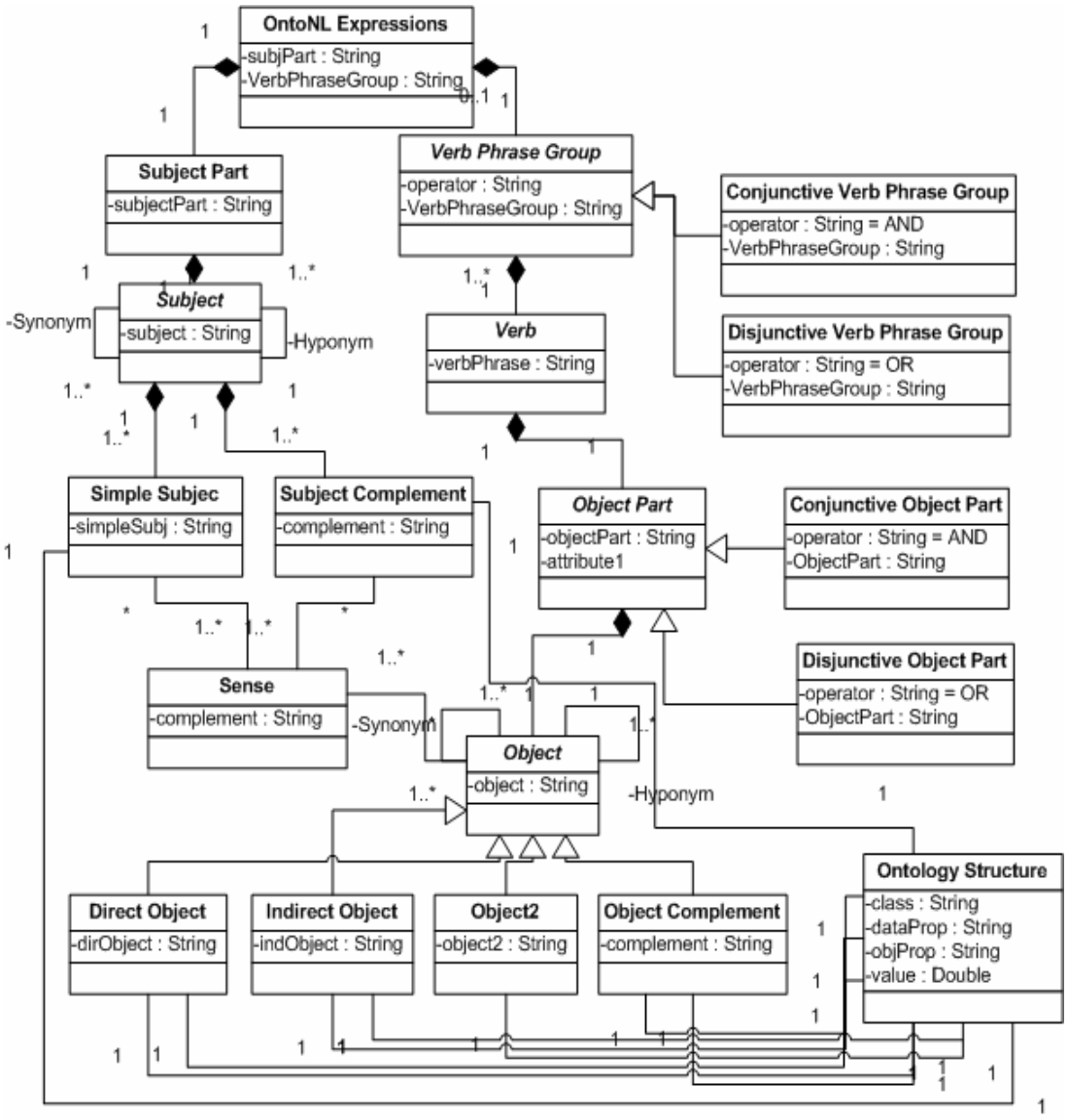


Figure 10: The language model that derives from the linguistic analysis

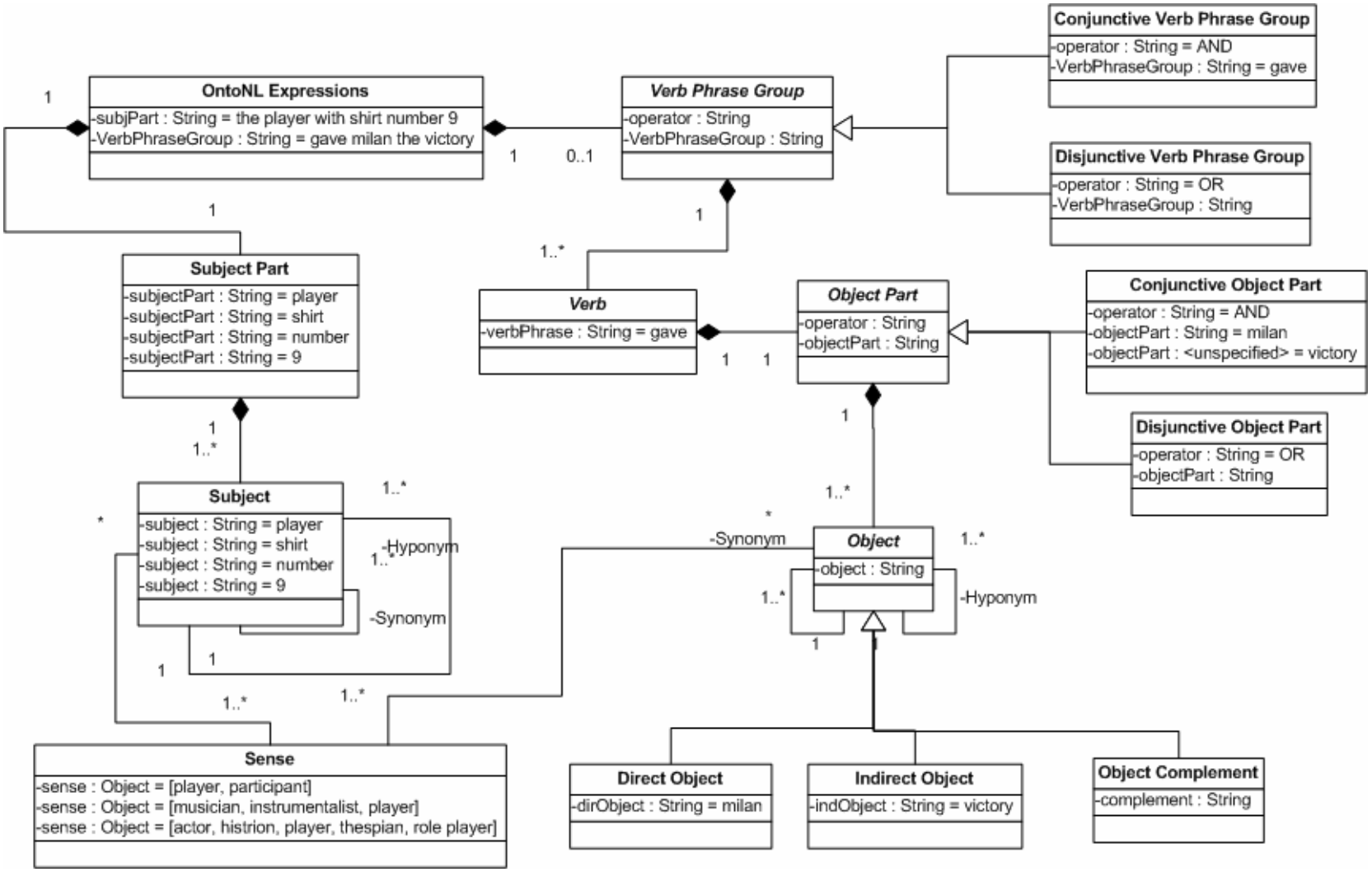


figure 11: An instance of the Language Model for the sentence "The player with shirt number 9 gave Milan the victory"

Chapter 4

Summary and Contributions

In this section we have presented in details the way the natural language interface framework deals with syntactic and domain independent semantic disambiguities (using the WordNet) of plain English words that constitute the user input. The disambiguation as far as it concerns the English language domain is complete. The language model that is produced by the linguistic analysis is easily extendable and is the result of a part of speech tagging process, a grammatical relation analysis, a noun compound bracketing process and a synonym and sense assignment with the help of a word ontology (Wordnet). The result structure is a first step of the natural language processing that is domain independent and can be used in any open domain question answering system using natural language.

The syntactic disambiguation algorithm in the OntoNL is described with the UML activity diagram (Figure 12). The steps are:

1. Parse the natural language expression from the Stanford Log-Linear Part-Of-Speech Tagger.
2. Search for synonyms of the nouns using the WordNet
3. Search for noun n-grams
    a. If yes, bracket the noun compounds
    b. If no, continue
4. Annotate the grammatical relations
5. Build the language model.

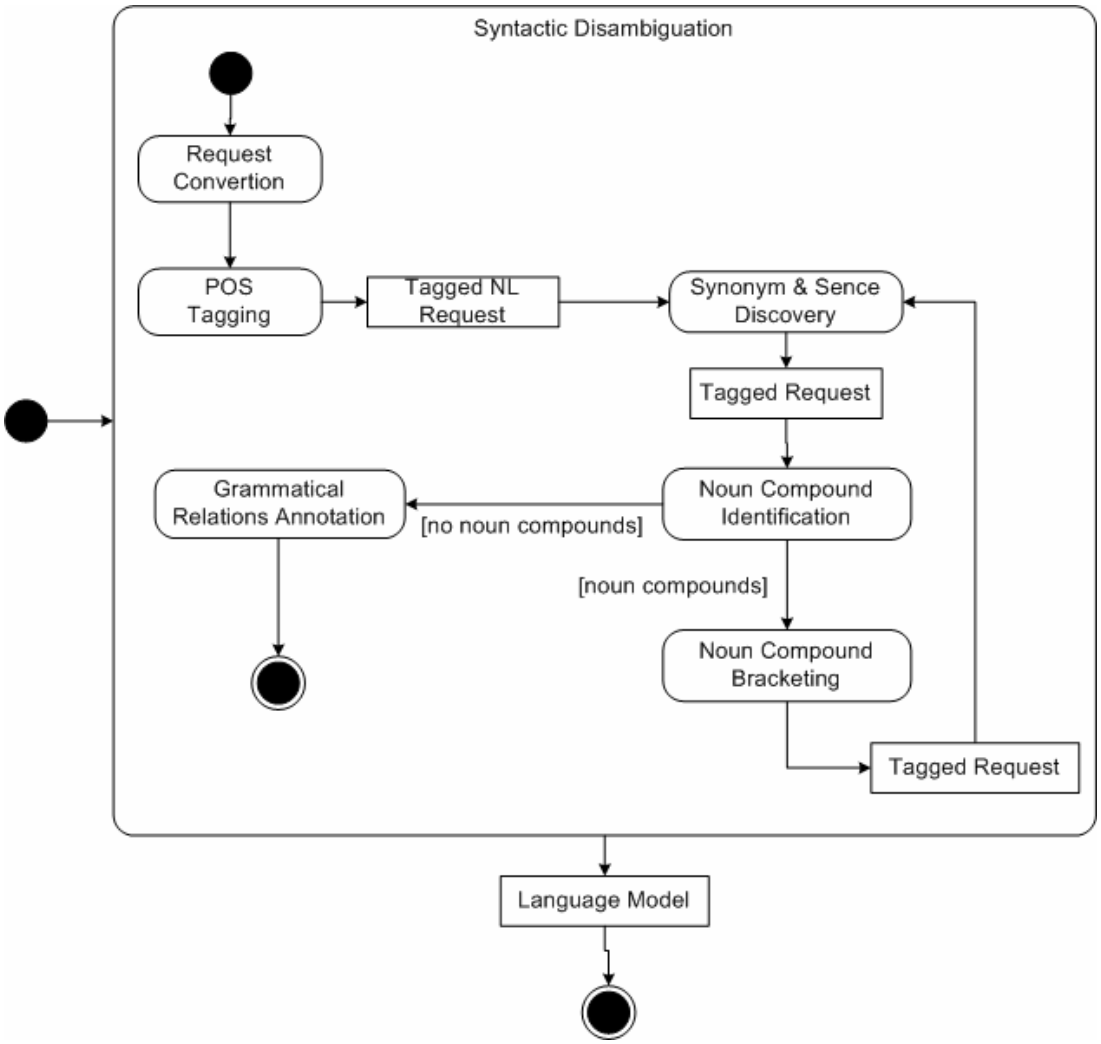


Figure 12: The OntoNL Syntactic Disambiguation Procedure

The result of the design and development of the OntoNL Syntactic Disambiguator is a component that fully explores the grammatical units and relations of the OntoNL natural language expressions, is easily expandable and shows very good results in the disambiguation. The noun compound bracketing mechanism is complete and succesfull and the grammatical relation annotation mechanism is quite promising. While the backbone of the hierarchy is quite similar to that in [Carroll et al., 1999], we use a number of extensions and refinements to facilitate use in applications. Many NP-internal relations play a very minor role in theoretically motivated frameworks, but are an inherent part of corpus texts and can be critical in real-world applications.

The produced language model is a structure that contains information that helps the process of information retrieval with the subject and its complements, the object and its complements and for each word, its synsets and hyponyms from the WordNet.

The OntoNL syntactic disambiguator is a very important component of the framework because it extracts from the natural language expression the subject and the constraints that are then used for information search and retrieval. We could not proceed to the semantic disambiguation without discriminate the units of the utterance that would matter to be semantically disambiguate. The reason for not using an already developed syntactic disambiguator is that we are not aware of a complete module for syntactic analysis that deals with all the grammatical aspects needed for fully syntactically disambiguate English Natural Language Expressions. The OntoNL NL Expressions may not capture all the grammatical rules of English but the structures that have been designed and used are easily expandable without the cost of remodeling the module.

Chapter 5

Domain-Specific Semantic Disambiguation in OntoNL

The purpose of semantic disambiguation in natural language processing, based on a particular domain is to eliminate the possible senses that can be assigned to a word in the discourse, and associate a sense which is distinguishable from other meanings (WordNet gives only generic categories of senses and not domain specific. This domain specific disambiguation is much more powerful).

Sources of Ambiguity

We have found two main factors, which affect an ambiguity of a query, the used vocabulary and syntax and the contents of an ontology:

1. used vocabulary (i.e. ontology)

   For example, if in the vocabulary the concept *researcher* is modelled through three subconcepts: *phDStudent*, *postDoc* and *professor*, the query "∀x ← researcher(x)" can be (mis)interpreted as the information need for information resources about (i) professors, or (ii) phDStudents, or (iii) postDocs or (iv) professors (this is called Clarity), or

   For a second example if we add the term "topic(y, TUC/MUSIC)" to the query "∀ x, y ← researcher(x) and project(y) and participate(x,y)" is redundant (i.e., its adding does not change the list of results), this query can be (mis)interpreted as the information need for information resources about (i) researchers and projects or about (ii) researchers and projects and TUC/MUSIC (it is called Context Clarity) and

2. the domain of the ontology (ontology repository)

   For example, the query for a Chair in an ontology related to an academic area will result in totally different results than the same query in an ontology related to an industry organization about furnitures;

Consequently, in order to handle these situations we define two types of the ambiguity of a query:

- the *semantic ambiguity*, as the characteristic of the used vocabulary (i.e. how many interpretations can be assigned to the words of the given query);

- the *content-related ambiguity*, as the characteristics of the information repository

(i.e. how a query relates the subject of the query with its "neighbour's")

To become more precise we present different levels of disambiguation by using examples from the domain of soccer:

1. The query contains generally keywords that can be resolved by using only the ontology repository (ontological structures and semantics)

ex.1 "… players of soccer team *Milan*"

In this example, the words **players** and **soccer team** are matched to the corresponding concepts of the domain ontology and the information that **Milan** is a soccer team comes from the syntax of the natural language expression (*object complement that follows a direct object*).

2. One of the subject or object part of the language model cannot be disambiguated by using the ontology repository

ex.2 "… information about soccer team *Milan*"

In this example, the word soccer team is matched to the corresponding concept of the domain ontology, but the system cannot resolve the ambiguity of the subject part of the

natural language expression, **information**. The system considers the word **information** as an unresolved concept.

ex.4 "…the players of Barcelona"

In this example, the word **players** is matched to the corresponding concept of the domain ontology but the system cannot resolve the ambiguity of the object part of the natural language expression, **Barcelona**. Since the system cannot find a concept or a property of the domain ontology that could be matched to the word **Barcelona** it "considers" is as a concept instance.

3. Neither the subject nor the object part contains terms disambiguated using the ontological structures

ex.6 "…information about Milan"

In this example neither the word **information** nor the word **Milan** are matched to the ontological structures of the domain ontology. The system "considers" the word **information** as an unresolved concept and the word **Milan** as an unresolved concept instance.

In case that the OntoNL framework had access to the repository information the ambiguities would be resolved according to the information of the repository. For example, from the expression "… the players of Barcelona" from ex.4 if the OntoNL applied queries to the repository about the subject (players) and the object part (Barcelona) of the natural language expression it would get a list of the potential senses of the two terms and it would construct accordingly queries to address again the repository and retrieve the results. However, such a system would have bad performance because it would need mechanisms to develop queries in the corresponding query language the repository schema refers to.

The content-related ambiguity is how to relate the subject with the object if there is more than one sense for either the keyword that describes the subject or the object, in the repository. For example in the query "Give me the players of Barcelona" we can find the instance Barcelona in the repository in more than one sense, (for example as a soccer team

and as a city). In this case the repository will "understand" that the user wants a list of players' names that are from city Barcelona or play for the soccer team Barcelona. We need a mechanism to assign a value (weight) of relatedness between the concept *soccer players* and all other concepts in the ontology in order to better rank candidate queries to address the repository.

Our approach for query refinement uses as source of information the structure and the content of the underlying ontology (vocabulary).

The semantic disambiguation algorithm includes the preprocessing of the ontologies methodology and the semantic search methodology that address the domain ontology. If the information from the ontology structures is enough for the disambiguation, then the procedure concludes to a query with the disambiguated concepts. If the disambiguation is not complete (keywords that cannot be matched to concepts of the ontology) we have developed a methodology that finds strongly related concepts to the subject or object concept of the query and concludes to a list of ranked queries.

The OntoNL Semantic Disambiguation Algorithm

For capturing semantic relatedness using ontologies that describe a domain, there is a need for a preprocessing phase to help and guide the process. The preprocessing of the ontologies serves two scopes; first to ensure meaningful matches between the natural language and the ontology naming and second to calculate the relatedness measure value between the concepts of the domain ontology, so to easily proceed to the semantic ranking. The input of this procedure is the domain ontology and the output is two structures. One for the naming conversions and one for the semantic relatedness measure values (see The OntoNL Ontology-Driven Semantic Ranking section).

The naming conversion procedure is described by the UML activity diagram in figure 13:

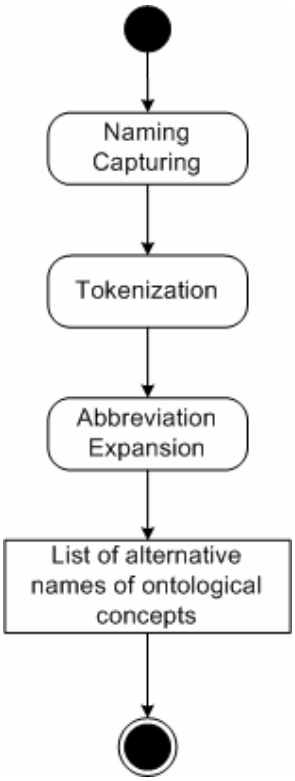


Figure 13: The procedure for finding alternatives natural language names to ontological concepts names

The strings that represent the user request for information may be described with multi-word terms that lead to a tokenization need. The tokenization must capture naming conventions used by database administrators, system integrators, programmers that are responsible for the creation and maintenance of the ontologies and repositories. Multi-word terms are parsed into tokens; by identifying common naming conventions used by programmers with the presence of delimiters such as underscore, numbers, spaces, etc. With this process for example, words such as SoccerTeam will be separated into Soccer and Team.

Another matter that must be taken into account is that in the ontologies we may find concept and relation names like TeamAddr, ShirtNumb. This leads to the need of abbreviation expansion of terms like those by using domain-independent and domain-specific vocabularies like BABEL [http://www.ciw.uni-karlsruhe.de/kopien/babel.html], a glossary of Computer Oriented Abbreviations and Acronyms or TypFast

[http://www.topshareware.com/TypFast-transfer-3297.htm]. Thus, TeamAddr will be expanded into TeamAddress, TeamsAddress, TeamAddressing, etc.

For the case were the ontological structures cannot fully disambiguate the terms of the user query, we need to define the value of relatedness (similarity) between the concepts that describe the domain, by assigning weight values, based on the characteristics concerning the concepts and the relation types between this related concepts. This is a part of the ontology preprocessing phase, but it is going to be fully described in a following section.

According to the level of ambiguity as it was described in the previous subsection, we present the methodology for the semantic search of the ontologies and the ontology repository that conclude to the disambiguation. Next, we describe the entire semantic disambiguation algorithm based on the different levels of ambiguities as they were described in the beginning of the chapter.

SEMANTIC DISAMBIGUATION ALGORITHM

```
Program OntologyStructure SemanticDisambiguation (List, DoubleList)
    List subjOper, objOper, Ambiguity, verbOper, ToOntoStruct;
    LanguageModel LangModel;
    OWLOntology DomOnto;
    OntologyStructure ontoStruct;
    Double relVal;
    DoubleList SemRelMeas, ListNLStoOnto, ListNLOtoOnto;
    String NLSubjTerm, NLObjTerm, string, related, conceptSName,
    conceptOName, instances, oper, conceptS, conceptO;

Begin

For each term NLSubjTerm of LangModel do
   conceptSName = FindConceptMatching(DomOnto)
   oper = FindOperators(NLSubjTerm)
   subjOper.add(oper)
   If conceptSName != null
      ListNLStoOnto.add(conceptSName)
   else
      If NLSubjTerm.size() = 1 && subjOper.size() = 0
         Ambiguity.add(NLSubjTerm)
      Else If NLSubjTerm.size() > 1 & subjOper.size() = 0
         instances = 'different'
         Ambiguity.add(NLSubjTerm)
      Else
         instances = 'different'
         Ambiguity.add(NLSubjTerm)

For each term NLObjTerm of LangModel do
```

```
    conceptOName = FindConceptMatching(DomOnto)
    oper = FindOperators(NLObjTerm)
    objOper.add(oper)
    If conceptOName != null
       ListNLOtoOnto.add(conceptOName)
    else
       If NLObjTerm.size() != 1 && objOper.size() = 0
          Ambiguity.add(NLObjTerm)
       Else If NLObjTerm.size() = 1 & objOper.size() = 0
          instances = 'different'
          Ambiguity.add(NLObjTerm)
       Else
          instances = 'same'
          Ambiguity.add(NLObjTerm)

If Ambiguity.size()= 0
    relVal = 1
    For each term of ListNLStoOnto do
       ToOntoStruct.add(ListNLStoOnto(getTerm(i)))
       ToOntoStruct.add(relVal)
       ontoStruct.add(ToOntoStruct)
       ToOntoString.removeAll()
    For each term of ListNLOtoOnto do
       ToOntoStruct.add(ListNLOtoOnto(getTerm(i)))
       ToOntoStruct.add(relVal)
       ontoStruct.add(ToOntoStruct)
       ToOntoString.removeAll()
Else
    If ListNLStoOnto != null && ListNLOtoOnto == null
       For each term conceptS of ListNLStoOnto do
          string = ListNLStoOnto.get(concept)
          ToOntoStruct.add(string)
          relVal = 1
          ToOntoStruct.add(relVal)
          OntoStruct.add(ToOntoStruct)
          ToOntoString.removeAll()
          related = SemRelMeas.getMostRelated(string)
          ToOntoStruct.add(related)
          relVal = SemRelMeas.getRelValue(string, related)
          ToOntoStruct.add(relVal)
          OntoStruct.add(ToOntoStruct)
          ToOntoString.removeAll()
    If ListNLStoOnto == null && ListNLOtoOnto != null
       For each term conceptO of ListNLOtoOnto do
          string = ListNLOtoOnto.get(concept)
          ToOntoStruct.add(string)
          relVal = 1
          ToOntoStruct.add(relVal)
          OntoStruct.add(ToOntoStruct)
          ToOntoString.removeAll()
          related = SemRelMeas.getMostRelated(string)
          toOntoStuct.add(related)
          relVal = SemRelMeas.getRelValue(string, related)
          ToOntoStruct.add(relVal)
          OntoStruct.add(ToOntoStruct)
          ToOntoString.removeAll()
```

```
    If ListNLStoOnto != null && ListNLOtoOnto != null
          For each term conceptO of ListNLOtoOnto do
          string = ListNLOtoOnto.get(concept)
          related = SemRelMeas.getMostRelated(string)
          toOntoStuct.add(related)
          relVal = SemRelMeas.getRelValue(string, related)
          ToOntoStruct.add(relVal)
          OntoStruct.add(ToOntoStruct)
          ToOntoString.removeAll()
End
```

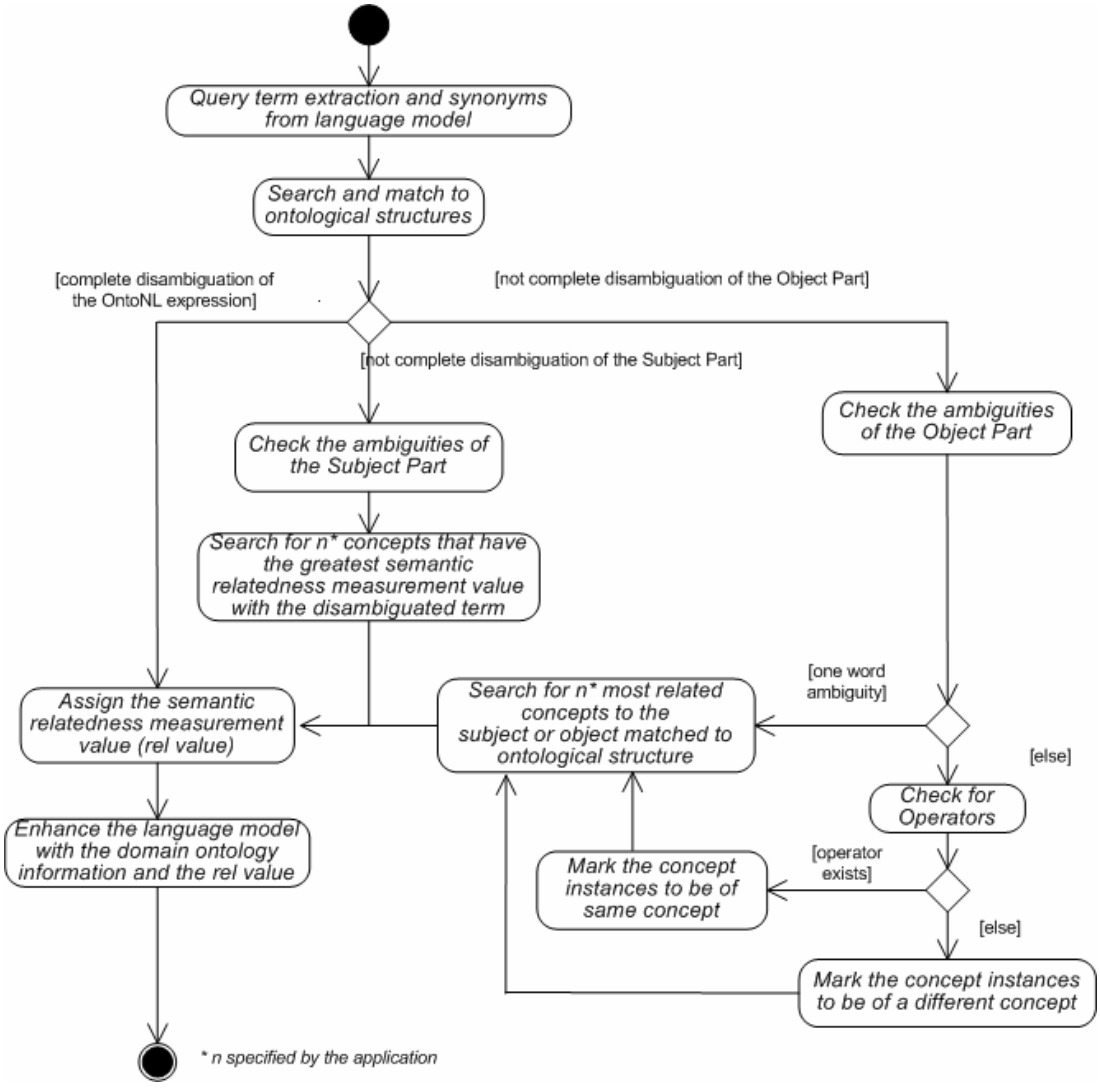


Figure 14: The OntoNL Semantic Disambiguation procedure

In Figure 14 we present a UML activity diagram representing the OntoNL Disambiguation Procedure. We explain the figure by a numbered list of actions based on the three types of the OntoNL ambiguity. It is a general approach where the disambiguation is based on an OWL repository. Before proceeding we need to mention that every time we semantically search the repository we apply not only the exact word as it was referred in the user query but all the synonym set provided by Wordnet.

Steps of the algorithm for the Semantic Disambiguation

1.

1.1. Search and match the query terms to the concepts and relations of the ontology.

1.2. Assign the terms that follow the matched concepts of the ontology as concept instances.

1.3. Create a language model with the correspondence of the query terms to the subject concept, the object/object complement concepts, relations and keywords and assign the weight value 1 to the structure.

This structure is an enhanced language model with the structural and semantic dependencies that derive from the ontological structures.

2.

2.1. The Subject Part consists of one word that cannot be matched with the concepts of the domain ontology

2.1.1. Search for a number specified by the application of ontology concepts that have the greatest relatedness value with the disambiguated term of the request.

2.1.2. Create a list of instances of the language model like step 1.3 for each pair of concepts from step 2.2.1. Assign the structure with the relatedness measure value.

2.2. The object part consists of one word that cannot be matched with the concepts of the domain ontology or

2.3. The object part consists of more than one word, and they are not separated with any other words. We gather the words in one string that comprises the query term to address the repository.

2.3.1. Search for a set of the strongest semantic related concepts to the concept of the subject.

2.3.2. Create a list of instances of the language model like step 1.3 for each pair of concepts from step 2.2.1. Assign the structure with the relatedness measure value.

2.4. The user query contains more than one query term (many words), that are separated with other words (we filter from a stop word file the in between the terms words to cut the less significant ones).

2.4.1. Check for the existence of operators AND-OR between the query terms. If there is an AND or OR between them consider that the concept the instances belong cannot be different for each query that the system creates.

2.4.2. Do Steps 2.2.1 and 2.2.2 with step 2.3.1 as constraint.

2.4.3. If there is no AND or OR between them consider that the concept the instances belong must be different for each query that the system creates.

2.4.4. Do Steps 2.2.1 and 2.2.2 with step 2.3.3 as constraint.

3. Create a query with the terms of the sentence connected with a Boolean AND and queries with the term of the subject connected with Boolean OR to the terms of the object.

The language model is described by the UML class diagram in figure 15. The semantic disambiguation algorithm contributes in the Ontology Structure class. The class contains two attributes: the class and the value. The class refers to the OWL Class and the value to the OWL Class Instance.

Chapter 5

The procedure of finding alternative, more natural language like names for the ontological structures is an offline procedure and happens before the Ontology-Driven Semantic Ranking, because the OntoNL Ontology-Driven Semantic Ranking is also an offline procedure but needs the alternative names because the ranking contains a part that calculates the related senses of the ontological concepts, so to function well the input must be as more natural language like as possible.

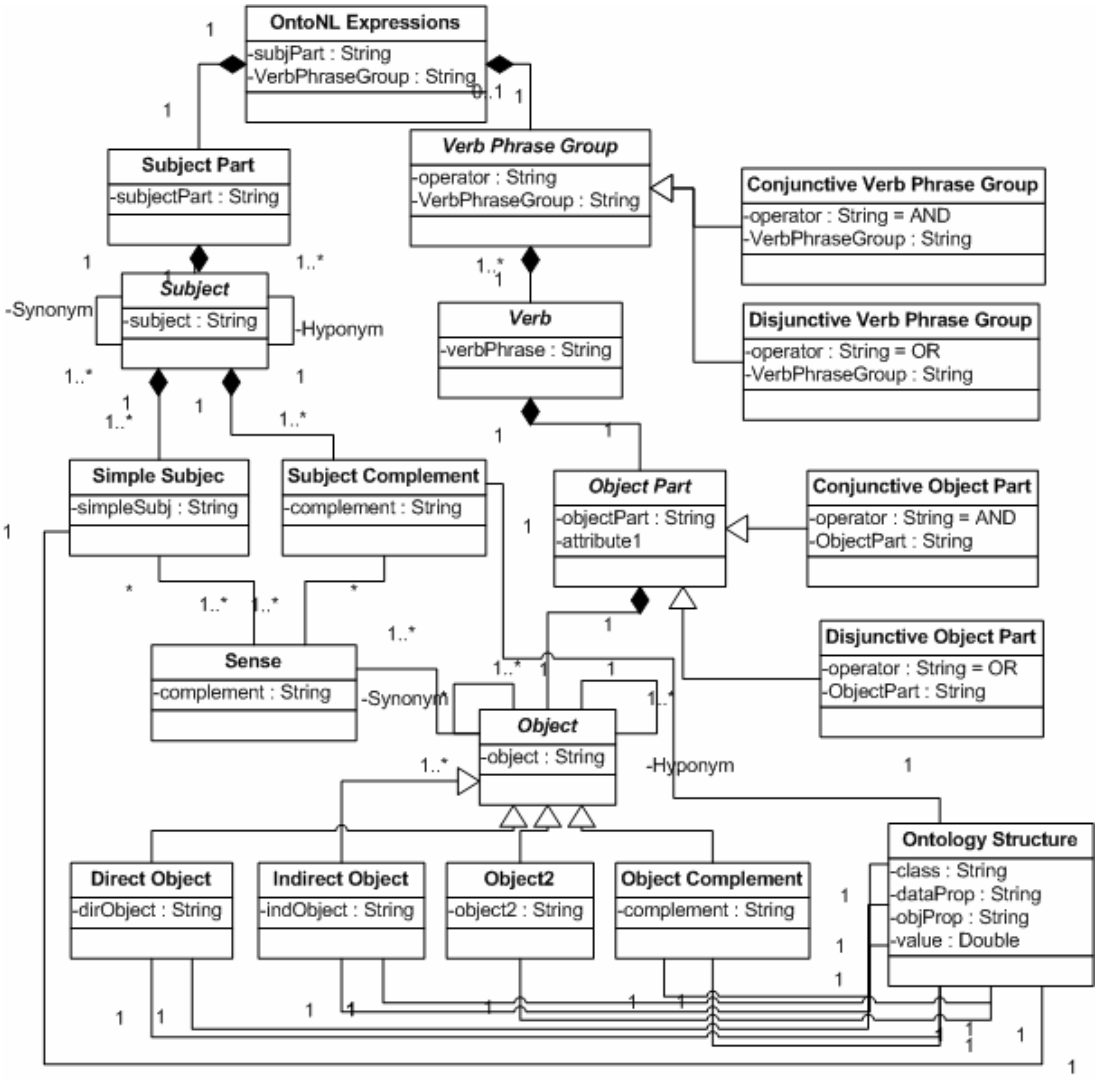


Figure 15: The language model that derives from the syntactic and semantic analysis

Chapter 5

The OntoNL Ontology-Driven Semantic Ranking

When a query cannot be disambiguated completely in the OntoNL Semantic Disambiguation phase, OntoNL returns all the possible results ranked according to the possibility that the user has requested them. To compute the ranking of possible results, OntoNL borrows ideas and develops new ones from the research of Semantic Relatedness of concepts in a semantic network. Semantic Relatedness using network representations is a problem with a long history in artificial intelligence and psychology, starting from the first approaches of Quillian [Quillian, 1968] and Collins [Collins, 1975], which used a spreading activation approach. Although many measures of relatedness are defined in the literature, they are seldom accompanied by an independent characterization of the phenomenon they measure, but the worth of a relatedness measure is in its fidelity to human behavior [Resnik, 1995].

A domain can be described by a set of core and domain ontologies. It is crucial to identify more specific domains inside a domain, based on concepts and relationships of those concepts. For doing that we must define the degree of relatedness between pairs of concepts. This leads to a matrix containing a weight of relatedness between any two concepts. Consider an n^{th} row in this matrix and a function $F_n(i)$ which takes the n^{th} row and returns the set of the largest values. Then this function defines a local association cluster around the concept C_n. The clustering has the effect of reducing the size of a domain by creating groups of more specific information from one or more ontologies to search for semantic information.

Clustering techniques rely on the existence of some suitable similarity metric for objects [Faloutsos, 1996], [Salton, 1989], [Steinbach et al., 2000], [Jain et al., 1988, 1999]. Clustering algorithms have been applied in many fields, like Marketing, for finding groups of customers with similar behavior, Biology, for the classification of plants and animals given their features, Libraries, for book ordering, Insurance, WWW, etc.

Clustering algorithms may be classified in the following categories; Exclusive Clustering, Overlapping Clustering, Hierarchical Clustering and Probabilistic Clustering. In the first

case data are grouped in an exclusive way, so that if a certain datum belongs to a definite cluster then it could not be included in another cluster. The second type uses fuzzy sets to cluster data, so that each point may belong to two or more clusters with different degrees of membership. In this case data will be associated to an appropriate membership value. Instead a hierarchical clustering algorithm is based on the union between the two nearest clusters. The algorithm starts with the assumption that every datum is a cluster and after a few iterations it reaches the final clusters wanted. Finally, the last kind of clustering uses a completely probabilistic approach.

The semantic relatedness clustering of the ontologies that describe a domain is based on a methodology for weighting. We describe next a methodology that leads to a semantic relatedness clustering in OWL ontologies.

All the research results presented in the literature so far [Rada et al, 1989], [Sussna, 1993, 1997], [Wu and Palmer, 1994], [Leacock and Chodorow, 1998], [Resnik, 1995], [Jiang and Conrath, 1997], [Lin, 1998] were tested in specific ontologies like the WordNet and the MeSH ontology that are well formed and contain synonym sets and glosses (descriptions of a term meaning and it consists of a descriptive part and an example of use case) that come with every synonym set. The methodologies that have been proposed are not general and have not been tested in domain ontologies.

Another issue that arises is that all the previous approaches are symmetric, but we need a measure to be asymmetric. Asymmetric relatedness denotes that the relatedness between A and B is not necessarily the same as the relatedness between B and A. This is an important aspect for natural language processing, since relations that are described with natural language do not indicate mathematical rules. For example, the relatedness between a father and his son is different and depends on the initial concept of consideration. The difference is that the son inherits the facial characteristics of the father but the opposite is not true.

We propose a method that can be used for computing semantic relatedness between concepts that belong to an OWL domain ontology by using also information coming from WordNet.

The measure will be based on the commonality (based on the semantic relations and the conceptual distance) and the related senses.

All ontology concepts and their weights after the relatedness calculation are stored in the system repository. Therefore, the ontology is ready to be used in the semantic disambiguation processor.

We also must take into account the semantic relation of **EquivalentClass**. The EquivalentClass of the source class (the class that was mapped from the ontological structures to the subject of the natural language expression) has a **similarity value 1** with the source class so we also consider the relatedness measurement value of the equivalent class with the remainder classes of the Ontology.

The commonality depends on the amount of the common information two concepts share. We cannot use commonality like Resnik or Jiang and Conrath have used when dealing with domain ontologies other than WordNet were there are no senses to count the frequency of a word. We can accept partly that the distance from the most specific common subsumer of the two concepts is a criterion that must be taken into account but also the number of common relations. We do need to keep the measure asymmetric so it will depend on the reference concept of which the relatedness to another concept we calculate. The commonality depends on the amount of the common information two concepts share. The commonality measure in general depends on two factors: The position of the concepts relatively to the position of their most specific common subsumer (how far is their common father) and the reciprocity of their properties (if the connecting OWL ObjectProperties have also inverse properties because this denotes that they are stronger related to each other). The position of the concepts relatively to the position of their common subsumer will be examined by the conceptual distance and the specificity measurement:

We first count the number of the common properties the two concepts share (numerator) and divide it with the number of the initial concept (denominator), with the factor rel_{OP1}:

$$for \sum_{i=1}^{n} p_{i1} > 0 : rel_{OP1}(c_1, c_2) = \frac{\sum_{i=1}^{n} p_{i12}}{\sum_{i=1}^{n} p_{i1}}, \quad (24)$$

The value p_{i1} represents the fact that concept C_1 is related to concept C_i (value: 0 or 1 in general). The value p_{i12} represents the fact that both concepts C_1 and C_2 are related to concept C_i (value: 0 or 1 in general). The basic idea of this measure is that concepts share more common properties with other concepts that relate.

For example, in the domain of soccer and in the particular domain ontology of soccer [http://elikonas.ced.tuc.gr/ontologies/] the concept **PlayerObject** has 20 Object Properties: PhysicalCondition, SufferCondition, CauserOfRelation, ReceiveRelation, PerformerOfRelation, ParticipantOfRelation, CommitRelation, PenalizedWithRelation, ScoresRelation, Agent, AgentRef, Header, Relation, Property, DefinitionSB, Label, AbstractionLevel, MediaOccurence, Object, ObjectRef.

The concept **CoachObject** has 13 Object Properties: OrderRelation, ProvideRelation, Agent, AgentRef, Header, Relation, Property, DefinitionSB, Label, AbstractionLevel, MediaOccurence, Object, ObjectRef.

The concept **SoccerTeamObject** has 18 Object Properties: AwardedRelation, SoccerTeamObjectAgentOfRelation, AgainstRelation, SubstanceRelation, ReceiveRelation, ParticipantOfRelation, ScoresRelation, Agent, AgentRef, Header, Relation, Property, DefinitionSB, Label, AbstractionLevel, MediaOccurence, Object, ObjectRef.

As we can see the **PlayerObject** concept shares 11 common Object Properties with the **CoachObject** concept**,** but it shares 14 common Object Properties with the **SoccerTeamObject** concept. Based on the common properties measure we claim that the PlayerObject is more related to the SoccerTeamObject than with the CoachObject

We then count the number of the common properties the two concepts share that are **inverseOf** properties (numerator) and divide it with the number of the common properties the two concepts share (denominator)), with the factor rel_{OP2}:

$$for \sum_{i=1}^{n} p_{i12} > 0 : rel_{OP2}(c_1, c_2) = \frac{\sum_{i=1}^{n} p_{invi12}}{\sum_{i=1}^{n} p_{i12}}, \quad (25)$$

were the p_{invi12} represents the fact that both concepts are inversely related.

The motivation for measuring the common **inverseOf** properties is to dissociate the relatedness measure from the similarity measure. If we only counted the common **ObjectProperties** then we would assign a great value of relatedness between siblings (subclasses with common superclass) which are similar but not semantically related as the OntoNL Framework defines.

For example, in the domain of soccer and in the particular domain ontology of soccer [http://elikonas.ced.tuc.gr/ontologies/] the concept **Goal** has 17 Object Properties: CauserRelation, ScoredByRelation, AgainstOfRelation, OnRelation, PlacedKickResultOfRelation, SemanticPlace, EventRef, Event, SemanticTime, Header, Relation, Property, DefinitionSB, Label, AbstractionLevel, MediaOccurence, PerformerRelation.

The concept **PlayerObject** has 20 Object Properties: PhysicalCondition, SufferCondition, CauserOfRelation, ReceiveRelation, PerformerOfRelation, ParticipantOfRelation, CommitRelation, PenalizedWithRelation, ScoresRelation, Agent, AgentRef, Header, Relation, Property, DefinitionSB, Label, AbstractionLevel, MediaOccurence, Object, ObjectRef.

The concept **Score** has 11 Object Properties: ScoreOfRelation, SemanticPlace, AttributeValuePair, SemanticTime, Header, Relation, Property, DefinitionSB, Label, AbstractionLevel, MediaOccurence.

From the domain ontology we can retrieve the information that the concept **Goal** has 9 common Object Properties with the **PlayerObject** and 9 common properties with the **Score** concept. The difference in this example is that from those 9 common properties between the concept **Goal** and the concept **PlayerObject,** 2 of them are inverse (**Goal**:CauserRelation←→**PlayerObject**:CauserOfRelation and **Goal**:ScoredBy←→**PlayerObject**:CauserOfRelation). We claim that the concept **Goal** is more related to the concept **PlayerObject** than to **Score** concept.

These two measures that concern the properties features relatedness are combined based on a factor that shows the relative importance of these two measures (*f* values):

$$rel_{prop}(c_1, c_2) = (f_1 \times \frac{\sum_{i=1}^{n} p_{i12}}{\sum_{i=1}^{n} p_{i1}}) + (f_2 \times \frac{\sum_{i=1}^{n} p_{invi12}}{\sum_{i=1}^{n} p_{i12}}), \quad (26)$$

were $f_2 \geq 0$ and $f_1 + f_2 = 1$.

The factors f_1 and f_2 take values based on the evaluation of the application of the OntoNL framework in each domain of use. By defining the factors f_1 and f_2 we allow the dynamic revision of the measure after the study of the domain ontology and the evaluation of the results of the measure in the particular application.

The benefit of using this measure is that it is asymmetric. A consequence of this measure is that when two concepts are quite close in the hierarchy ((c_1 subclass of c_2) or (c_1 and c_2 share common superclass)) they share more common relations (it captures the IS-A semantic through object properties). The second part of the measure is very important because it captures the concept relatedness in a way that cannot be expressed through IS-A taxonomies.

We need to point that when we detect that a source concept-class is related via an **ObjectProperty** with the target concept, the measure value becomes 1.

The **conceptual distance measure** is based on two factors; the path distance and the specificity. The **specificity** of the concepts is based on their position in the ontology (the

leaf nodes are the most specific concepts in the hierarchy). The path distance counts the minimal path of edges in a path from a concept to another. Within one conceptual domain, the relatedness of a concept (c_1) to another concept (c_2) is defined by how closely they are related in the hierarchy, i.e., their structural relations (IS-A relation). In the OntoNL, the IS-A relations are implemented through the rdfs:subclassOf syntax of OWL. The parameter that differentiates our measure from the classic measures of distance counting is the change of direction that is combined with the specificity factor. We claim that when the change of direction (from superclassing to subclassing) is close to the initial concept-c_1 (that is the **subject** of the natural language expression) of the pair we test the relatedness, the two concepts are more related. When the direction of the path changes far from the first concept then the semantics change quite as well (more specialization). Also we take into account the place of the concepts in the hierarchy. The terms located higher in the hierarchy have higher values of relatedness than located terms lower in the hierarchy.

The value of distance can be measured with the following measure

$$pathDist(c_1, c_2) = \frac{d_{C1} + d_{C2}}{2 * D} \in (0,1]$$

where dc_1 is the number of edges to go from the concept 1 to the closer common superconcept (subsumer) and dc_2 the number of edges to go from the concept 2 to the closer common superconcept (subsumer). With D we count the maximum depth of the ontology. The OntoNL disambiguation algorithm uses the relatedness of concepts of the domain ontologies and not the similarity, so the measure excludes the cases were $d_{C1} = 0$, $d_{C2} = 0$ and $d_{C1} + d_{C2} = 2$. So, the path distance measure becomes

$$\forall d_{C1}, d_{C2} \geq 1, d_{C1} + d_{C2} > 2 : \; pathDist(c_1, c_2) = \frac{d_{C1} + d_{C2}}{2 * D} \in (0,1] \text{ , } (27)$$

We need to define a factor to determine the specificity of the concepts inside the ontology. As we have already stated when the value of d_{C1} is close to the value of $(d_{C1}+d_{C2})/2$ then the relatedness must be decreased, because the initial concept C1 is specialized a lot in comparison with the subsumer concept. So:

$$w1_{spec_{C1}} = \begin{cases} -\log \frac{2 \times d_{C1}}{d_{C1} + d_{C2}} \in (0,1], \text{ if } d_{C1} < \frac{d_{C1} + d_{C2}}{2} \\ \\ 0, \qquad \text{if } d_{C1} \geq \frac{d_{C1} + d_{C2}}{2} \end{cases} \quad (28)$$

We also propose a method of counting the specialization of the concept – c1 based on the object properties of the subsume. We define the factor:

$$spec_{C1} = \frac{\#ObjP_{C1} - \#ObjP_S}{\#ObjP_S} \in [0, \infty) \quad (29)$$

were $ObjP_{C1}$ is the number of Object Properties of the concept C1 and $ObjP_S$ is the number of ObjectProperties of the subsumer concept. If the factor becomes 1 or greater then the specialization is so big that we cannot count the relatedness based on the specificity. The range of the $spec_{C1}$ is $[0, \infty)$. To limit the range in $[0,1]$ we need to restrict the number of ObjectProperties of the concept C1. We normalize the factor and we subtract it from 1, with the restriction that the number of the ObjectProperties of the concept – c1 is at most 10 times the number of the ObjectProperties of the subsumer.

$$\forall \#ObjP_{C1} \leq 10 \times \#ObjP_S : w2_{spec_{C1}} = 1 - \log \frac{\#ObjP_{C1}}{\#ObjP_S} \in [0,1] \quad (30),$$

$$\text{else } w2_{spec_{C1}} = 0 \quad (31)$$

The conceptual distance measure then becomes

$$rel_{CD} = (w1_{specC1} + w2_{specC1} + 1 - pathDist(c_1, c_2)) / 3 \quad (32)$$

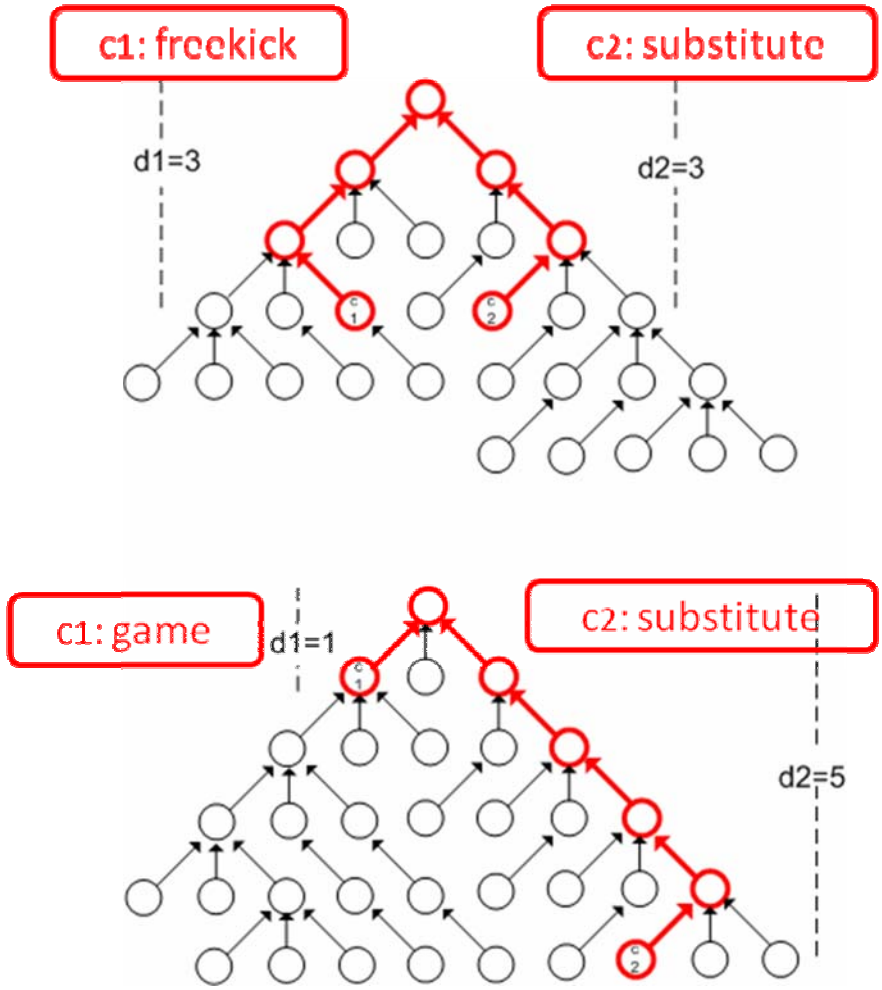


Figure 16: Two figures with the same Conceptual Distance ($conceptualDist(c_1, c_2) = \frac{6}{2*5}$ **) and different specificity (** $w_{spec} = -\log\frac{2\times 3}{3+3} = 0$ **and** $w_{spec} = -\log\frac{2\times 1}{3+3} = \log 3 \approx 0.48$ **respectively)**

The **amount of related senses measure** is a measure that concerns the domain ontology and the WordNet Ontology. As we have already mentioned, glosses are descriptions of a word's sense and it consists of a descriptive part and an example of use case. In case where the domain ontology does not have descriptions of each concept-class we need to extract this information from the WordNet. The measure is based on sets of each concept that contain synonyms and nouns extracted from the descriptions used in the OWL

domain ontology or if there are no descriptions from the descriptive part of the glosses of the concept:

$$rel_{RS}(c_1, c_2) = \frac{\left|S_1 \cap S_2\right|}{|\, S_1 \cap S_2 \,| + \left|S_1 \setminus S_2\right|}, \quad (33)$$

were S_1 is the description set of senses for concept c_1 and S_2 the description set of senses for concept c_2.

For example, we consider the terms **football game** and **football player**. After applying the term **football game** in the WordNet and after pos-tagging the synonyms and the descriptive parts of them we retrieve 6 different nouns: football (5 times), game (4 times), teams (2 times), players (1 time), ball (3 times), and goal (2 times). After following the same procedure for the term **football player** we retrieve 9 nouns: athlete (2 times), football(1 time), football player(1 time), footballer(1 time), jock(1 time), person(1 time), sports(1 time), player(2 times), participant(1 time) and game(1 time). The description set of the **football game** is S_1 = (football, game, teams, players, ball, goal). Accordingly, the description set of the **football player** is S_2 = (athlete, football, footballer, jock, person, sports, player, participant, game). The equation (27) then becomes:

$$rel_{RS}(\textit{football game, football player}) = \frac{3}{3+3} = \frac{1}{2}$$

Now we will try to measure the same measure also for the terms **football game** and **football coach**. The description set of the **football coach** is S_2 = (football, player, manager, handler, athlete, team). The equation (27) then becomes:

$$rel_{RS}(\textit{football game, football coach}) = \frac{3}{3+3} = \frac{1}{2}$$

We see that the amount of the related senses measure value is the same for the two pairs, which is essentially correct since the two concepts (football players and football coaches) participate jointly in a football game.

The overall relatedness measure is the following:

$$rel_{OntoNL}(c_1, c_2) = w_1 \times rel_{PROP}(c_1, c_2) + w_2 \times rel_{CD}(c_1, c_2) + w_3 \times rel_{RS}(c_1, c_2) \quad (34)$$

were $w_1 + w_2 + w_3 = 1, (w_1, w_2, w_3) > 0$ and $rel_{PROP}(c_1, c_2), rel_{CD}(c_1, c_2), rel_{RS}(c_1, c_2) \in [0,1]$.

The range of values for the three factors w_1, w_2 and w_3, will be discussed in details in the evaluation chapter.

The measure is applied to all concepts of the ontology in the preprocessing phase and constructs a NxN matrix, the OntoNL Semantic Ranking Matrix, were N is the total number of concepts, with the relatedness values of each concept with all the other concepts inside the disambiguation ontology.

Representation of the processed Natural Language Interactions in OntoNL

After the syntactic and semantic disambiguation we have concluded to the subject of the query specialized by additional description that forms the object or possible objects of the query. We need a formal way to represent the query, a standardized query language that will meet the specification of the ontology language (OWL) and will be easily mapped to various forms of repository constructions. Although we could in principle use an internal representation of the preprocessed NL interactions, we opted to use a representation that is near to the languages used in the Semantic Web, so that when the repository is based on OWL or RDF to be able to directly use it to access the repository.

OWL has been developed as a vocabulary extension of RDF (the Resource Description Framework). The Resource Description Framework (RDF) is a flexible and extensible way to represent information about World Wide Web resources and enables data to be decentralized and distributed. It is used to represent, among other things, personal information, social networks, metadata about digital artifacts, as well as to provide a means of integration over disparate sources of information. A standardized query language for RDF data with multiple implementations offers developers and end users a

way to write and to consume the results of queries across this wide range of information. Used with a common protocol, applications can access and combine information from across the Web.

Working on RDF query languages has been progressing for a number of years. Several different approaches have been tried, ranging from familiar looking SQL-style syntaxes, such as RDQL (http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/) and Squish (http://ilrt.org/discovery/2001/02/squish/), through to path-based languages like Versa (http://copia.ogbuji.net/files/Versa.html).

Among these approaches, those that emulate SQL syntactically have probably been the most popular and widely implemented. This is perhaps surprising given the very different models that lurk behind relational databases and RDF -- familiarity with syntax has no doubt contributed to this success.

Query Formulation to SPARQL

In order to describe the mechanism that translates the disambiguated natural language query to a SPARQL query for the disambiguation ontology, we first summarize the information that the OntoNL system gathered after the semantic and syntactic disambiguation procedures.

In this point we should remind the reader that even in an application of the system that acts as a question answering system, we isolate the part of the query sentence that contains the information to be processed and act as if it was a natural language expression.

The cases of possible natural language expressions represented in a class diagram are presented in figure 13:

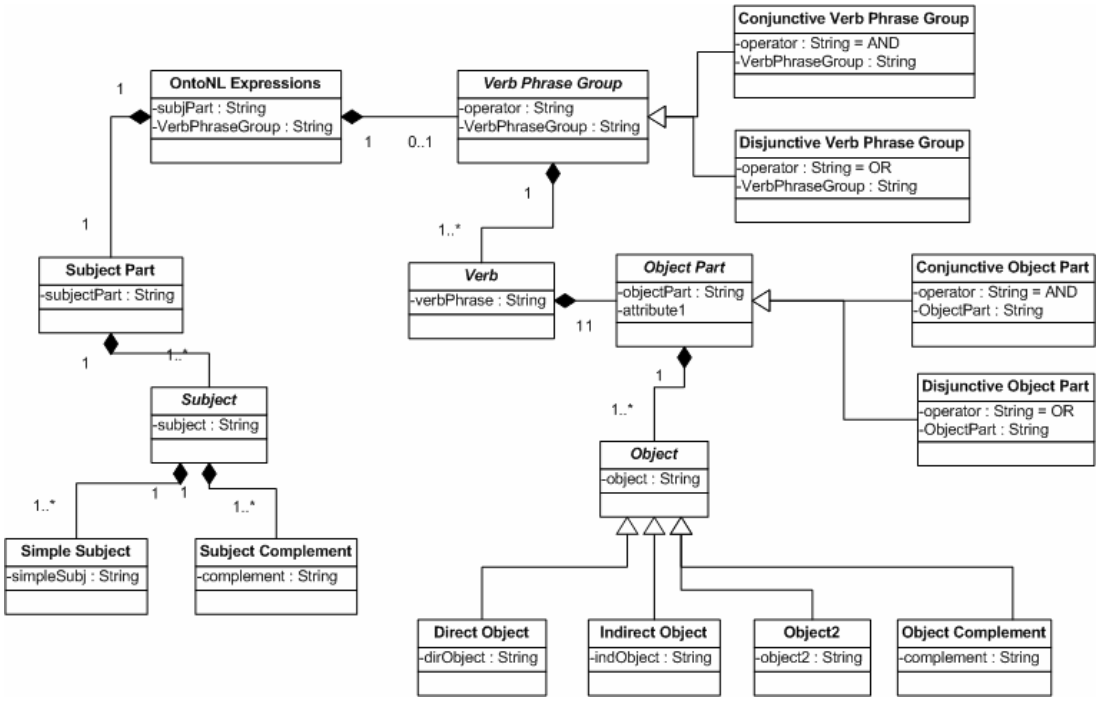


Figure 17: A class diagram representing the grammatical relationships in an OntoNL sentence

A user query can have one or more subjects with one or more specific attributes, the object and its complement. Also, a query can consist of more than one subject and object by using operators like OR and AND. We are going to present in table 3 the potential interpretations from the ontological structures that can be assigned to the terms of the subject and the object part. The term *Class* is the correspondent **OWL Class** in the domain ontology that matched the word that the user used in his/her query and the *Object Property* and *Datatype Property* are the correspondent **OWL ObjectProperty** and **OWL DatatypeProperty**. In the presence of an operator that was detected inside the user query we can consider that either one of the subject or object or both of them will be more than one.

After the semantic disambiguation phase (WordNet and Ontological Structures) the different types of queries that have to be processed by OntoNL are:

	SUBJECT PART	OPER.	VERB	OBJECT PART	OPER.
1	Class or Datatype Property				
2	Class or Datatype Property	AND/ OR			
3	Class or Object - Datatype Property		1	Class or Object - Datatype Property Value	
4a	Class or Object - Datatype Property	AND/ OR	1	Class or Object - Datatype Property Value	
4b	Class or Object - Datatype Property		1	Class or Object - Datatype Property Value	AND/ OR
4c	Class or Object - Datatype Property	AND/ OR	1	Class or Object - Datatype Property Value	AND/ OR
5	Value				
6	Value		AND/ OR		
7	Value		1	Value [1, N]	
8a	Value	AND/ OR	1	Value	
8b	Value		1	Value	AND/ OR
8c	Value	AND/ OR	1	Value	AND/ OR

Table 3: Types of user queries after the structural disambiguation from Ontologies

Figure 18 shows an example ontology which we will be using in order to explain the construction of SPARQL queries for each one of the query types of Table 3. Figure 18 shows a total of 20 classes interconnected in a semantic network. The objective of this example semantic network is to simulate a part of an ontology structure that our query types will be using. Figure 1 shows the same sematic network of Figure 18 but structured in a more "OWL" representation where the various properties are named. We consider that the general ontology is located in the http://www.owl-ontologies.com/general.owl# path.

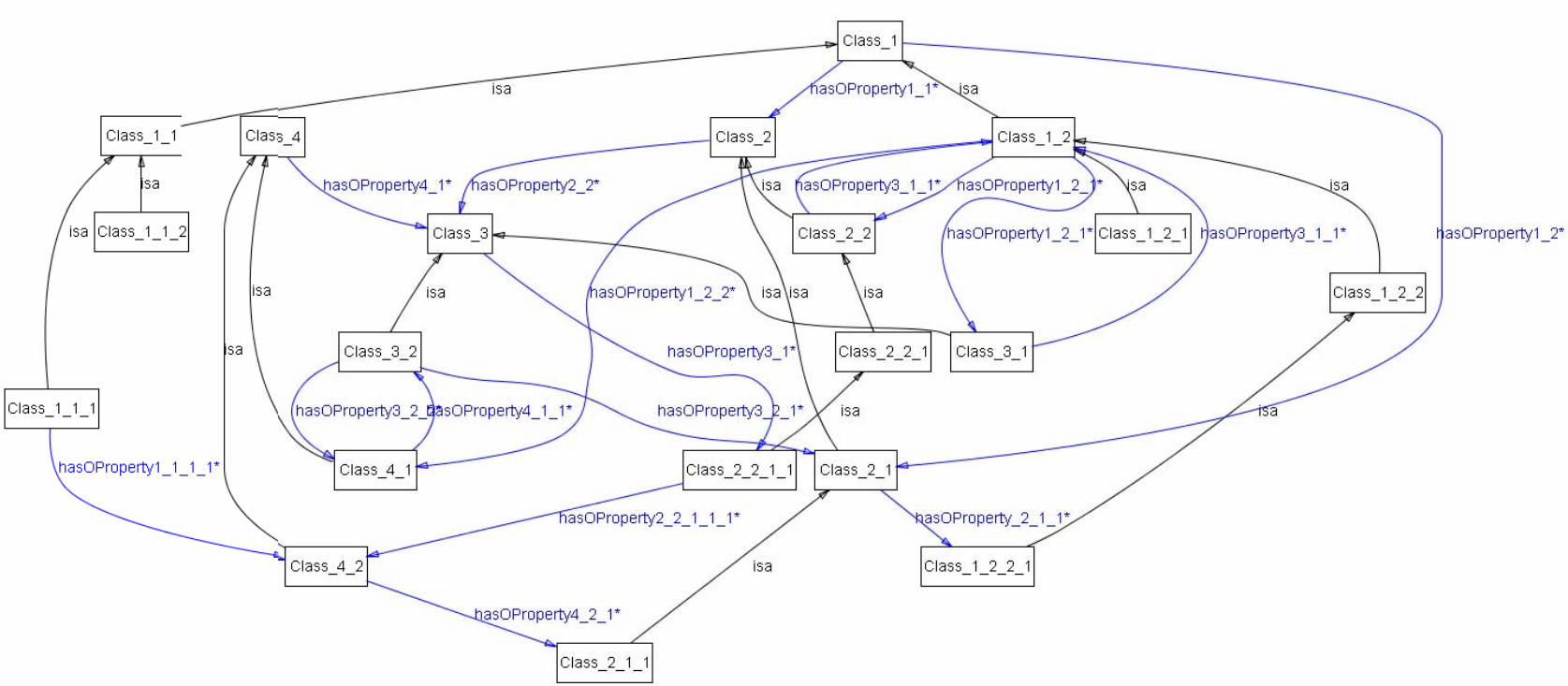


Figure 18: A graphical view of a general OWL ontology with IS-A relations and Object Properties

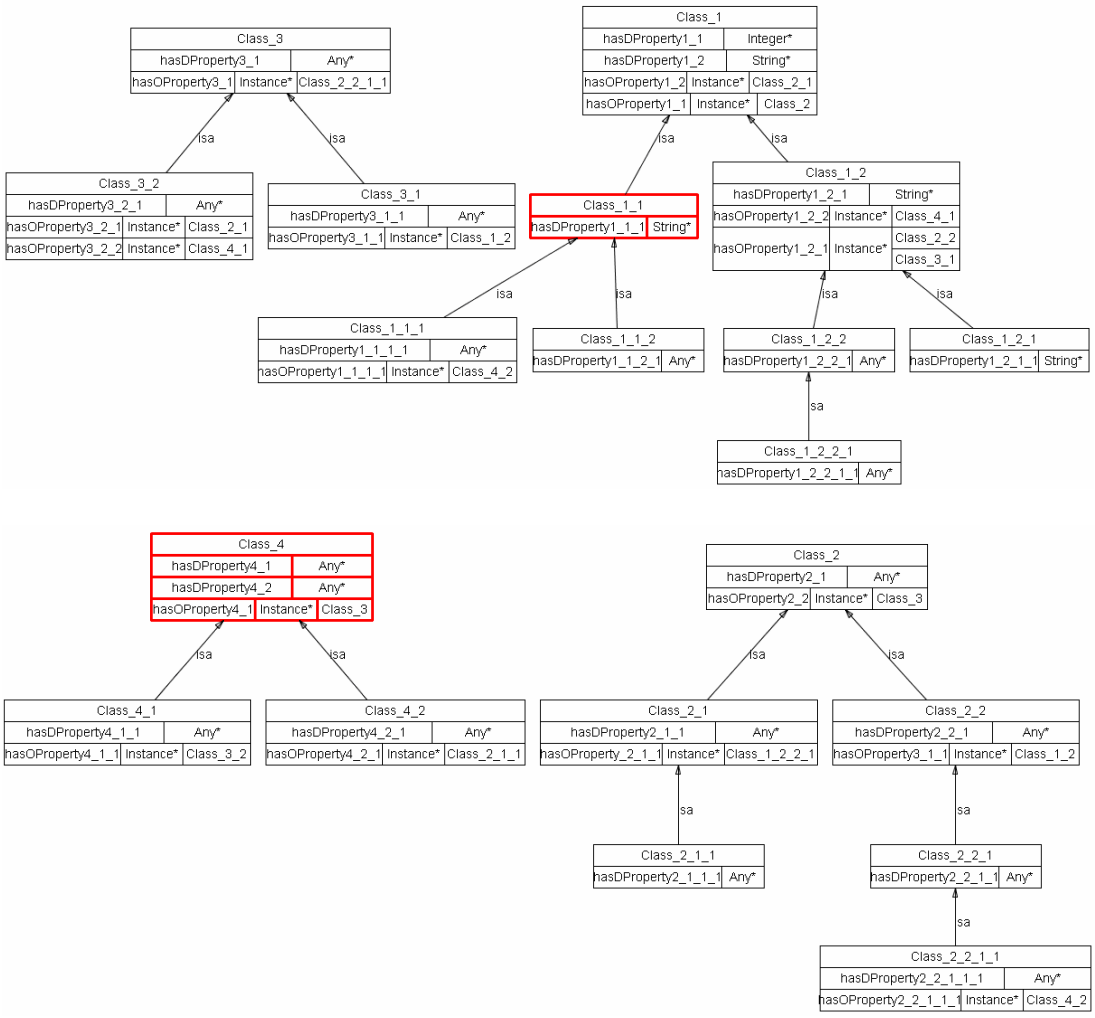


Figure 19: A graphical instance of the general OWL ontology as an IS-A hierarchy with structures that show the ObjectProperties and the DatatypeProperties

SPARQL syntax depends on OWL ObjectProperties that link related OWL Classes. To provide an automatic construction of SPARQL queries we need at any point to define the path that leads from the subject part to the object part of the natural language expression by taking into account the constraints that are declared from the keywords and the relatedness value between the related classes of the ontology. The path connecting the classes directed from the user expression is found by the following algorithm:

Given a connected graph G = (V, E), a weight d: E->R+ and a fixed vertex s in V, find a optimized path from s to each vertex v in V. The optimized path is determined by the highest normalized sum value of the weights of the related concepts.

The differentiation here is that the edges linking the classes of the ontology graph are the objectProperties of the OWL syntax and the weight values are specified by the relatedness measure calculation described earlier in this chapter.

In what follows we present the general algorithm of the OntoNL query representation of domain-ontology disambiguated natural language expression in SPARQL.

OntoNL Query Representation Algorithm

```
Program String SPARQLRepr (List, List, DoubleList)
 List subjOper, objOper, Values, OptPath;
 Double relVal;
 DoubleList SemRelMeas, ListNLStoOnto, ListNLOtoOnto;
String Query, QueryTemplate, OntoSubjTerm, OntoObjTerm,
Begin
QueryTemplate=" PREFIX ins:<ontology_path>"
     "SELECT ?OntoSubjTermIDs "
     "WHERE {?OntoSubjTermIDs rdf:type ?OntoSubjTerm."
 If ListNLOtoOnto.size()=0 && subjOper.size()=0
  OntoSubjTerm = ListNLStoOnto.get(term)
  Query =   QueryTemplate + "}";
 ElseIf ListNLOtoOnto.size()=0 && subjOper.size()!=0
  For all terms i of ListNLStoOnto
  OntoSubjTerm(i) = ListNLStoOnto.getTerm(i)
  Query = QueryTemplate + "}";
 Else
  OntoObjTerm = ListNLOtoOnto.get(term)
  relVal = ListNLOtoOnto.get(relatedness value)
  value = Values.get(not_Disambiguated_Term)
  If objOper.size()=0 && relVal=1
    Query = QueryTemplate +
    "{{?OntoSubjTerm ins:hasObjPropTo ?OntoObjTerm."
    "?OntoObjTerm ins:hasDataProp "value"}"
  ElseIf objOper.size()=0 && relVal!=1
    OptPath = findOptPath(OntoSubjTerm, OntoObjTerm)
    Query = QueryTemplate + "
    For all ObjProperties of OptPath
    "{{?OntoSubjTerm ins:OptPath.get(hasObjProp) ?OntoObjTerm . "
    "?OntoObjTerm ins:hasDataProp "value"}"
```

```
  Else
   If Values.size() = 1
    If relVal=1
     Query = QueryTemplate +
     "{{?OntoSubjTerm ins:hasObjPropTo ?First_Related_Class."
     "?First_Related_Class ins:hasDataProperty ?val1}UNION"
     "{{?OntoSubjTerm ins:hasObjPropTo ?Second_Related_Class."
     "?Second_Related_Class ins:hasDataProp ?val2}"
     "FILTER(?val1 = "value" || ?val2 = "value")"
    Else
     Query = QueryTemplate +
     For all ObjProperties of OptPath
     "{{?OntoSubjTerm ins:OptPath.get(hasObjProp) ?OntoObjTerm."
     "?OntoObjTerm ins:hasDataProp ?val1} UNION"
     "{{?OntoSubjTerm ins: OptPath.get(hasObjProp)
     "?Second_Related_Class ins:hasDataProp ?val2}"
     "FILTER(?val1 = "value" || ?val2 = "value")"
   Else
    For all terms of Values
    If relVal=1
     Query = QueryTemplate +"
     "{{?OntoSubjTerm ins:hasObjPropTo ?First_Related_Class."
     "?First_Related_Class ins:hasDataProp "value1"}UNION"
     "{{?OntoSubjTerm ins:hasObjPropTo ?Second_Related_Class."
     "?Second_Related_Class ins:hasDataProp "value2"}"
    Else
     Query = QueryTemplate +"
     For all ObjProperties of OptPath
     "{{?OntoSubjTerm        ins:        OptPath.get(hasObjProp)
     "?First_Related_Class ins:hasDataProperty "value1"}UNION"
     "{{?OntoSubjTerm        ins:        OptPath.get(hasObjProp)
     "?Second_Related_Class ins:hasDataProp "value2"}"
End
```

Next, we present a number of examples for each case of the table 3 as they find correspondence in the ontology of figures 18, 19.

In the 1st query type of Table 3, were the user has asked for information that was matched to an ontology class (for example Class_1 in figure 14), the query in SPARQL becomes:

PREFIX rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#

PREFIX ins: http://www.owl-ontologies.com/general.owl#

SELECT ?ClassInstancesIDs

WHERE { ?ClassInstancesIDs rdf:type ins: Class_1 };

This query returns the ids of the class's instances. Since we have knowledge about the exact ontology if we prefer instead of id values to receive values of the datatype properties of all instances the query becomes (in a loop):

PREFIX rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#

PREFIX ins:http://www.owl-ontologies.com/general.owl#

SELECT ?value1 ?value2

WHERE { ins: ClassInstanceID ins:hasDProperty1_1 ?value1 ;

ins:hasDProperty1_2 ?value2 }

In case the keyword of the query was matched to a datatypeProperty (for example hasDProperty4_1 of Class_4), the query becomes:

PREFIX rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#

PREFIX ins:http://www.owl-ontologies.com/general.owl#

SELECT ?DataValue

WHERE { ins:class1ID ins:hasDProperty4_1 ?DataValue }

In the 2nd query type of Table 3, were the user has asked for more than one thing that matched to ontology classes separated with an operator (and/or), we present to the user a Boolean OR of the results the query in SPARQL becomes:

PREFIX rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#

PREFIX ins:http://www.owl-ontologies.com/general.owl#

SELECT ?Class1InstancesIDs ?Class2InstancesIDs

WHERE { ? Class1InstancesIDs rdf:type ins: Class1 .

? Class2InstancesIDs rdf:type ins: Class2 }

In case the keywords of the query were matched to dataProperties (for example in Class 3_2 and Class 1_2_2) then:

PREFIX rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#

PREFIX ins:http://www.owl-ontologies.com/general.owl#

SELECT ?DataValue1 ?DataValue2

WHERE {?Class3_2 ins:hasDProperty3_2_1 ?DataValue1 .

?Class1_2_2 ins:hasDProperty1_2_2_1 ?DataValue2 };

The Class3_2 and Class1_2_2 are variables and not the classes. So we do not need to declare their exact names as they are in the ontology.

In the 3rd query type of Table 3, were the user has asked for information that corresponded to a subject – class of the ontology, an object or object complement – class of the ontology and a value that follows the object-class (Class_3_1 and Class_1_2 that are connected via the objectProperty3_1_1 in figure 13), the query in SPARQL becomes:

PREFIX rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#

PREFIX ins:http://www.owl-ontologies.com/general.owl#

SELECT ?ClassInstancesIDs

WHERE { ?ClassInstancesIDs rdf:type ?Class_3_1 .

{{?Class_3_1 ins:hasOProperty3_1_1 ?Class_1_2 .

?Class_1_2 ins:hasDProperty1_2_1“Keyword”} UNION

{ ?Class_1_2 ins:hasOProperty1_2_1 ?Class_2_2 .

?Class_2_2 ins:hasDProperty2_2_1 “Keyword” } };

If the subject of the query is a datatype property (for example of the Class_3_1) then the query becomes:

PREFIX rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#

PREFIX ins:http://www.owl-ontologies.com/general.owl#

SELECT ?DataValue

WHERE { ins:"Class3_1ID" ins:hasDProperty3_1_1 ?DataValue } ;

where the ID of the instance comes from the previous query.

If the object or object complement of the query is an object property (for example, the objectProperty of the Class_1_2 of figure 14) then the keyword characterizes the object property and the query becomes:

PREFIX rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#

PREFIX ins:http://www.owl-ontologies.com/general.owl#

SELECT ?ClassInstancesIDs

WHERE { ?ClassInstancesIDs rdf:type ?Class_3_1 .

?Class_3_1 ins:hasOProperty3_1_1 ?Class_1_2 .

?Class_1_2 ins:hasOProperty1_2_1 ?Class_2_2 .

{?Class_2_2 ins:hasDProperty2_2_1 "Keyword" UNION

{ ?Class_2_2 ins:hasOProperty3_2_1 ?Class_1_2 .

?Class_1_2 ins:hasDProperty1_2_1 "Keyword" }}}

In the 4th query type of Table 3, we can distinguish three different types of queries, based on where the operator applies. We can meet a query like the 3rd query type with more than one objects – classes, that are separated with and/or operators or we can meet a query with one subject – class and multiple object/object complement – keyword pairs separated with

and/or operators or we can meet a query with more than one subjects – classes that are separated with and/or operators and multiple object/object complement – keyword pairs separated with and/or operators. In the first case (for example, Class_3_1, Class_2_2 and Class_1_2 that are connected via the objectProperty3_1_1 in figure 13) the query in SPARQL becomes:

```
PREFIX rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#

PREFIX ins:http://www.owl-ontologies.com/general.owl#

SELECT ?Class3_1InstancesIDs ?Class2_2InstancesIDs

WHERE {  ?Class3_1 InstancesIDs rdf:type ?Class_3_1 .

                ?Class2_2 InstancesIDs rdf:type ?Class_2_2 .

                ?Class_3_1 ins:hasOProperty3_1_1 ?Class_1_2 .

                ?Class_2_2 ins:hasOProperty3_1_1 ?Class_1_2 .

                { {?Class_1_2 ins:hasDProperty1_2_1“Keyword”} UNION

                { ?Class_1_2 ins:hasOProperty1_2_1 ?Class_2_2 .

                  ?Class_2_2 ins:hasDProperty2_2_1 “Keyword” }} };
```

In the second case we need to distinguish two cases: there is only one keyword for all the classes AND/OR object-datatype properties or there are keywords for all the classes AND/OR object/datatype properties. In the first case (for example, Class_1, and Class_3 (and/or) Class_2_1 in figure 14), the query in SPARQL becomes:

```
PREFIX rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#

PREFIX ins:http://www.owl-ontologies.com/general.owl#

SELECT ?Class1InstancesIDs

WHERE { ?Class1InstancesIDs rdf:type ?Class_1 .

        {{ ?Class_1 ins:hasOProperty1_1 ?Class_2 .
```

```
?Class_2 ins:hasOProperty2_1_1 ?Class3 .
?Class_3 ins:hasDProperty3_1 ? value1 } UNION
{?Class_3 ins:hasOProperty3_1 ?Class_2_2_1_1 .
?Class_2_2_1_1 ins:hasDProperty2_2_1_1_1 ?value1}}} UNION
{{{?Class_1 ins:hasOProperty1_2 ?Class_2_1 .
?Class_2_1 ins:hasDProperty2_1_1 ?value2} UNION
{?Class_2_1 ins:hasOProperty2_1_1 ?Class_1_2_2_1 .
?Class_1_2_2_1 ins:hasDProperty1_2_2_1_1 ?value2}} }}
FILTER (?value1 = “Keyword” || ?value2 = “Keyword”)};
```

In the second case:

```
PREFIX rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#
PREFIX ins:http://www.owl-ontologies.com/general.owl#
SELECT ?Class1InstancesIDs
WHERE { ?Class1InstancesIDs rdf:type ?Class_1 .
{{ ?Class_1 ins:hasOProperty1_1 ?Class_2 .
?Class_2 ins:hasOProperty2_1_1 ?Class3 .
?Class_3 ins:hasDProperty3_1 “Keyword1” } UNION
{?Class_3 ins:hasOProperty3_1 ?Class_2_2_1_1 .
?Class_2_2_1_1 ins:hasDProperty2_2_1_1_1 “Keyword1”}}} UNION
{{{?Class_1 ins:hasOProperty1_2 ?Class_2_1 .
?Class_2_1 ins:hasDProperty2_1_1 “Keyword2”} UNION
{?Class_2_1 ins:hasOProperty2_1_1 ?Class_1_2_2_1 .
```

?Class_1_2_2_1 ins:hasDProperty1_2_2_1_1 “Keyword2”}} }} };

In the third case we assume that the objects/object complements refer to all the subjects and we have a unique keyword for every object/object complement, (for example, Class_3_1 (and/or) Class_2_2 and Class_1_2 (and/or) Class_4_1 in figure 14), so the query in SPARQL becomes:

PREFIX rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#

PREFIX ins:http://www.owl-ontologies.com/general.owl#

SELECT ?Class3_1InstancesIDs ?Class2_2InstancesIDs

WHERE {

{

{ ?Class3_1InstancesIDs rdf:type ?Class_3_1 .

?Class2_2InstancesIDs rdf:type ?Class_2_2 .

?Class_3_1 ins:hasOProperty3_1_1 ?Class_1_2 .

?Class_2_2 ins:hasOProperty3_1_1 ?Class_1_2 .

?Class_1_2 ins:hasOProperty1_2_2 ?Class_4_1.

{ {?Class_4_1 ins:hasOProperty4_1_1 “Keyword”} UNION

{ ?Class_4_1 ins:hasOProperty4_1_1 ?Class_3_2 .

?Class_3_2 ins:hasDProperty3_2_1 “Keyword” ;

ins:hasDProperty3_2_2 “Keyword” }}}} UNION

{{ ?Class3_1InstancesIDs rdf:type ?Class_3_1 .

?Class2_2InstancesIDs rdf:type ?Class_2_2 .

?Class_3_1 ins:hasOProperty3_1_1 ?Class_1_2 .

?Class_2_2 ins:hasOProperty3_1_1 ?Class_1_2 .

{ {?Class_1_2 ins:hasDProperty1_2_1“Keyword”} UNION

{ ?Class_1_2 ins:hasOProperty1_2_1 ?Class_2_2 .

?Class_2_2 ins:hasDProperty2_2_1 “Keyword” }}}}};

In the 5th query type of Table 3, the disambiguation cannot be confronted and the only information that we have is the grammatical position of the keyword inside the sentence (subject). The query in SPARQL is:

PREFIX rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#

PREFIX ins:http://www.owl-ontologies.com/general.owl#

SELECT ?x

WHERE { ?x <http://www.owl-ontologies.com/general.owl#> ?z .

FILTER regex (?z, "keyword", "i")}

In a more detailed expression of SPARQL we can determine that the keyword refers to an instance of a class - the subject of the user query:

PREFIX rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#

PREFIX ins:http://www.owl-ontologies.com/general.owl#

SELECT ?ClassInstanceID

WHERE { ?ClassInstanceID rdf:type ins:ClassX .

?ClassX ins:hasDPropertyX ?z .

FILTER regex (?z, "keyword", "i")}

The 6th query type of Table 3 is the exact same with the 7th with the difference that we have more than one keywords in the subject that are separated with and/or operators. The SPARQL query becomes:

PREFIX rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#

PREFIX ins:http://www.owl-ontologies.com/general.owl#

SELECT ?x

WHERE { ?x <http://www.owl-ontologies.com/general.owl#> ?z .

FILTER regex ((?z, "keyword1", "i") || (?z, "keyword2", "i")…)}

In a more detailed expression of SPARQL we can determine that the keywords refer to class instances - the subjects of the user query:

PREFIX rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#

PREFIX ins:http://www.owl-ontologies.com/general.owl#

SELECT ?ClassInstanceID

WHERE { ?ClassInstanceID rdf:type ins:ClassX .

?ClassX ins:hasDPropertyX ?z .

FILTER regex (?z, "keyword", "i"|| (?z, "keyword2", "i")…)}

In the 7th query type of Table 3, the syntactic analysis concluded to a schema with a subject and one or more object/object complements but the semantic disambiguation using the disambiguation ontology did not help by mapping the keywords to ontological structures. So, the SPARQL query becomes:

PREFIX rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#

PREFIX ins:http://www.owl-ontologies.com/general.owl#

SELECT ?DataValue

WHERE { ins:"Class_X" ins:hasDPropertyX_X ?DataValue }

?Class_X ins:hasOPropertyX_X ?Class_X' .

{?Class_X' ins:hasDPropertyX'_X "Keyword" UNION

{ ?Class_X' ins:hasOPropertyX'_X ?Class_X'' .

?Class_X'' ins:hasDPropertyX''_X "Keyword" }}}

In the 8th query type of Table 3, we find again the grammatical relations as in the 9th query with no semantic disambiguation. We distinguish again the three types of query based on the operator application: the keywords – subjects are separated with and/or operators and share common object/object complement or the subject is unique and we have multiple objects/object complements – keywords separated with and/or or the combination of the two previous approaches. We are not going to present the corresponded SPARQL queries because of the level of ambiguity.

Summary

In this chapter we have presented the semantic ambiguity of a natural language expression in the OntoNL Framework and an algorithm to confront with different levels of ambiguity based on OWL Ontologies. We presented a brief overview of semantic relatedness algorithms and we developed a new semantic relatedness measure for OWL ontologies.

We have presented the OntoNL ontology-driven semantic ranking methodology for ontology concepts used for natural language disambiguation. The methodology uses domain specific ontologies for the semantic disambiguation. The ontologies are processed offline to identify the strength of the relatedness between the concepts. Strongly related concepts lead to higher ranked pairs of results during disambiguation. The disambiguation procedure is automatic and quite promising, since it is enhanced with information based on the domain that the request refers to. It is easily reusable in many domains since the only restrictions are the used language (English) and OWL as the standard language for representing ontologies of a specific domain. The OntoNL semantic ranking methodology depends on the OntoNL semantic relatedness measure for OWL domain ontologies. The

relatedness value computation is based on the following factors: the commonality (based on the semantic relations and the conceptual distance) and the related senses.

The motivation of this work came from the absence of a general, domain-independent Natural Language Interface Generator with good results in the Natural Language Disambiguation process. The disambiguation process depends on the domain ontologies and when necessary, the OntoNL Semantic Relatedness Measure is used to rank ontological, grammatically-related concepts. We have developed a semantic relatedness measure over OWL ontologies that is general, domain independent and covers the lack of a systematic way for calculating asymmetric semantic relatedness of concepts.

Overall, we state that the semantic relatedness measure that leads to the ontology-based semantic ranking of concepts for natural language disambiguation is quite complete and shows very good results (see Chapter 7).

Also, we used the formal query language SPARQL to form the disambiguated queries. We choose SPARQL (http://www.w3.org/TR/rdf-sparql-query/) as the query language to represent the natural language queries after the syntactic and semantic disambiguation since SPARQL is defined in terms of the W3C's RDF data model and works for any data source that can be mapped into RDF.

In the next chapter we are going to present the implementation of a specific application in the domain of soccer.

Chapter 6

Implementation of the OntoNL Framework

In this chapter we introduce the general architecture of the OntoNL and the description of the implementation of the components that comprise the architecture. A specific application in question answering for the domain of soccer has been developed and the details of the implementation are presented afterwards. We conclude with an example of using the specific question answering system where the data flow is presented.

The OntoNL Infrastructure

The parts of the platform that constitute the Natural Language Interface framework have been discussed in detail, previously in sections 3, 4 and 5. Their implementation details follow. The system infrastructure is depicted in Figure 20.

The OntoNL framework has five core modules for processing the natural language information. These are presented separately with their implementation details.

The Linguistic Analysis Component

As we have presented in Chapter 4, in the Linguistic Analysis Component we have developed a specific methodology to deal with syntactic ambiguities. This methodology is a combination of word sense disambiguation techniques modified to serve best the needs of a system for natural language interactions like the OntoNL system.

The Linguistic Analysis component consists of a sentence conversion mechanism, a part-of-speech tagger, a noun compound detector, a typed-dependencies producer and a module responsible for the synonyms and sense discovery using the word ontology, WordNet.

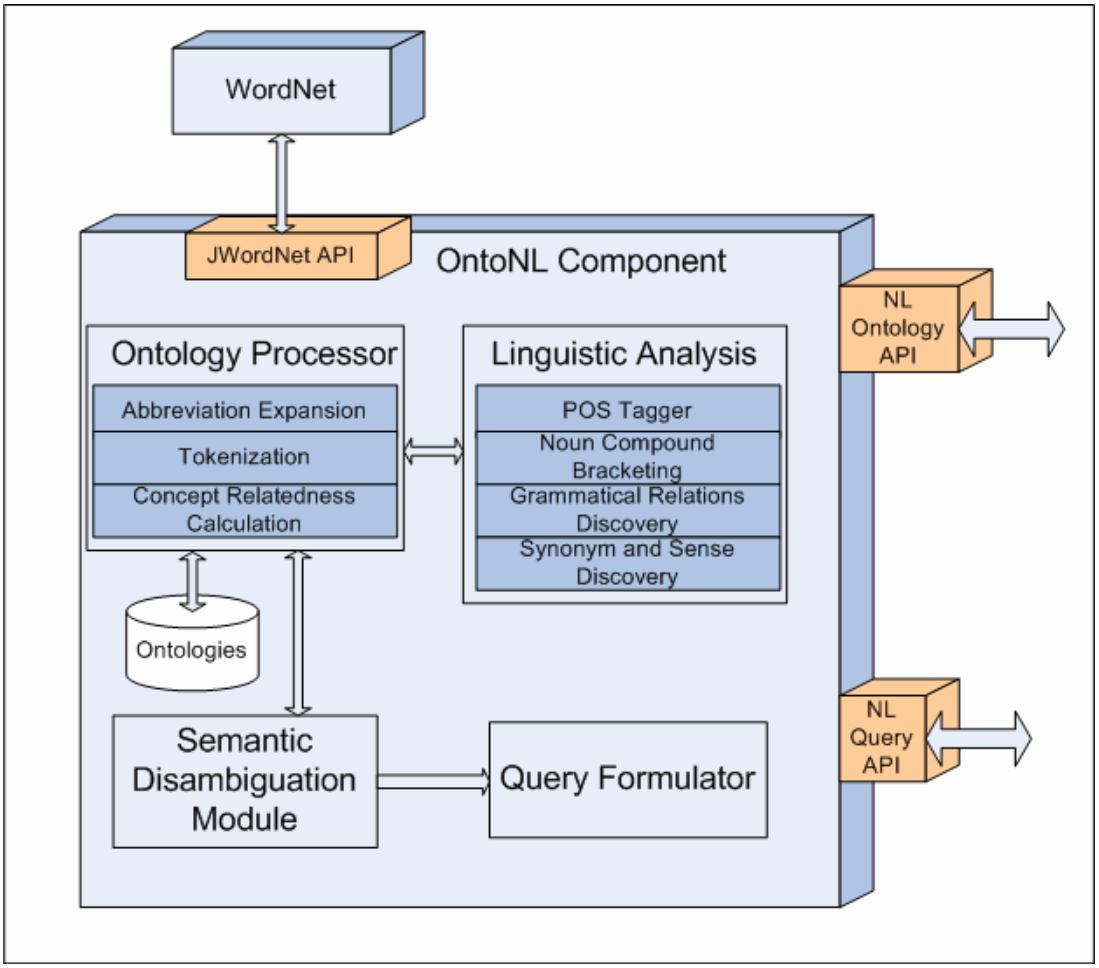


Figure 20: The OntoNL Infrastructure

For the request conversion mechanism, there is an effort to eliminate the first words that a request or a question may have because, the words that do not contain semantics for the retrieval of information from a repository. We distinguish 3 different types of requests and questions in which we have a different approach when dealing with the information of the utterance:

1. Requests for metadata (ex. Show me the goals scored in the game between Italy and France)

2. WH-questions (ex. What was the score in the game between Italy and France?)

In the literature (http://www.eslgold.com/grammar/wh_questions.html) we find that the wh-questions testify the subject of the request according to the type of input. We present in what follows the semantics of request of each different type of wh-question.

When?	Time
Where?	Place
Who?	Person
Why?	Reason
How?	Manner
What?	Object/Idea/Action
Which (one)?	Choice of alternatives
Whose?	Possession
Whom?	Person (objective formal)
How much?	Price, amount (non-count)
How many?	Quantity (count)
How long?	Duration
How often?	Frequency
How far?	Distance
What kind (of)?	Description

3. Yes/No questions (ex. Were there any goals in the game between Italy and France?)

In the input conversion mechanism we identify the type of the input and we use an indicator to distinguish the three different types. After the conversion, the input becomes like the one's in the examples below:

- the goals scored in the game between Italy and France
- the score in the game between Italy and France
- any goals in the game between Italy and France

The grammatical dependencies (subject, object, verb, complements) of those converted sentences have the semantics that we retrieve by interacting with the application-domain ontology. This information enhances the OntoNL Language Model.

For implementing the tagger we used the software provided by the Stanford Natural Language Processing Group (http://nlp.stanford.edu/software/tagger.shtml). The tagger is licensed under the GNU GPL. The main class **edu.stanford.nlp.tagger.maxent.MaxentTagger** is a class for end users to part of speech tag text using an already trained and saved maximum entropy tagger. We tag the modified natural language expression through the Java API. We used the one from the two taggers included in the distribution, a bi-directional dependency network tagger, with accuracy 97.24% on Penn Wall Street Journal. A MaxentTagger is made with a constructor taking as argument the location of parameter files for a trained tagger. Alternatively, a constructor with no arguments can be used, which reads the parameters from a default location.

The choices we had were:

To tag a string of words and get a string of tagged words.

String taggedString = maxentTagger.tagString("Here's a tagged string.")

To tag a Sentence and get a TaggedSentence

Sentence taggedSentence=maxentTagger.tagSentence(Sentence sentence)

Sentence taggedSentence=maxentTagger.apply(Sentence sentence)

To tag a list of sentences and get back a list of tagged sentences

List taggedList=maxentTagger.process(List sentences)

Here is an example of using the static tagString method. The MaxentTagger can be initialized using a static call init that takes in a trained model, which is loaded immediately (takes a long time...); subsequent attempts to initialize the tagger will be no-ops, therefore it is safe to call init ad infinitum. Otherwise the default trained model is used. Then subsequent calls to tagString can be executed, passing in an untagged String; a tagged String is returned, unless there was a serious problem in the Tagging machinery, in which case null is returned.

Example:

MaxentTagger.init("stanford-tagger/bidirectional/wsj0-18.holder");

String taggedString = MaxentTagger.tagString("Here's a tagged string.");

String taggedString2 = MaxentTagger.tagString("This is your life.");

The output is

Here's/JJ a/DT tagged/VBD string./NNP

and

This/DT is/VBZ your/PRP$ life./NN respectively.

The problem of dealing with noun compounds was discussed in Chapter 4 in detail. The problem with applying lexical association to noun compounds is the enormous number of parameters required one for every possible pair of nouns. This leads to the need of a vast amount of memory space and to the severe data sparseness problem. We used the following equation, for the dependency model and a given compound of w_1, w_2, w_3 , that gives the estimation of the ratio

$$R_{dep} = \frac{\sum_{t_i \in cats(w_i)} P(t_1 \to t_2) P(t_2 \to t_3)}{\sum_{t_i \in cats(w_i)} P(t_1 \to t_3) P(t_2 \to t_3)}$$

where t_1, t_2 and t_3 are conceptual categories in a taxonomy or thesaurus, and the nouns $w_1,\ldots,w_n$ are members of these categories. For a correct result we must sum over all possible categories for the words in the compound. In any case, the estimation of probabilities over concepts reduces the number of model parameters. Especially in our case, the conceptual categories are extracted by the WordNet, (hyponyms).

In contrast with other methods we use a method to expand n-grams into all morphological forms by the use of morphological tools [Minnen, et. al., 2001]. For example, if we have a bigram ‘player scores’, then we create a list of all possible forms: ‘player scores’, ‘player score’, ‘players score’, etc. Based on the successful performance of the dependency model over the adjacency we adopt the use of it. The innovation in this work is that the domain

ontologies used for every different domain constitute the training corpus. This may lead to the conclusion that the test set is very limited in comparison to a linguistic corpus, but it is more specific to the needs of the application. We are interested in the particular needs of the user based on a specific domain. Since we use the noun compound bracketing methodology to be more accurate when dealing with the user request's ambiguities we use as a test set the noun compounds that may appear in the exact user request and as a training corpus the total of domain ontologies used.

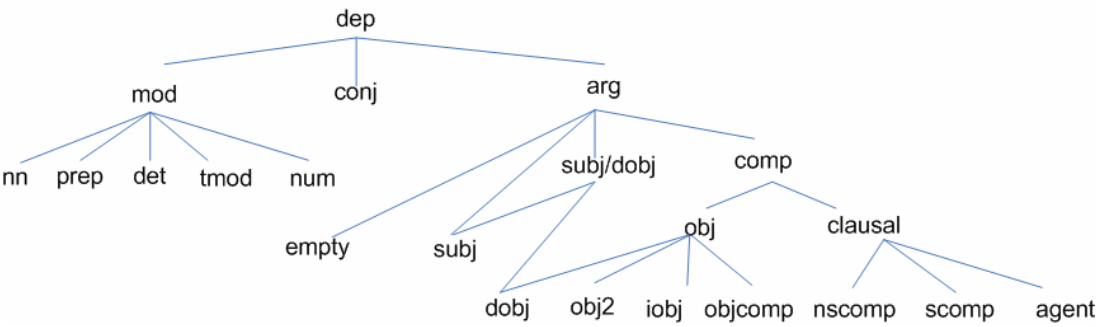


Figure 21: The grammatical relation hierarchy

For the grammatical relationship detection, we first distinguished 8 different clause types and then constructed an annotation scheme for locating grammatical relations. We used a rich set of grammatical relations that provide detailed information about syntactic relationships. When the relation between a head and its dependent can be identified more precisely, relations further down in the hierarchy can be used. For example, the dependent relation can be specialized to aux (auxiliary), arg (argument), or mod (modifier). The arg relation is further divided into the subj (subject) relation and the comp (complement) relation, and so on. The scheme is summarized in figure 21:

The method that we developed and implemented typed dependencies is essentially based on rules – or patterns – applied on phrase structure trees. The method is general, but requires appropriate rules for each language and treebank representation. Here we present details only for Penn Treebank English. The method for generating typed dependencies has two phases: **dependency extraction** and **dependency typing**. In the dependency extraction phase first, a sentence is parsed with a phrase structure grammar parser. We used the Stanford Parser [Klein and Manning, 2003], a high accuracy statistical phrase

structure parser trained on the Penn Wall Street Journal Treebank. The head of each constituent of the sentence is then identified, using pattern rules according to the Collins head rules [Collins, 1999].

In the second phase, we label each of the dependencies extracted with a grammatical relation which is as specific as possible. For each grammatical relation, we define one or more patterns over the phrase structure parse tree (using the tree-expression syntax defined by tregex [Levy and Andrew, 2006].

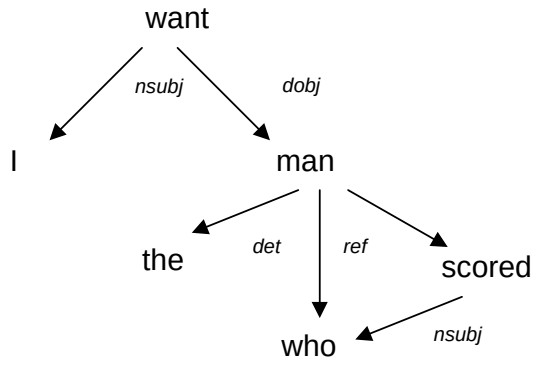


Figure 22: An example of a typed dependency parse for the sentence "I want the man who scored"

For the synonyms and sense discovery we used the word ontology WordNet and in particular we used the JWordNet API, a pure Java standalone object-oriented interface to the WordNet database of lexical relationships. We provide a simple example of usage:

```
import edu.brandeis.cs.steele.wn.*;

/* This prints the senses of the noun 'dog' to the console. */
public class Main {
  static void main(String[] args) {
  DictionaryDatabase dictionary = new FileBackedDictionary();
  IndexWord word = dictionary.lookupIndexWord(POS.NOUN, "dog");
    Synset[] senses = word.getSenses();
    int taggedCount = word.getTaggedSenseCount();
    System.out.print("The " + word.getPOS().getLabel() + " " +
word.getLemma() + " has " + senses.length + " sense" +
(senses.length == 1 ? "" : "s") + " ");
    System.out.print("(");
    if (taggedCount == 0) {
      System.out.print("no senses from tagged texts");
    } else {
      System.out.print("first " + taggedCount + " from tagged
texts");
```

```
        }
        System.out.print(")\n\n");
        for (int i = 0; i < senses.length; ++i) {
          Synset sense = senses[i];
          System.out.println("" + (i + 1) + ". " +
    sense.getLongDescription());
        }
      }
    }
```

The procedure of the linguistic analysis concludes to a structure that was also presented in Chapter 4, the language model.

The Ontologies Processing Component

The Ontologies Processing Component is responsible for two main tasks, the representation of the ontology in a more natural language way (naming conversions) and the detection of semantic relatedness between the concepts of the ontology.

The first task includes the tokenization process and the abbreviation expansion of names. To tokenize words, we find word boundaries in a multi-term word attribute using changes in font, presence of delimiters, etc. The OntoNL CheckAbbreviation algorithm uses a custom abbreviation dictionary, from the TypFast abbreviation expander (http://www.topshareware.com/TypFast-download-3297.htm).

The second task includes the calculation of the relatedness measure value between the concepts of the ontology. For implementing the measures we used the a part of the GraphOnto API, the OWL API and the JWordNet API. For the first measure we retrieve the object properties and we calculate their number and the number of inverse object properties for each class inside the ontology (by using functions from the GraphOnto API and the OWL API). By this way we can calculate the number of the common properties and of the inverse properties two concepts share.

$$rel_{prop}(c_1, c_2) = (f_1 \times \frac{\sum_{i=1}^{n} p_{ijk}}{\sum_{i=1}^{n} p_{ij}}) + (f_2 \times \frac{\sum_{i=1}^{n} p_{invijk}}{\sum_{i=1}^{n} p_{ijk}}),$$

were $f_1 \geq f_2$ and $f_1 + f_2 = 1$.

The second measure we developed concerns the conceptual distance between two classes in the ontology. The two factors we took into account were the path distance and the specificity. For implementing this measure we again used the GraphOnto API and the OWL API that provided us the functions of retrieving knowledge for the position and the semantics of each class inside the ontology. The equation of the path distance is:

$$\forall d_{C1} \geq 1, d_{C2} \geq 1, d_{C1} + d_{C2} > 2 : pathDist(c_1, c_2) = \frac{d_{C1} + d_{C2}}{2 * D} \in (0,1]$$

The specificity measure is described by the equation:

$$w1_{spec_{C1}} = \begin{cases} -\log \frac{2 \times d_{C1}}{d_{C1} + d_{C2}} \in (0,1], \text{ if } d_{C1} < \frac{d_{C1} + d_{C2}}{2} \\ 0, \qquad \text{if } d_{C1} \geq \frac{d_{C1} + d_{C2}}{2} \end{cases}$$

The equation for counting the specialization of the concept – c1 based on the object properties of its subsumer is:

$$\forall \#ObjP_{C1} \leq 10 \times \#ObjP_S : w2_{spec_{C1}} = 1 - \log \frac{\#ObjP_{C1}}{\#ObjP_S} \in [0,1] \quad (31),$$

$$\text{else } w2_{spec_{C1}} = 0 \quad (32)$$

and the overall measure is:

$$rel_{CD} = (w1_{specC1} + w2_{specC1} + 1 - pathDist(c_1, c_2))/3 \quad (33)$$

For the third measure, we used the JWordNet API and the part-of-speech tagger (Stanford POS Tagger). The JWordNet provided the synonyms and the descriptions of each noun and then by filtering the descriptions from the pos tagger we obtained the nouns and completed the description set of each class. In the set we also include the initial noun – class name. To remind the reader, the measure that calculates the amount of related senses is

$$rel_{RS}(c_1, c_2) = \frac{\left|S_1 \cap S_2\right|}{|S_1 \cap S_2| + \left|S_1 \setminus S_2\right|},$$

where S_1 is the description set of senses for c_1 and S_2 the description set of senses for c_2.

The Semantic Disambiguation Component

We next describe the Semantic Disambiguation Procedure as it was presented in Chapter 5, with an initial and limited pseudo code.

```
Program OntologyStructure SemanticDisambiguation (List, DoubleList)
    List subjOper, objOper, Ambiguity, verbOper, ToOntoStruct;
    LanguageModel LangModel;
    OWLOntology DomOnto;
    OntologyStructure ontoStruct;
    Double relVal;
    DoubleList SemRelMeas, ListNLStoOnto, ListNLOtoOnto;
    String  NLSubjTerm,  NLObjTerm,  string,  related,  conceptSName,
    conceptOName, instances, oper, conceptS, conceptO;

Begin

For each term NLSubjTerm of LangModel do
   conceptSName = FindConceptMatching(DomOnto)
   oper = FindOperators(NLSubjTerm)
   subjOper.add(oper)
   If conceptSName != null
      ListNLStoOnto.add(conceptSName)
   else
      If NLSubjTerm.size() = 1 && subjOper.size() = 0
         Ambiguity.add(NLSubjTerm)
      Else If NLSubjTerm.size() > 1 & subjOper.size() = 0
         instances = 'different'
         Ambiguity.add(NLSubjTerm)
      Else
         instances = 'different'
         Ambiguity.add(NLSubjTerm)

For each term NLObjTerm of LangModel do
   conceptOName = FindConceptMatching(DomOnto)
   oper = FindOperators(NLObjTerm)
   objOper.add(oper)
   If conceptOName != null
      ListNLOtoOnto.add(conceptOName)
   else
      If NLObjTerm.size() != 1 && objOper.size() = 0
         Ambiguity.add(NLObjTerm)
      Else If NLObjTerm.size() = 1 & objOper.size() = 0
         instances = 'different'
```

```
            Ambiguity.add(NLObjTerm)
         Else
            instances = 'same'
            Ambiguity.add(NLObjTerm)

   If Ambiguity.size()= 0
      relVal = 1
      For each term of ListNLStoOnto do
         ToOntoStruct.add(ListNLStoOnto(getTerm(i)))
         ToOntoStruct.add(relVal)
         ontoStruct.add(ToOntoStruct)
         ToOntoString.removeAll()
      For each term of ListNLOtoOnto do
         ToOntoStruct.add(ListNLOtoOnto(getTerm(i)))
         ToOntoStruct.add(relVal)
         ontoStruct.add(ToOntoStruct)
         ToOntoString.removeAll()
   Else
      If ListNLStoOnto != null && ListNLOtoOnto == null
         For each term conceptS of ListNLStoOnto do
            string = ListNLStoOnto.get(concept)
            ToOntoStruct.add(string)
            relVal = 1
            ToOntoStruct.add(relVal)
            OntoStruct.add(ToOntoStruct)
            ToOntoString.removeAll()
            related = SemRelMeas.getMostRelated(string)
            ToOntoStruct.add(related)
            relVal = SemRelMeas.getRelValue(string, related)
            ToOntoStruct.add(relVal)
            OntoStruct.add(ToOntoStruct)
            ToOntoString.removeAll()
      If ListNLStoOnto == null && ListNLOtoOnto != null
         For each term conceptO of ListNLOtoOnto do
            string = ListNLOtoOnto.get(concept)
            ToOntoStruct.add(string)
            relVal = 1
            ToOntoStruct.add(relVal)
            OntoStruct.add(ToOntoStruct)
            ToOntoString.removeAll()
            related = SemRelMeas.getMostRelated(string)
            toOntoStuct.add(related)
            relVal = SemRelMeas.getRelValue(string, related)
            ToOntoStruct.add(relVal)
            OntoStruct.add(ToOntoStruct)
            ToOntoString.removeAll()
      If ListNLStoOnto != null && ListNLOtoOnto != null
            For each term conceptO of ListNLOtoOnto do
            string = ListNLOtoOnto.get(concept)
            related = SemRelMeas.getMostRelated(string)
            toOntoStuct.add(related)
            relVal = SemRelMeas.getRelValue(string, related)
            ToOntoStruct.add(relVal)
            OntoStruct.add(ToOntoStruct)
            ToOntoString.removeAll()
End
```

Steps of the OntoNL Disambiguation Algorithm

1.

1.1. Search and match the query terms to the concepts and relations of the ontology.

1.2. Assign the terms that follow the matched concepts of the ontology as concept instances.

1.3. Create a language model with the correspondence of the query terms to the subject concept, the object/object complement concepts, relations and keywords and assign the weight value 1 to the structure.

This structure is an enhanced language model with the structural and semantic dependencies that derive from the ontological structures.

2.

2.1. The Subject Part consists of one word that cannot be matched with the concepts of the domain ontology

2.1.1. Search for a number specified by the application of ontology concepts that have the greatest relatedness value with the disambiguated term of the request.

2.1.2. Create a list of instances of the language model like step 1.3 for each pair of concepts from step 2.2.1. Assign the structure with the relatedness measure value.

2.2. The object part consists of one word that cannot be matched with the concepts of the domain ontology or

2.3. The object part consists of more than one word, and they are not separated with any other words. We gather the words in one string that comprises the query term to address the repository.

2.3.1. Search for a set of the strongest semantic related concepts to the concept of the subject.

2.3.2. Create a list of instances of the language model like step 1.3 for each pair of concepts from step 2.2.1. Assign the structure with the relatedness measure value.

2.4. The user query contains more than one query term (many words), that are separated with other words (we filter from a stop word file the in between the terms words to cut the less significant ones).

2.4.1. Check for the existence of operators AND-OR between the query terms. If there is an AND or OR between them consider that the concept the instances belong cannot be different for each query that the system creates.

2.4.2. Do Steps 2.2.1 and 2.2.2 with step 2.3.1 as constraint.

2.4.3. If there is no AND or OR between them consider that the concept the instances belong must be different for each query that the system creates.

2.4.4. Do Steps 2.2.1 and 2.2.2 with step 2.3.3 as constraint.

3. Create a query with the terms of the sentence connected with a Boolean AND and queries with the term of the subject connected with Boolean OR to the terms of the object.

The Query Formulation Component

In Chapter 5 we have presented the way the natural language queries after the disambiguation phase are translated to SPARQL queries. In order to write the SPARQL queries we used Jena [http://jena.sourceforge.net/]. Jena is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine. Jena is open source and grown out of work with the HP Labs Semantic Web Programme.

The Jena Framework includes:

- A RDF API
- Reading and writing RDF in RDF/XML, N3 and N-Triples
- An OWL API
- In-memory and persistent storage
- SPARQL query engine

We used the javadoc to guide the implementation part. http://jena.sourceforge.net/ARQ/javadoc/index.html

The package com.hp.hpl.jena.query is the main application package. The Query Class is a class that represents the application query. It is a container for all the details of the query. Objects of class Query are normally created by calling one of the methods of QueryFactory methods which provide access to the various parsers.

The basic steps in making a SELECT query are outlined in the example below. A query is created from a string using the QueryFactory. The query and model or RDF dataset to be queried are then passed to QueryExecutionFactory to produce an instance of a query execution.

In this example we show how to obtain results from the execution of a SPARQL query. This is not necessary since as we discussed in Chapter 5 we use SPARQL to structure a query interface between the Natural Language Interface generator and a knowledge repository.

To capture the information we need in order to syntax the SPARQL queries we use the GraphOnto API and the OWL API. These APIs provide us access to the structure and the semantics of OWL ontologies.

```
import com.hp.hpl.jena.query.* ;

Model model= ... ;
String queryString= " .... " ;
Query query=QueryFactory.create(queryString);
QueryExecution qexec=QueryExecutionFactory.create(query, model);
try {
  ResultSet results=qexec.execSelect();
```

```
  for ( ; results.hasNext(); )
  {
    QuerySolution soln=results.nextSolution();
    RDFNode x=soln.get("varName");//Get a result variable by   name
    Resource r=soln.getResource("VarR"); //Get a result variable -
must be a resource
    Literal l=soln.getLiteral("VarL"); //Get a result variable -
must be a literal
  }
} finally { qexec.close(); }
```

The Result Processing Component

The OntoNL infrastructure can manipulate results from knowledge bases but the format of the results retrieved from a knowledge repository is application specific. It is proportional to the type of content and service the knowledge repository provides and to the device that the results are displayed.

If the format of the results is textual (data or metadata), we propose a way of presenting them to the user in a natural language based manner. From the syntactic disambiguation procedure we have concluded to a language model (see Chapter 4) were the grammatical relationships have been identified. A nice way to present the results to the user is by enhance the answer with a template message that will contain the subject of the query. For example if the user query was "*Give me the players of Milan*" then a user friendly answer will be "*The players are …*".

A more appropriate message to the user will be the one that will distinguish the categories of results retrieved by the repository. For example if the user query was "*Give me the players that participated in the game between Milan and Barcelona*" the most appropriate answer will be "*The players that participated in the game between Milan and Barcelona from Milan are … and from Barcelona are…* ". This second case is also tied to the repository structure and its ability to provide such information.

If the repository offers audiovisual information then the results to display to the user are video segments. The video segments must be provided initially as links to the server that

contains the segments and then the user to download them and reproduce them in a media player.

In case the request was of type that the answer is **yes** or **no** (Request Type 2: see Chapter 4) then we propose that the most appropriate reply would be the one that in the case of a positive answer (yes) the retrieved results should also be presented to the users since the system tries to avoid dialogues and wants to be fast and accurate. Other way the answer is just a negative answer (no).

A Natural Language Interface to a Knowledge Repository can also be used as a service to provide information in a more complicated system. In such a case, the information needed it can be urls or xml documents. The repository and the goal of use of the interface in a particular application will determine the format of the information exchange.

The device for the results to be displayed also makes a lot of difference. When we have as a front end, a handheld device (mobile phones, PDAs) the display is very limited. From previous applications [Karanastasi et. al, 2004] we have seen that the notion of ranking receives a bigger importance when dealing with small devices. If the results are textual we need to cut off the user friendly messages and present to the user as less detailed as it can get the information he/she asked. In case the user needs more details we ought to provide a service for further details of the presented information. If the results are video segments we need to provide the user a link from where he/she can download the specific segment.

NL Query API and NL Ontology API

The OntoNL Framework provides support for Natural Language disambiguation in knowledge repositories. The NL Query API takes as input a natural language query and after the disambiguation outputs a number of weighted SPARQL queries, based on the ontologies used for the disambiguation. It implements functions for the data transfer between the framework and the repository.

The NL Ontology API consists of the total of functions used for manipulating the ontologies that interfere with the system. It implements functions like *insertOntology*( ),

storeOntology( ), *getClasses*( ) and a number of other functions responsible for accessing the information about the structure and the semantics of the disambiguation ontologies.

Implementation of an Application in the domain of Soccer

In order to test the OntoNL, we developed an application for the domain of soccer [Zwtos, 2007]. The overall architecture is shown in figure 4. The reference ontology we used is an application of the he DS-MIRF ontological infrastructure [Tsinaraki et. al., 2006] and the WordNet for the syntactic analysis. The repository for accessing the instances is the DS-MIRF Metadata Repository [Tsinaraki et al, 2006].

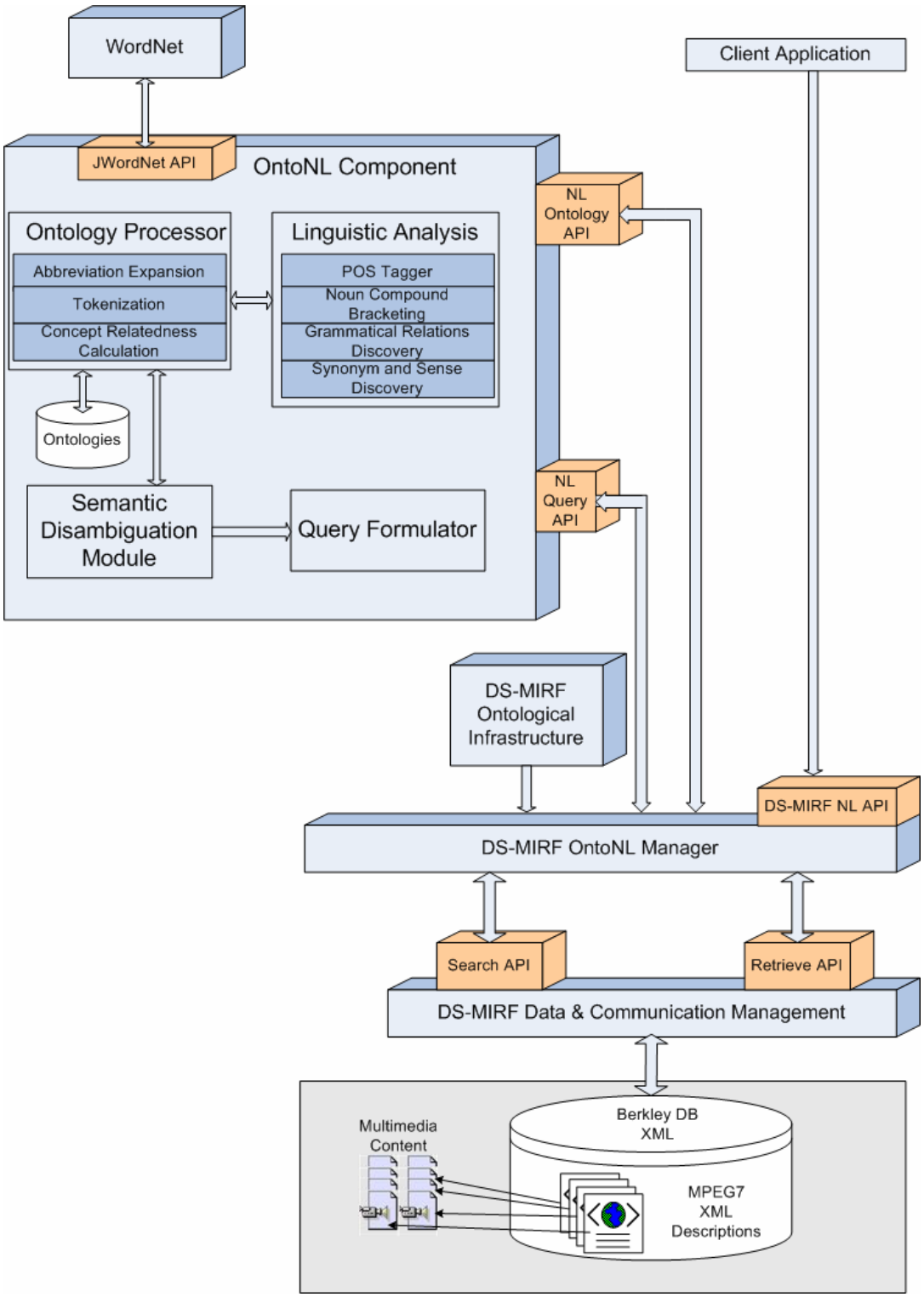


Figure 23: The Architectural Representation of the Application

The DS-MIRF Ontological Infrastructure

The DS-MIRF ontological infrastructure [Tsinaraki et. al, 2004] is shown in Figure 24 and includes:

- An *OWL Upper Ontology* that fully captures the MPEG-7 MDS and the MPEG-21 DIA Architecture, which is the cornerstone of the ontological infrastructure of the DS-MIRF framework and the basis for interoperability between OWL and MPEG-7/21.

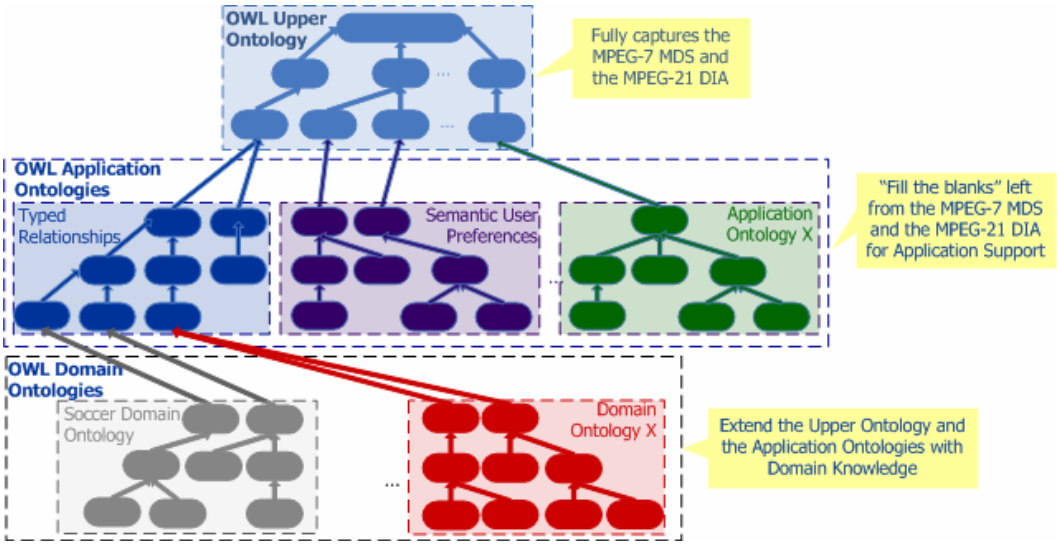


Figure 24: The ontological Infrastructure

- A set of *OWL Application Ontologies* that provide additional functionality in OWL that either makes easier for the user the use of the MPEG-7/21 (usually constructs implied in the MPEG-7/21 text like the typed relationships) or supports advanced multimedia content services (like, for example, semantic user preferences). The Application Ontologies provide general-purpose constructs that are either implied in the text of MPEG-7/21 (but missing in the syntax) or not available in MPEG-7/21.

An application ontology that contains a set of extensions for the MPEG-7 MDS has been developed, which allows the full representation of typed relationships that are literally described in the MPEG-7 MDS text but their features are not fully captured in the MPEG-7 MDS syntax. An application ontology for the description of semantic user preferences for multimedia content has also been developed, as the MPEG-7/21 user preference descriptions allow keyword-only descriptions of the semantics of the preferred content. The application ontology is based on a semantic user preference model we have proposed that also allows for the explicit specification of the boolean operators to be used in the different phases of multimedia content search and filtering. The **semantic user preference model** that extends the MPEG-7/21 user preferences is described next.

- The *Domain Ontologies*, which extend the Upper Ontology and the Application Ontologies with domain knowledge. For example, consider sports ontologies that extend the abstract semantic description capabilities of the MPEG-7 MDS. A methodology has been developed for defining and integrating domain ontologies in the DS-MIRF framework and ontologies for soccer and Formula 1 have been developed in order to test the methodology proposed.

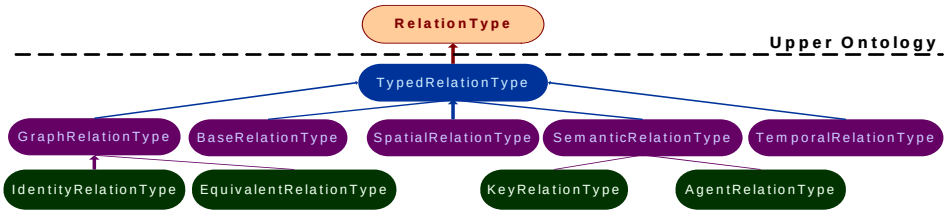


Figure 25: The OWL class hierarchy defined for the representation of typed relationships

MPEG-7 OWL Ontologies

The MPEG-7 standard sees the relationships as types. This fact leads the need of creating an application ontology for the representation of typed relationships, which contains a set of extensions for the MPEG-7 MDS that allow the full representation of typed relationships that are described in the MPEG-7 MDS text but their features are not fully

captured in the MPEG-7 MDS syntax [Tsinaraki, 2006]. The relationships are represented in the MPEG-7 MDS as instances of the "RelationType" class. An OWL class hierarchy has been defined rooted in the "TypedRelationType" (which is a subclass of the "RelationType" class of the Upper Ontology).

Additional restrictions for the general-purpose relationships expressed in the Upper Ontology and the typed relationship application ontology are usually needed (e.g. a 'Goal' event may be related to player instances as goal agents). In these cases, properties are defined that permit relating relationships to the allowed domain-specific entities only.

- A subproperty of the "Relation" property ("Relation" links semantic entities with relationships) is defined. As an example, assume that we would like to express the restriction that goals should be scored only by players. The "ScoresRelation" object property (subproperty of the "Relation" property), should be defined, having the "PlayerObject" class as domain and as range the "AgentRelationType" class [Tsinaraki et al, 2006].

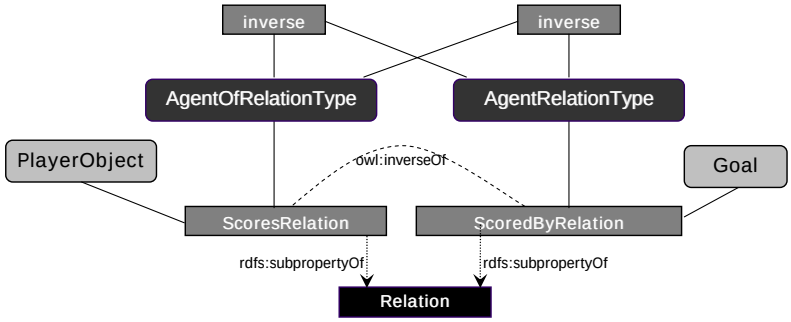


Figure 26: The "ScoresRelation" and "ScoredByRelation" object properties

- The inverse property of the one defined above is defined in the domain of the classes the individuals of which are capable of being targets of the relationship. If the relationship used in the previous step is not directed, it becomes the range of the newly-defined property. If the relationship used is directed, its inverse relationship becomes the range of the property. Since the "AgentRelationType" relationship is directed, in the example above, the "ScoredByRelation" object property (inverse of the "ScoresRelation" property) should be defined, having the "Goal" class as domain and as range the "AgentOfRelationType" class.

These features are parameters that must be taken into account when measuring the semantic relatedness of concepts inside the ontology. So Equation 26,

$$sim_{prop}(c_1, c_2) = (f_1 \times \frac{\sum_{i=1}^{n} p_{ijk}}{\sum_{i=1}^{n} p_{ij}}) + (f_2 \times \frac{\sum_{i=1}^{n} p_{invijk}}{\sum_{i=1}^{n} p_{ijk}}), \text{ becomes}$$

$$sim_{prop}(c_1, c_2) = (f_1 \times \frac{\sum_{i=1}^{n} p_{ijk}}{\sum_{i=1}^{n} p_{ij}}) + (f_2 \times \frac{\sum_{i=1}^{n} p_{invijk}}{\sum_{i=1}^{n} p_{ij}}),$$

and we consider $f_2 = f_1$ because we are more interested of the inverse properties that are subproperties of the Relation Property, since they model the relationships in MPEG-7. Also, in the RelationType classes the datatypeProperty *target* and *source* act as the intermediate objectProperties of the Classes that have ObjectProperties, inverse properties that are also subproperties of the Relation Property. So, when the OntoNL system detects an objectProperty with Range a SubClass of the RelationType Class it checks for the domain of its inverse Property.

The conclusion after this analysis is that by using the weighted shortest path calculation algorithm in the DS-MIRF ontologies we will not get the most accurate results because strongly related Classes are not connected through objectProperties. We need a more specialized way of dealing with this differentiation and this will be to detect the objectProperties that are subProperties of the Relation Property. We will consider that this objectProperty has as Range, the Domain of its inverse property and we will continue with the path calculation.

The WordNet Ontology

The WordNet Ontology that is accessed by the OntoNL Infrastructure through the JWordNet API was presented in Chapter 2: Related Research and Work pp.38.

The DS-MIRF Metadata Repository

The DS-MIRF Metadata Repository (the logical architecture of the platform is depicted in Figure 29) is a part of the DS-MIRF Framework, which is accessed by the end-users through appropriate *application interfaces*. The application interfaces may provide the end-users with multimedia content services like multimedia content retrieval, filtering and delivery.

Retrieval and Filtering Support

The retrieval and filtering support in the DS-MIRF Metadata Repository is based on semantic queries that may be specified by the end-users using appropriate query editors on top of the DS-MIRF framework. The semantic queries may have implicit or explicit boolean operators.

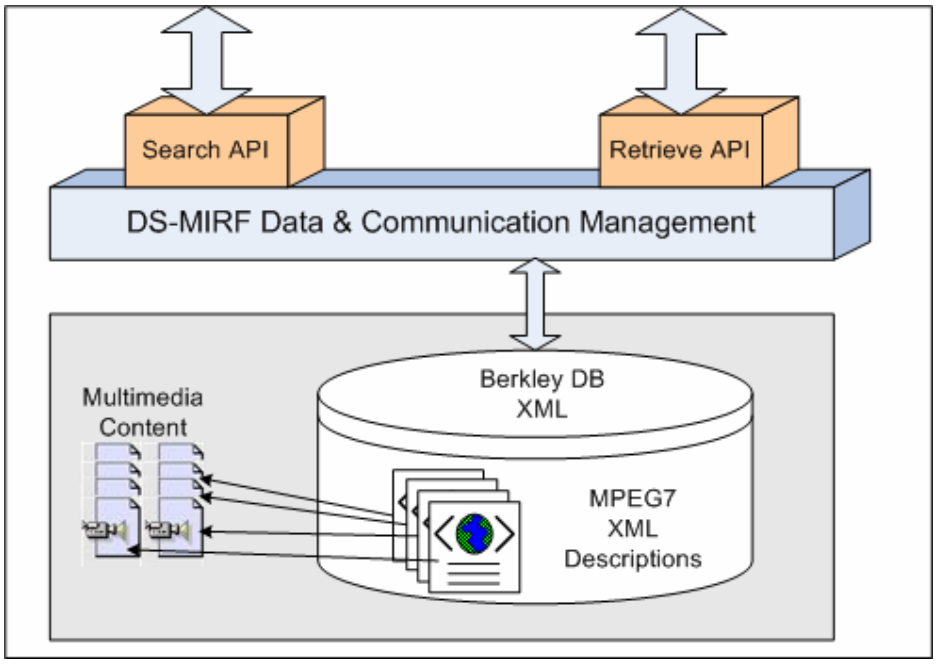


Figure 27: The DS-MIRF Metadata Repository

A semantic query with implicit boolean operators (Q) is described by the regular expression of Expression 1, while a semantic query with explicit boolean operators (QB)

is described by the regular expression of Expression 2. In both cases, pv is a preference value in the range [-100, 100] and T a semantic entity (described as shown in Expression 3).

Q = (T pv)*

Expression 1: Formal syntax of a semantic query with implicit boolean operators (Q)

QB = (((T(OR T)*) pv) | (((T(AND T)*) pv))*

Expression 2: Formal syntax of a semantic query with explicit boolean operators (QB)

T = (Tid TType) | (Tid TType) AND ((EName (EAName EAValue)* (E)*)|(RType RTarget [RSource] [RStrength])|(AName AValue)) (AND((EName (EAName EAValue)* (E)*)| (RType RTarget [RSource] [RStrength])|(AName AValue)))*

Expression 3: Formal syntax of a semantic entity (T)

In the Table 4 below we see examples of queries using the formal syntax specified in Expression 1 and Expression 2. The queries use either the general constructs provided by MPEG-7/21 (queries 1, 2) or the MPEG-7/21 constructs and domain knowledge (queries 3, 4, 5 and 6).

Query	Natural Language Description
1. (Zuninio, AgentObjectType) 100	Give me the segments where Zuninio appears (not only as a player!)
2. ((D, SemanticTimeType) AND (after, D1)) AND ((D1, SemanticTimeType) AND (Time, 11/6/2004)) 100)	Give me the segments referring to time after 11/6/2004
3. ((Zuninio, AgentObjectType) AND (exemplifies, Player)) 100	Give me the segments where the player Zuninio appears
4. ((BGoal, EventType) AND ((exemplifies, Goal) AND (agent, Barcelona)) 100)	Give me the segments where Barcelona scores
5. ((ZGoal, EventType) AND ((exemplifies, Goal) AND (agent, Zuninio) AND (patient, Kahn)) 100)	Give me the segments where the player Zuninio scores against Kahn
6. ((DLGoal, EventType) AND ((exemplifies, Goal) AND (time D) AND (place, da-Luz) AND ((D, SemanticTimeType) AND (Time, 4/7/2004)) AND ((da-Luz, SemanticPlace-	Give me the segments where a goal takes place on 4/7/2004 in the soccer stadium da Luz

Type) AND (exemplifies, SoccerStadium))) 100)	

Table 4: **Semantic Query Examples**

The OntoNL to DS-MIRF Communication Management

The OntoNL framework is a component that every repository can use to provide a natural language interface to users to access its information. The OntoNL in order to maintain its reusability and independence from application specific requirements provides the NL Query API that receives a Natural Language expression and gives back a query disambiguated based on the reference ontology the system used for disambiguation.

In each case, the repository needs to analyze the request coming from the OntoNL disambiguation process (SPARQL query) using mappings from domain ontology concepts to internal structures/data model in order to exploit its query/access mechanisms and retrieve the required information. This procedure takes place in the DS-MIRF OntoNL Manager (see figure 4).

The DS-MIRF OntoNL Manager provides the OntoNL component with the ontologies for the disambiguation and the natural language expression for disambiguation. The NL Ontology API receives the ontology-ies and continues with the processing of the structures and the semantics. The NL Query API receives the natural language expression and continues with the syntactic disambiguation.

The DS-MIRF follows a specific schema for queries. This Schema allows the specification of queries that refer to: *(a)* multimedia content that satisfies specific criteria; *(b)* semantic entities that satisfy specific criteria and can be used for the semantic descriptions of multimedia con+tent; and *(c)* constructs of domain ontologies expressed using MPEG-7 syntax. This is specified in the *From* attribute of the query form for the DS-MIRF Metadata Repository.

The query schema allows the explicit specification of boolean operators and preference values for the query elements. Three subtypes of query types have been defined for the representation of all the possible types of queries.

The *WeightedMPEG7QueryType*, which represents queries with explicit preference values (WQ) that are formally, described using the regular expression syntax of (1).

$$WQ=(WQS\ pv)^* \quad (1)$$

pv is an explicit preference value and *WQS* is a query specification with explicit preference values. The preference values are integers in the range [-100, 100], with default value 10 [Tsinaraki et. al, 2006]. The query specification represents the user's search and filtering criteria, corresponds to the WHERE part of the SELECT-FROM-WHERE languages.

The *BooleanMPEG7QueryType*, which represents queries with explicit boolean operators (BQ) that are formally described using the regular expression syntax of (2).

$$BQ=WQS[NOT]\ ((AND|OR)\ WQS\ [NOT])^* \quad (2)$$

BQS is a query specification with explicit boolean operators.

The *BooleanWeighedMPEG7QueryType*, which represents queries with explicit preference values and boolean operators (BWQ) that are formally described using the regular expression syntax of (3).

$$BWQ=WQS\ pv\ ((AND|OR)\ WQS\ pv)^* \quad (3)$$

BWQS is a query specification with explicitly specified preference values and boolean operators.

These three query types give to the DS-MIRF OntoNL Manager the initial specification of how to translate the SPARQL Query to the query language of the repository. This query has been expressed using OWL syntax (available at http://elikonas.ced.tuc.gr/Queries/MP7QL_OWL.zip).

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Show me penalties or red cards of France-->
<urn:Mpeg7Query xsi:type="urn:BooleanMpeg7QueryType"
xmlns:urn="urn:mpeg:mp7q:schema:2001"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <urn:QuerySpecification booleanOperator="AND"
xsi:type="urn:BooleanMPEG7QuerySpecificationType">
    <urn:SemanticPreferences booleanOperator="OR">
      <urn:SemanticBase booleanOperator="AND" xsi:type="urn:BooleanEventType">
        <urn:Relation booleanOperator="AND"
type="urn:mpeg:mpeg7:cs:SemanticRelationCS:2001:exemplifies"
target="soccerevents#PenaltyKick"/>
        <urn:Relation booleanOperator="AND"
target="mundial06teams.xml#FranceNationalTeamObject"/>
      </urn:SemanticBase>
      <urn:SemanticBase booleanOperator="AND" xsi:type="urn:BooleanEventType">
        <urn:Relation booleanOperator="AND"
type="urn:mpeg:mpeg7:cs:SemanticRelationCS:2001:exemplifies"
target="soccerevents#RedCard"/>
        <urn:Relation booleanOperator="AND"
target="mundial06teams.xml#FranceNationalTeamObject"/>
      </urn:SemanticBase>
    </urn:SemanticPreferences>
  </urn:QuerySpecification>
</urn:Mpeg7Query>
```

Figure 28: An example of the query "Show me penalties or red cards of France " in the query language of the repository.

System Flow

From a data flow perspective, the Natural Language Parser receives a natural language query and proceeds the complete syntactic and at a first level, semantic disambiguation, using the Stanford POS-Tagger, a syntactic analyzer, for applying grammar rules and rules for noun compound bracketing, and WordNet, as a complete word ontology. The output of the parser is a language model that needs to be fulfilled semantically, after consulting the ontologies.

The ontologies that are used for disambiguation are processed. It receives input graphs codified into a standard OWL format. This module implements the preprocessing phase, the tokenization, the abbreviation expansion and the clustering based on weight assignment. The semantic disambiguation module implements the disambiguation algorithm we proposed in Chapter 5.

After the disambiguation a list of one or more weighted query structures with information about instances of the concepts that a user may have declared.

The OWL ontologies that we used for a specific application are an upper OWL ontology fully capturing the MPEG-7 MDS [Tsinaraki et al., 2004] and a methodology for its extension with domain knowledge has been developed in the context of the DS-MIRF framework [Tsinaraki et al., 2003]. OWL/RDF metadata for audiovisual content description are produced, which are transformed, using appropriate transformation rules, to MPEG-7 compliant metadata, thus providing a basic level of MPEG-7 interoperability.

The weighted query structures are translated to SPARQL queries based on the structure of the reference disambiguation ontologies and then they are applied to the DS-MIRF OntoNL Manager, responsible for the translation of the query to the query language the repository uses. In particular the DS-MIRF Metadata Repository is an MPEG-7 XML repository contains XML Documents, which are MPEG-7 compliant audiovisual content descriptions.

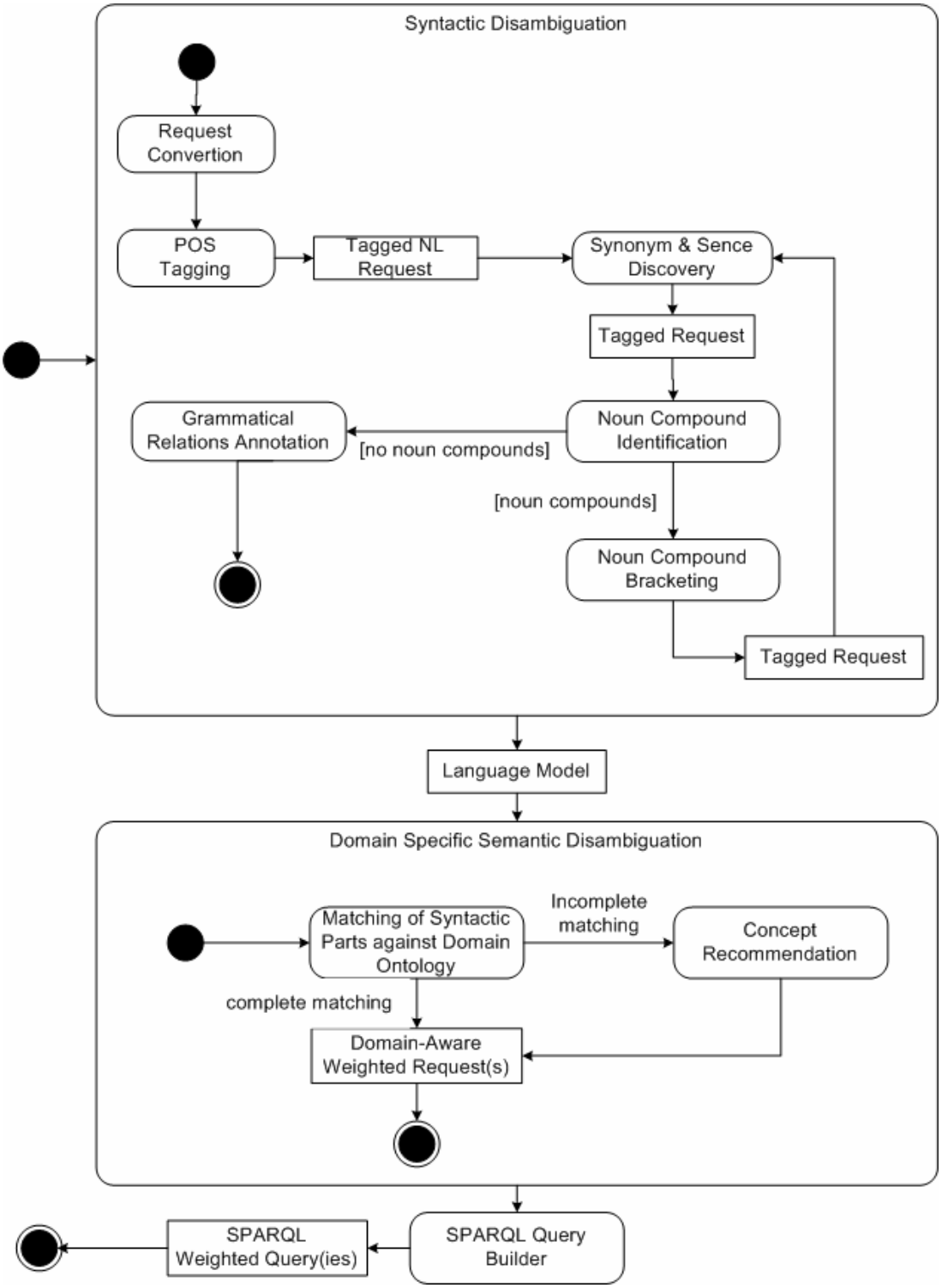


Figure 29: OntoNL System Flow

We are going to use an example query to describe the exact methodology and the data transformation after the process in every module.

Let's consider a user's request in the context of FIFA World Cup 2006: "Give me the goals scored in the soccer game between Italy and France".

First the system cuts off the subject and the verb of the request and converts the rest of the sentence to a new one with the object being the subject:

Give me the goals scored in the soccer game between Italy and France →

The goals scored in the soccer game between Italy and France

The parser handles to syntactically analyze the sentence. The part of speech tagger assigns the correct parts of speech to each word inside the sentence.

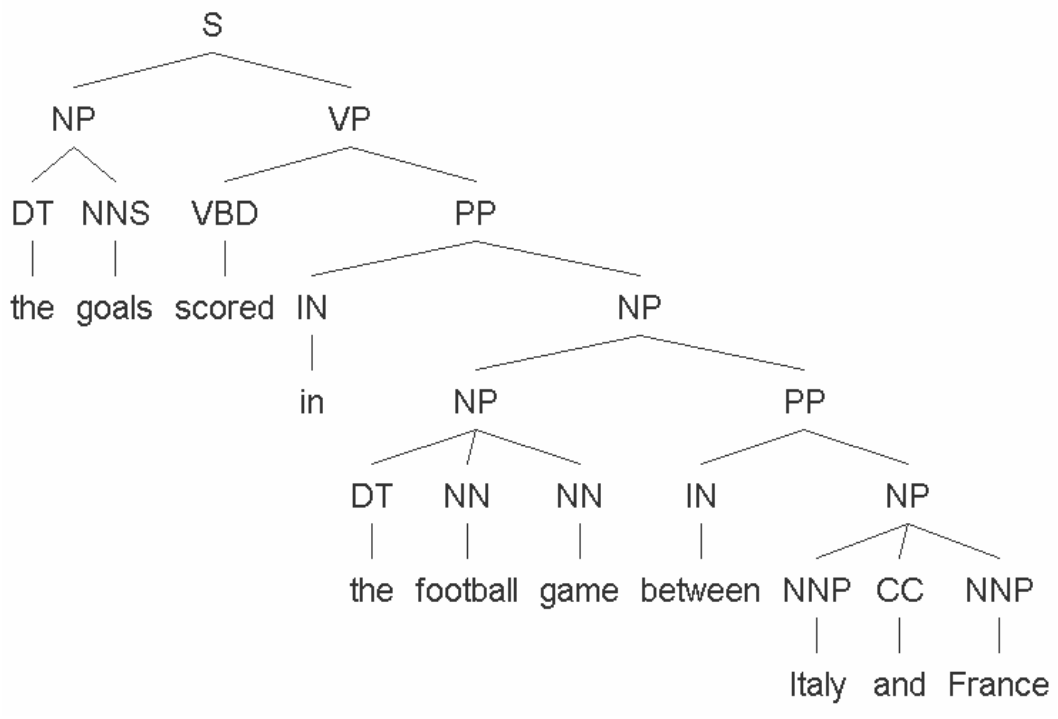


Figure 30: The part of speech assignment in the sentence *the goals scored in the football game between Italy and France*

After the pos tagging procedure the sentence has a bigram of nouns (football/NN game/NN). The noun compound analysis to achieve the maximum accuracy needs the synonym sets in the various senses of the two nouns. The parser receives all the synonyms from the WordNet

football →	football, football game,
	soccer, association football
game →	contest, competition
	activity
	diversion, recreation
	animal, animate being, beast, brute, creature, fauna
	occupation, business, job, line of work, line
	score
	meat
	scheme, strategy
	play, frolic, romp, gambol, camper

and the noun compound analysis component filters the nouns and their synonyms from the reference ontology. The ontology has a Class with name:

SoccerGame

The noun compound analysis component assigns the brackets in the bigram and the system considers the two nouns as one.

The grammatical relationship annotator generates the typed dependencies following the procedure that was described in Chapter 4, by following two phases, the **dependency extraction** and **dependency typing**.

The dependency extraction is an enhancement of the pos tagging procedure. The head of each constituent of the sentence is identified, but modified to retrieve the semantic head of the constituent rather than the syntactic head.

The goals of the soccer game between Italy and France

(NP the new phone book and tour guide)

Using the Collins rule, the head for this example is the word “guide”, and all the words in the NP depend on it. In order to find semantically relevant dependencies, we need to identify two heads, “book” and “guide”. We will then get the right dependencies (the noun “book” still has primacy as a governing verb will link to it, but this seems reasonable):

det(goals, the)

det(football game, the)

CC and(Italy, France)

prep(Italy, between)

prep(France, between)

prep(game, in)

In the second phase, we label each of the dependencies extracted with a grammatical relation which is as specific as possible. For each grammatical relation, we define one or more patterns over the phrase structure parse tree (using the tree-expression syntax defined by tregex [Levy and Andrew, 2006]. Conceptually, each pattern is matched against every tree node, and the matching pattern with the most specific grammatical relation is taken as the type of the dependency (in practice, some optimizations are used to prune the search).

The resulted phrase structure parse tree is the shown in figure 12

Figure 31: An example of a typed dependency parse for the sentence "the goals scored in the football game between Italy and France"

The synonyms and sense discovery procedure resulted in lists of senses and their synonyms for all the nouns of the sentence. For example the noun goal:

goal :: (1) goal, end → the state of affairs that a plan is intended to achieve and that (when achieved) terminates behavior intended to achieve it

synonyms: content, cognitive content, mental object → the sum or range of what has been perceived, discovered, or learned

(2) goal → a successful attempt at scoring

synonyms: score → the act of scoring in a game or sport

(3) goal → game equipment consisting of the place toward which players of a game try to advance a ball or puck in order to score points

synonyms: game equipment → equipment or apparatus used in playing a game

(4) finish, destination, goal → the place designated as the end (as of a race or journey

synonyms: end → either extremity of something that has length

The resulted structure of the syntactic analysis is an instance of the language model as it was presented in Chapter 4.

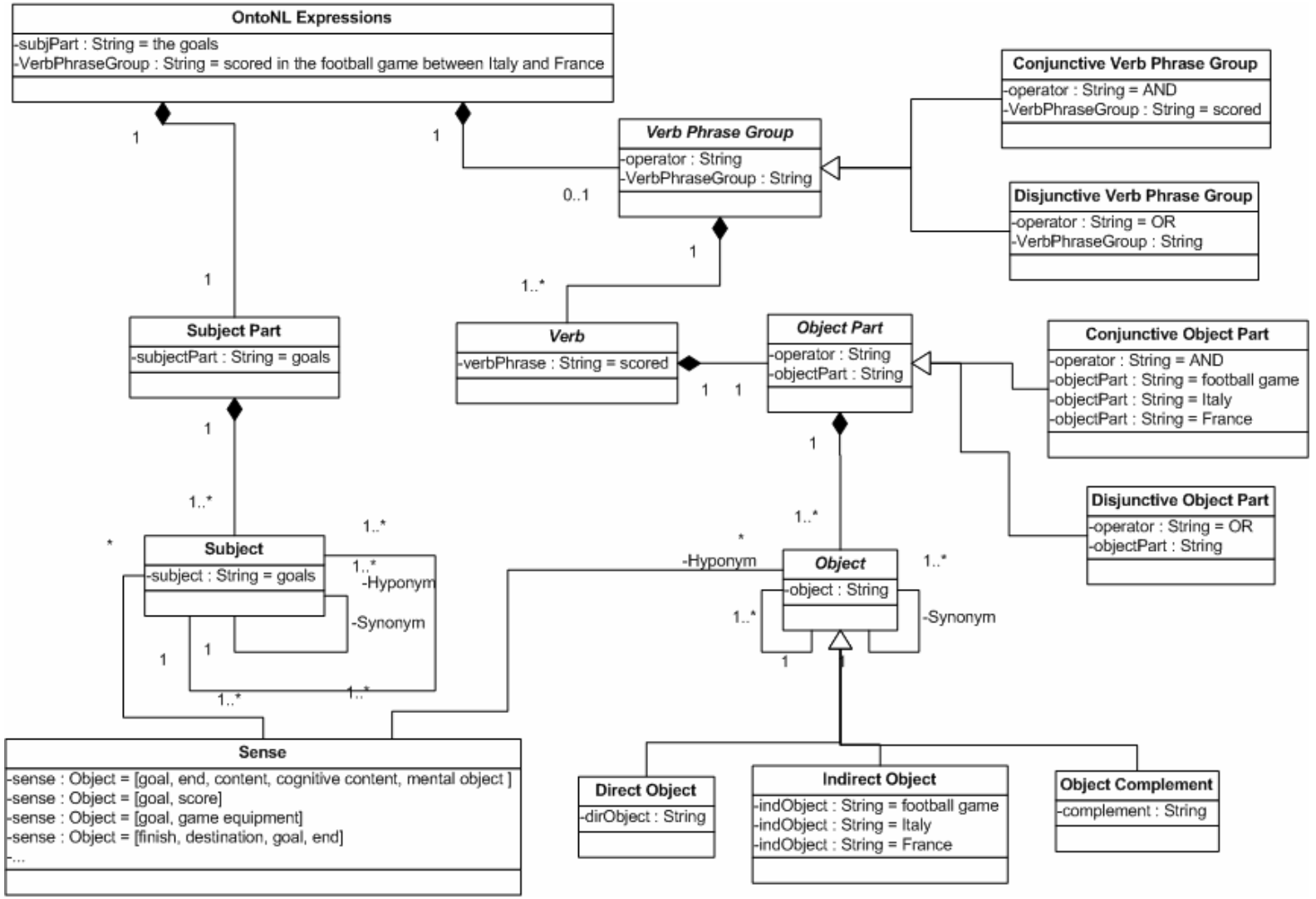


Figure 32: The language model for the sentence "the goals scored in the football game between Italy and France"

After the syntactic disambiguation the system proceeds with the semantic disambiguation. The disambiguation algorithm checks for the type of ambiguity based on the analysis of the ambiguities we made in Chapter 5. The algorithm returns an ambiguity type 2. In such case, as already discussed the OntoNL system needs to propose to the system a number of possible queries.

The systems component Ontology Processor from figure 4 has completed the tasks of word tokenization, abbreviation expansion and context clustering after the input of the reference ontologies for disambiguation. The semantic disambiguation procedure assigned a measure showing the relatedness between the concepts/classes of the ontologies. The algorithm for applying the measure was shown in detail in Chapter 5. In equation (26)

$$sim_{prop}(c_1, c_2) = (f_1 \times \frac{\sum_{i=1}^{n} p_{ijk}}{\sum_{i=1}^{n} p_{ij}}) + (f_2 \times \frac{\sum_{i=1}^{n} p_{invijk}}{\sum_{i=1}^{n} p_{ijk}}), \quad (26)$$

suggestively we applied f_2= 0.8 and f_1= 0.2, values that came up after the evaluation (see Chapter 7) because we are more interested of the inverse properties that are subproperties of the Relation Property, since they model the relationships in MPEG-7.

We also applied the three factors w_1=0.6, w_2=0.1 and w_3=0.3 of equation (31), in the particular application (see Chapter 7)

$$sim_{OntoNL} = w_1 \times sim_{COM} + w_2 \times sim_{RS} + w_3 \times sim_{CD} \quad (31)$$

If we check the relatedness value of the class **Goal** with all the other classes of the ontology and choose the three greater values of relatedness the result is:

PlayerObject: 0.17647058823529413

SoccerTeamObject: 0.11764705882352941

GoalKeeperObject: 0.23529411764705882

The number of related classes of the subject that we can apply to the repository by constructing the corresponding SPARQL queries is not standard. The best case for the

author would be to observe the behavior of the system in the beginning of use in an application and adjust the threshold of the resulted disambiguated queries and the values of the factors of the equations for similarity.

We will demonstrate in this particular case the syntax of a query were the proposed class of the class instances France and Italy is the SoccerTeam

```
PREFIX rdf: http://www.w3.org/2001/XMLSchema-instance#
PREFIX core1:file:/C:/owltool/ontologies/AV_MDS03/av_semantics#
PREFIX core2:file:/C:/owltool/ontologies/AV_MDS03/TypedRelationships#
PREFIX domain1:file:/C:/owltool/ontologies/AV_MDS03/soccer/socceragents#
PREFIX domain2:file:/C:/owltool/ontologies/AV_MDS03/soccer/soccertimes#
PREFIX domain3:file:/C:/owltool/ontologies/AV_MDS03/soccer/soccerobjects#
PREFIX domain4:file:/C:/owltool/ontologies/AV_MDS03/soccer/soccerplaces#
PREFIX domain5:file:/C:/owltool/ontologies/AV_MDS03/soccer/soccerstates#
PREFIX domain6:file:/C:/owltool/ontologies/AV_MDS03/soccer/soccerevents#
SELECT ?ScoredByRelation ?PlacedKickResultOfRelation ?AgainstOfRelation
WHERE {          ?goal domain6:CauserRelation ?CauserRelationType ;
                        Domain6:OnRelation ?ExperiencerRelationType .
                 ?CauserRelationType core2:target ?value1 .
                 ?ExperiencerRelationType core2:target ?value2
                 FILTER (?value1 = "…#France" || ?value1 = "…#Italy" &&
                 ?value2 = "…#Italy" || value2 = "…#France")};
```

In what follows, a number of screenshots will be shown that demonstrate two different implementations of the particular application.

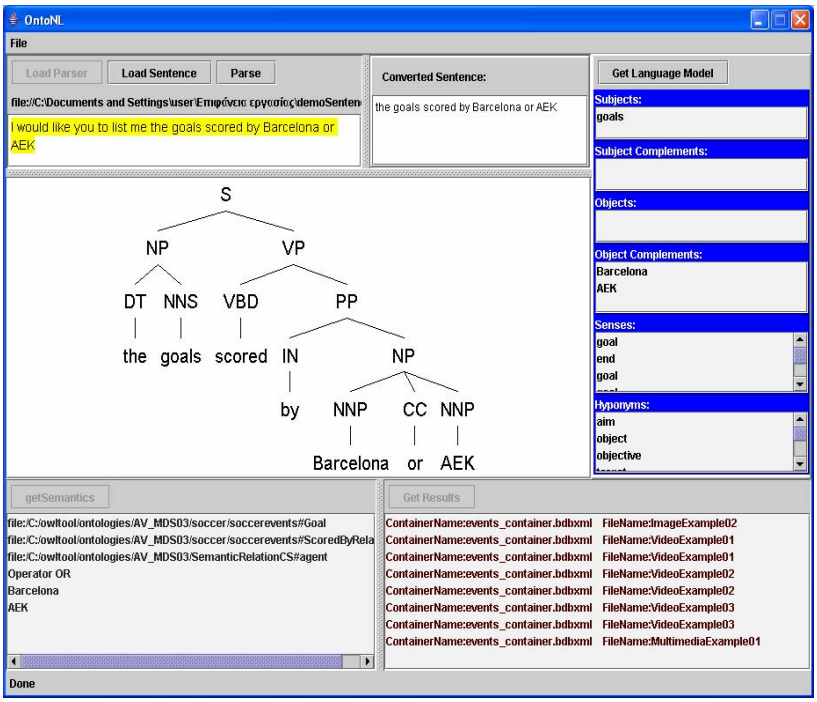


Figure 33: Screenshots of the OntoNL Framework for the domain of soccer.

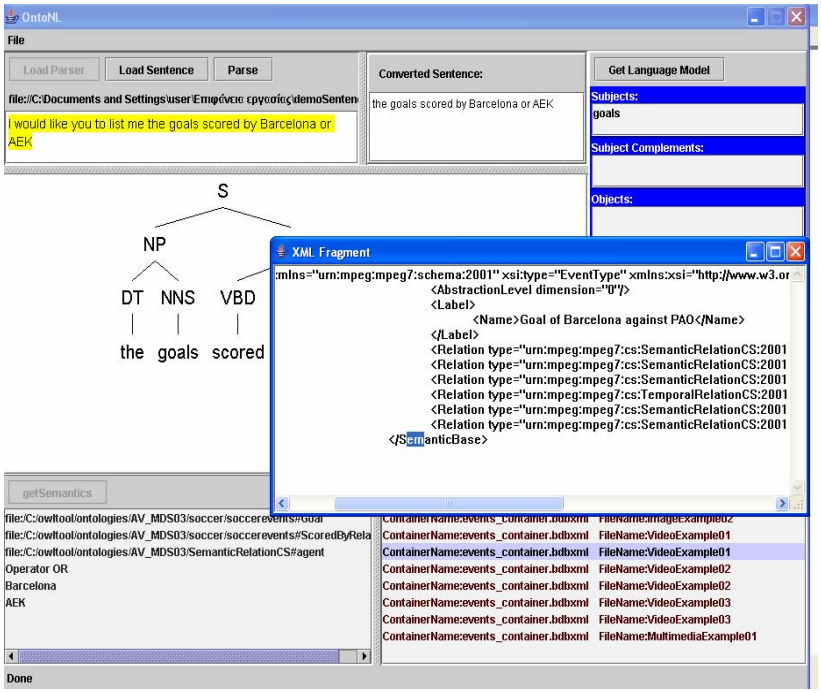


Figure 34. Screenshots with the XML fragment that contains the information asked by the user

In Figures 31 and 32 we see an extended implementation of the application were we see in details the syntactic analysis and the mapping of the natural language to the ontology structures. What we see is a graphical user interface that is comprised by a number of views that have the role to navigate us through the whole procedure followed from natural language to machine language and the retrieval of information. The options for inputting the request is by either loading a file that contains a sentence or a text in the editor or either write our request. The grammar structure of requests is of the format subject part-verb-object part were the actual usable information for the system is contained in the object part since the beginning of the requests is like “I would like you to give me”, “Show me”, “Find me” that has no usable information. So we convert the sentence in a new one where the object becomes the subject and the object complement the verb and object of the new sentence. The first part of the natural language parsing is the part-of speech tagging.

In the right column we proceed with the linguistic analysis. We first check for noun compounds consulting the information from the ontologies and then we annotate the converted sentence with the grammatical relations. By this procedure we get the subjects objects and their complements and by clicking in any of these words we retrieve its senses, synonyms and hyponyms from a word ontology, the WordNet. The lower part of the tool contains two views that concern domain information. The domain disambiguation view shows us how the significant parts of the sentence are mapped by using the disambiguation ontology. After this translation we get the results that are represented either by a pair of values of the container and the id that shows where to find documents with the requested information either by MPEG-7 XML fragments that contain the requested information either plain strings to be shown to the end user.

In figures 33 and 34 we see screenshots of an application of use of the OntoNL framework for retrieving audio visual content in the domain of the FIFA World Cup 2006. In this application we only include the request editor and the result view. This application also included the option of inserting the request using speech, but this is not one of the concerns of this work. The result view presents a list with the labels of the XML

descriptions that comprise the requested information. The user can choose to see the audiovisual content of the results, like in figure 17.

Summary

In this section we have presented the implementation of the OntoNL Framework and of a particular application in the domain of soccer. The Soccer Application of the OntoNL Framework has been demonstrated in various conferences and in the 2nd and 3rd Annual Review of the DELOS II Network of Excellence (IST 507618). We concluded with the system flow via an example. In the next section we are going to present the evaluation framework we developed for evaluating the OntoNL Framework with the experimental details and results.

Figure 35: Screenshot of an application of the OntoNL framework with use of a speech recognizer for the input of the natural language request

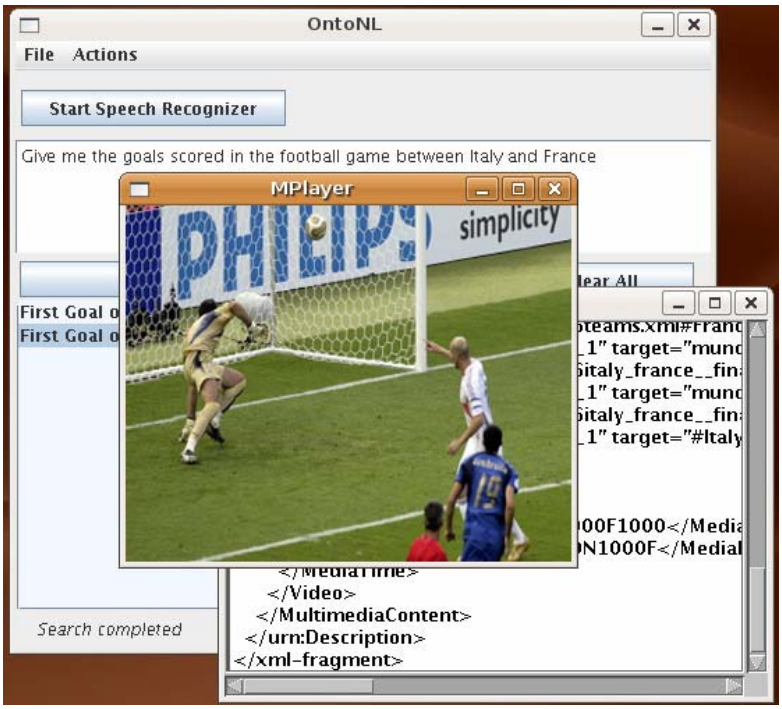


Figure 36: The multimedia object retrieved from the user selection

Chapter 7

Evaluation

One of the essential activities when providing a software system in general, is to evaluate the system based on qualitative and quantitative measures. We have considered as a starting point an existing standard, ISO 9126 [http://en.wikipedia.org/wiki/ISO_9126], which is concerned primarily with the definition of quality characteristics to be used in the evaluation of software products. ISO 9126 sets out six quality characteristics, which are intended to be exhaustive. From this it follows that each quality characteristics is very broad. We indicate two illustrative examples:

4.1 Functionality

> *A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.*

Note:

> *This set of attributes characterises what the software does to fulfil needs, whereas the other sets mainly characterise when and how it does so.*

For the stated and implied needs in this characteristic, the note to the definition of quality applies (see 3.6)

were

3.6 Note: In a contractual environment, needs are specified, whereas in other environments, implied needs should be identified and defined (ISO 8402:1986, note 1)

A second quality characteristic that will be important in what follows is usability:

4.3 Usability

> *A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.*

Notes:

1. *"Users" may be interpreted as most directly meaning the users of interactive software. Users may include operators, and users and indirect users who are under the influence of or dependent on the use of the software. Usability must address all of the different user environments that the software may affect, which may include preparation for usage and evaluation of results.*
2. *Usability defined in this International Standard as a specific set of attributes of a software product differs from the definition from an ergonomic point of view, where other characteristics such as efficiency and effectiveness are also seen as constituents of usability.*

Taking into account information from the ISO 9126 Standard we can summarize and broadly distinguish three kinds of evaluation, appropriate to three different goals.

1. Adequacy Evaluation

This is determination of the fitness of a system for a purpose---will it do what is required, how well, at what cost, etc. Typically for a prospective user, it may be comparative or not, and may require considerable work to identify a user's needs.

2. Diagnostic Evaluation

This is production of a system performance profile with respect to some taxonimization of the space of possible inputs. It is typically used by system

developers, but sometimes offered to end-users as well. It usually requires the construction of a large and hopefully representative *test suite*.

3. Performance Evaluation

This is measurement of system performance in one or more specific areas. It is typically used to compare like with like, whether two alternative implementations of a technology, or successive generations of the same implementation. It is typically created for system developers and/or R&D programme managers.

When systems have a number of identifiable components associated with stages in the processing they perform, it is important to be clear as to whether we approach the system as a whole, or try to evaluate each component independently. When considering individual components, a further distinction between *intrinsic* and *extrinsic* evaluation must be respected---do we look at how a particular component works in its own terms (intrinsic) or how it contributes to the overall performance of the system (extrinsic).

Measures Description

1. Adequacy Evaluation

The Adequacy Evaluation can be divided in two further evaluations: the **Expert-based** and the **User-based evaluation.** The **Expert-based evaluation** is performed by HCI experts who evaluate the usability of the interfaces according to a defined set of heuristics. These heuristics address mainly the Natural Language Interfaces usability. The user interface (UI) can be critical to the success or failure of a computer system. The development of UIs requires an iterative design and evaluation process involving users at every stage.

Specifically, the most significant parts to be considered are:

- developing a UI in a flexible, iterative manner, working in close collaboration with the users;

- identifying who will use the system, the tasks they want to carry out and the environment in which they will be working;
- creating a conceptual design;
- choosing the most appropriate interaction style;
- choosing appropriate interaction devices;
- using text, colour, images, moving images and sound effectively;
- evaluating the UI,

Optimized User Interface Design requires a systematic approach to the design process. But, to ensure optimum performance, Usability Testing is required. This is what we call **User-based evaluation** and is performed by the end users of the system. This empirical testing permits expert and naïve users to provide data about what does work as anticipated and what does not work. Only after the resulting repairs are made can a system be deemed to have a user optimized interface.

HCI-expert users can define the form of the information provided to users (expert or naïve users), who is most important and reasonable to be shown and the way to be presented regardless of the graphical UI design.

As we presented in Chapter 6, an application of the OntoNL system has been implemented that is as a question answering system for the domain of soccer, were naïve users are the actual users of the system using a pc or a web-based user interface for entering a request. We want to measure the satisfaction of users for the effectiveness of applying their requests to the system using the web after presenting them the results.

2. Diagnostic Evaluation

The Diagnostic Evaluation is about testing the range of possible sentences that the OntoNL system can parse and disambiguate linguistically. It is conducted by system developers and it refers to the successfully parsing of natural language expressions and to different categories of grammatical relations combinations that need to be disambiguated.

Below we present the different categories of request types that the system disambiguates and obtains results, through a class diagram and a description of the diagram.

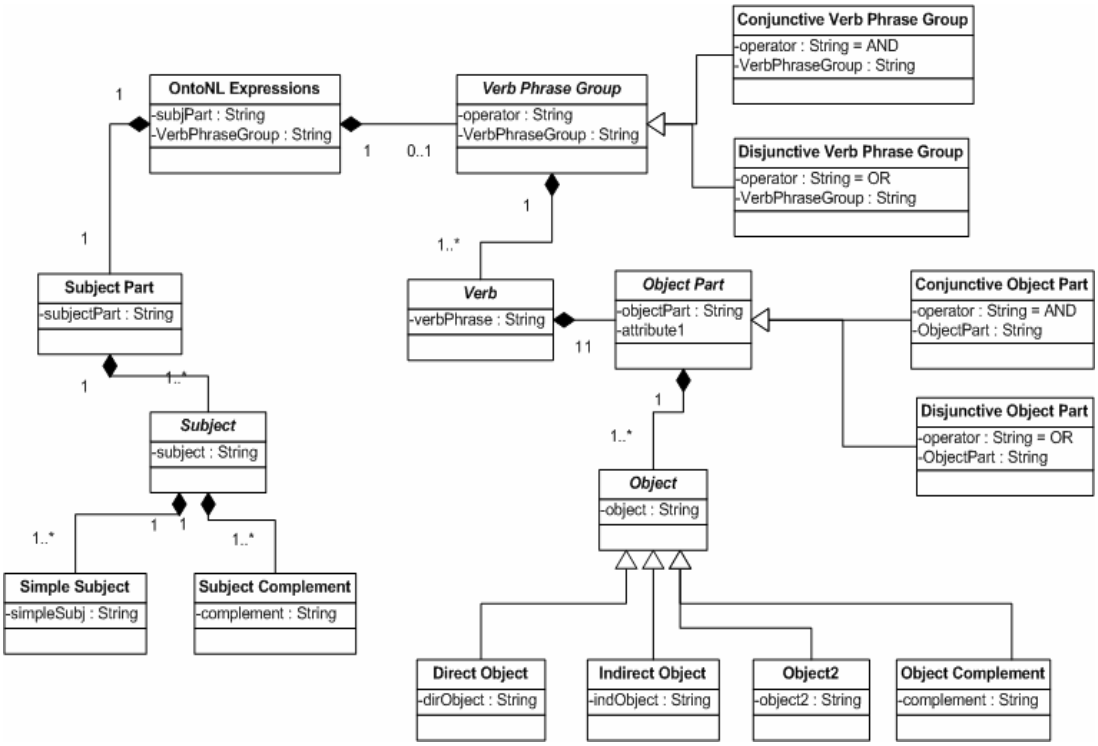


Figure 37: The language model that describes the different categories of Natural Language expressions that the OntoNL can parse

OntoNL Natural Language Expressions						
SubjectPart						
Subject	1	VerbPhraseGroup		ObjectPart		
	Boolean					
	AND	Verb	1	Object	1	Direct
			Boolean AND			Indirect
	Subject					Object2
	Complement		Boolean OR			Object
					Boolean AND	
					Boolean OR	

Figure 38: Syntax of the natural language expressions

3. Performance Evaluation

The performance evaluation can be distinguished to the quantitative and the qualitative evaluation. We are interested in the qualitative performance.

The qualitative performance evaluation concerns the performance of the relatedness measure and the query formulation. How can we reason about computational measures of semantic relatedness? Given a single measure, can we tell whether it is a good or a poor one? Given two measures, can we tell whether one is better than the other?

Evaluation of semantic relatedness measures remains an open question [Agirre and Rigau, 1997, Resnik, 1995, Hirst and St-Onge, 1998]. In our survey of literature on the topic, we have come across three prevalent approaches: mathematical analysis, comparison with human judgement, and application-specific evaluation.

The first approach (see, e.g., [Wei, 1993, Lin, 1998]) consists in a (chiefly) theoretical examination of mathematical properties of a measure, such as whether it is actually a metric, whether it has singularities, whether its parameter-projections are smooth functions, etc. Such analyses, in our opinion, may certainly aid the comparison of several measures but perhaps not so much their individual assessment.

The second approach, comparison with human judgments of relatedness, does not appear to suffer from the same limitations; in fact, it arguably yields the most generic assessment of the 'goodness' of a measure; however, its major drawback lies in the difficulty of obtaining such judgements (i.e., designing a psycholinguistic experiment, validating its results, etc.). In his [1995] paper, Resnik presented a comparison of the ratings produced by his measure simR (and a couple of others) with those produced by human subjects on a set of 30 word pairs from an experiment by Miller and Charles [1991]. The fact that others [Jiang and Conrath, 1997, Lin, 1998] followed his lead and employed the same modestly sized dataset in their work appears to be a testament to the seriousness of the problem.

Because of these deficiencies, we, generally, have to take sides with the remaining group of researchers who have chosen to evaluate their measures in the framework of a particular NLP application.

However, since the trend has been established and since we have also found a use for the results in our application-specific evaluation, we decided to have the measures implemented as part of the application-specific evaluation.

Evaluation Results

1. Adequacy Evaluation

The adequacy evaluation addresses the evaluation of the usability of the interfaces according to a defined set of heuristics. The graphical user interface that was evaluated is the application of the OntoNL Framework in the domain of soccer (OntoNL2DS-MIRF) as it was described in the implementation chapter and is presented in figure 39 and not the OntoNL Component. This evaluation has as a starting point the ten heuristics of Nielsen [Nielsen, 1994]. The evaluation comments are based on comments of an evaluation team of three HCI experts, Prof. Tiziana Catarci, Dr. Yael Dubinsky and Dr. Stephen Kimani, from the Department of Computer and Systems Science (Dipartimento di Informatica e Sistemistica) of the University of Rome "La Sapienza" and the reflection on these comments as was expressed by people who are involved in the development of this system.

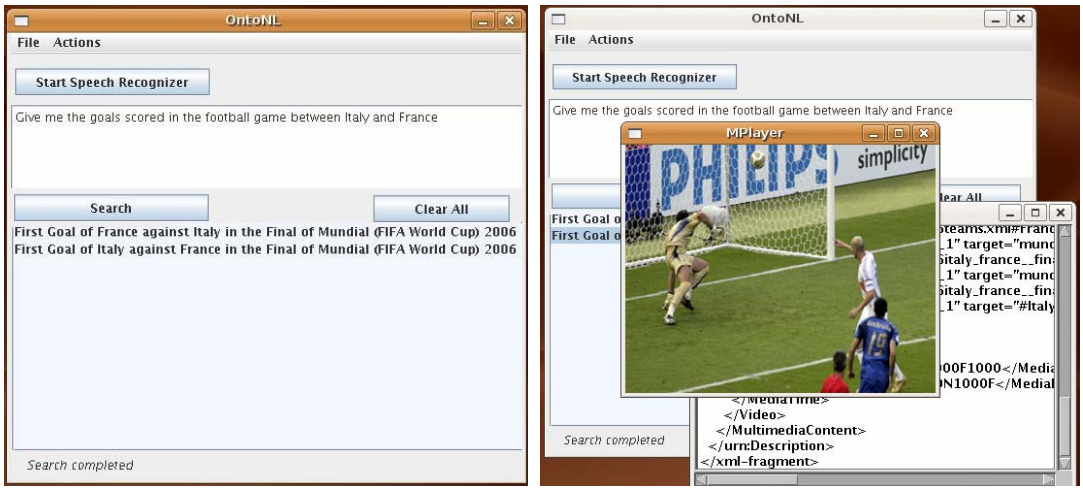


Figure 39: Screenshot of the graphical user interface of the OntoNL Framework application for the domain of soccer.

The useful comments that derive from the Adequacy Evaluation were taken into account for further refinements of the application.

1. Visibility of System Status

Description: The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

Comments:

• In the current system, users should only press one button in order to continue with the parsing of the question till reaching the final answer for their question. The users are not interested with this parsing process (parsing tree, possible ambiguities, etc.) and this information is being kept invisible to the user.

• The system is expected to provide progress information towards the final answer/s. The case of progress information that relate to a series of questions/answers should be considered.

2. Match between the System and the Real World

Description: The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

Comments:

• Globally, the words that are used are familiar to users. Users expect answers to be displayed in their (natural) language vs. XML data presentation, so maybe the XML fragment should not be displayed.

• Users can start each question with phrases like “I would like to know…” and a more straight forward approach i.e., how, what, when questions that is a better approach when the system is being used extensively.

3. User control and freedom

Description: Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

Comments:

• Options of delete, move up/down should be added assuming users are able to manipulate their already asked questions set.

• Options of undo and redo should be added assuming users are able to manipulate the list of results (answers).

4. Consistency and Standards

Description: Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

No special comment.

5. Error prevention

Description: Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

Comments:

• Errors should be prevented, e.g., when high probability that the question cannot be parsed correctly.

6. Recognition rather than recall

Description: Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue

to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

Comments:

• Users are able to manipulate their already asked questions set. This way they will not have to remember them and it will be available for reuse.

7. Flexibility and efficiency of use

Description: Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

Comments:

• Experienced users are not provided with shortcuts.

8. Aesthetic and minimalist design

Description: Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

Comments:

• All areas in the current interface are useful and will be used by far by users.

9. Help users recognize, diagnose and recover from errors

Description: Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

Comments:

• More error messages and confirmation dialogues should be added.

10. Help and documentation

Description: Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

Comments:

• On-line help should be provided for users in order to know which questions are permitted, and how to manipulate series of questions/answers.

2. Diagnostic Evaluation

We are interested in the successful parsing of sentences with the syntax shown in figure 34. Direct comparison between our system and other dependency parsers like Minipar [Lin, 1998] and the Link Parser [Sleator and Temperlay, 1993] is complicated by differences between the annotation schemes targeted by each system, presumably reflecting variations in theoretical and practical motivations. The systems do not always agree about which words should be counted as the dependents of a particular sentence. Even when the systems agree about whether two words are in a dependency relation, they may diverge about the type of the dependency. Each system assigns dependency types from a different set of grammatical relations and it is not straightforward to establish mappings between these sets. Also, the names used for relations vary considerably, and the distinctions between different relations may vary as well. Such differences make it difficult to directly compare the quality of the three systems. The most salient difference between the schemes is the level of granularity. Carroll's scheme contains 23 grammatical relations, MiniPar 59, Link 106 and ours 22. Based on De Marneffe's evaluation attempts [De Marneffe et. al, 2006] for the Stanford Parser against Link and Minipar parsers, Lin [Lin, 1998] proposes two ways to evaluate the correctness of a dependency parse against a gold standard. In the first method,

one simply examines whether each output dependency also occurs in the gold standard, while ignoring the grammatical type of the dependency; this method is therefore sensitive only to the structure of the dependency tree. The second method also considers whether the type of each output dependency matches the gold standard. But because the correctness of a dependency parser must be evaluated according to the annotation scheme it targets, and because each parser targets a different scheme, quantitative comparison is difficult.

To provide a qualitative comparison, we tagged, with the three taggers, fifteen sentences chosen from the Brown Corpus. The sentences we examined (table 4) agree with the language model we have developed for the OntoNL Framework. In what follows, we present in figures 37, 38 and 39 the dependency graphs that are produced after the parsing of the sentence "Bills on ports and immigration were submitted by Senator Brownback". We chose this sentence as an illustrative example because it is short but shows typical structures like prepositional phrases, coordination and noun compounding. The dependency graph is a tree, a singly rooted directed acyclic graph with no re-entrances. The graph representing Minipar output collapses directed paths through preposition nodes. It also adds antecedent links to 'clone' nodes between brackets. The graph for the Link Parser presents the same collapsing of directed paths through preposition nodes.

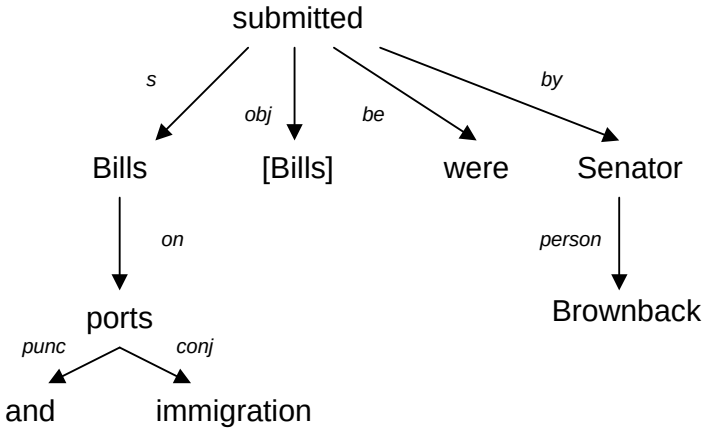


Figure 40: Minipar's dependency parse for the sentence "Bills on ports and immigration were submitted by Senator Brownback"

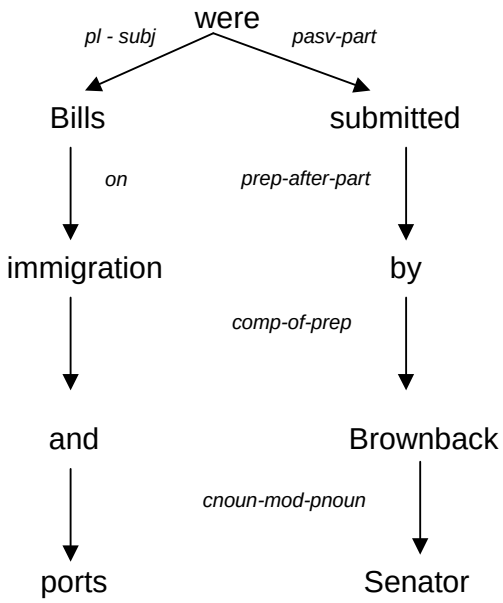


Figure 41: Link Parser's dependency parse for the sentence "Bills on ports and immigration were submitted by Senator Brownback"

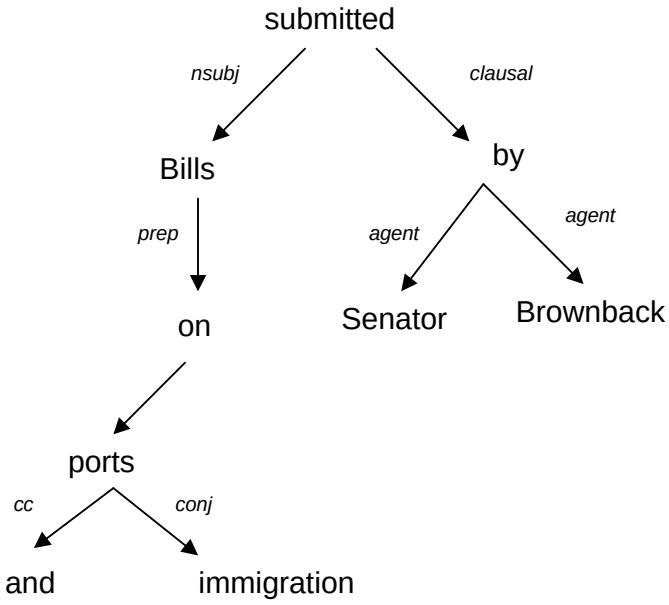


Figure 42: OntoNL Parser's dependency parse for the sentence "Bills on ports and immigration were submitted by Senator Brownback"

Generally, the Stanford (the tagger, we also use in our system) and the Link tagger lead to more accurate structures than Minipar [De Marneffe et. al, 2006]. The Stanford tagger was

trained on the Penn Wall Street Journal Treebank and does a poor job at parsing questions, though. This is easily explained by the fact that the parser is trained on the Wall Street Journal section of the Penn Treebank in which not many questions occur. Minipar is confused by punctuation (already mentioned in [Lin, 1998]) and is also confused by conjunctions. Our parser behaves very well in conjunctions because of the strict language model it follows. An advantage of the Minipar is its capacity to identify collocations. The Link parser also has trouble with conjuction: it did not parse correctly sentences 6 and 15. We evaluated our system on this sample of 15 sentences. We obtained a dependency accuracy of about 80%. However it can be only considered as a rough estimate because of the quite small sample size and the complexity of the sentence structure. Our objective was to evaluate the OntoNL parsing mechanism in comparison with other well known parsers in order to conclude to advantages and future refinements.

1	She lived and was given a name. ID: cm05 | genre: scifi
2	He had better write a postcard to Walter. ID: cn19 | genre: adventure
3	People came in and out all evening to see the baby. ID: cp02 | genre: romance
4	Spencer said nothing. ID: cp07 | genre: romance
5	They make us conformists look good. ID: cp15 | genre: romance
6	A cookie with caramel filling and chocolate frosting won the cooking competition. ID: ca30 | genre: reportage
7	Everywhere I went in Formosa I asked the same question ID: cb23 | genre: editorial
8	The letters of the common soldiers are rich in humor. ID: cf18 | genre: popularlore
9	This time he was making no mistake ID: cg32 | genre: belleslettres
10	It usually turned out well for him ID: cg60 | genre: belleslettres
11	The author of the anonymous notes seemed to be all-knowing. ID: cn11 | genre: adventure
12	Below he could see the bright torches lighting the riverbank ID: ck21 | genre: generalfiction
13	Beckworth handed the pass to the colonel. ID: ck21 | genre: generalfiction

14	Must Berlin remain divided? ID: cb02 | genre: editorial
15	Old, tired, trembling the woman came to the cannery. ID: cb08 | genre: editorial

Table 5: 15 sentences from the Brown Corpus, to compare outputs of Minipar, the Link Parser and the OntoNL parser.

3. Performance Evaluation

The Performance Evaluation is comprised of the two parts of evaluation; the quantitative and the qualitative evaluation. In this thesis we are going to deal with the qualitative evaluation of specific processes of the OntoNL Framework. The qualitative evaluation concerns measuring the effectiveness of the **noun compound bracketing mechanism**, the **semantic relatedness measure** and **an application-based evaluation of measures of relatedness**.

NOUN COMPOUND BRACKETING

In this section we describe the methodology of training the noun compound bracketing algorithm we described in Chapter 4 and evaluating its accuracy by using two large OWL domain ontologies freely available in the web, the Soccer Ontology (http://www.music.tuc.gr/ontologies/mpeg7/mds/socccer/) and the Biopax-Level 2 Ontology (http://www.biopax.org/).

Method

In all the experimental work we will only consider English compound nouns. Nonetheless, compounds appear in many other languages and there seems no reason why the same techniques we used would work less well in these.

We also assume that the possible compound has been recognised from the surrounding text based on the linguistic analysis as it was described in Chapter 4, so that the system is presented with a sequence of nouns known to be a compound.

Given an identified compound, it is simplest to define the parsing task as one of bracketing. That is, the system must select the most likely binary bracketing of the noun sequence, assuming that it is a compound noun.

According to most views of compounding, the composition of two or more nouns yields an element with essentially the same syntactic behaviour as the original nouns. An n-word compound noun acts exactly like a single noun, as do three word compounds and so forth.

To define the primary goal of the work in the OntoNL noun compound bracketing mechanism we conclude to the next statement:

Problem Statement: Given a three word English compound noun predict whether the most likely syntactic analysis is left-branching or right-branching.

Extracting a Test Set

Two test sets of syntactically unambiguous noun compounds was extracted from a 67 pages document describing molecular binding interactions, protein post-translational modifications, basic experimental descriptions, and hierarchical pathways and a 115 official document from FIFA describing the rules of football in the following way. Because the corpus is not tagged or parsed, a somewhat conservative strategy of looking for unambiguous sequences of nouns was used. To distinguish nouns from other words we used once again the Stanford Log-Linear Tagger to generate the set of words that can only be used as nouns. Let's call this set from now on *N*. All consecutive sequences of these words were extracted, and the three word sequences used to form the test set. The result was 98 test trigrams.

These triples were manually analysed using as context the entire article in which they appeared. In some cases, the sequence was not a noun compound (nouns can appear adjacent to one another across various constituent boundaries) and was marked as an error. Other compounds exhibited what Hindle and Rooth (1993) have termed SEMANTIC INDETERMINACY where the two possible bracketings cannot be

distinguished in the context. The remaining compounds were assigned either a left-branching or right-branching analysis. The number of each kind is shown in Table 6.

Type	Number	Proportion
Error	7	7%
Indeterminate	11	11%
Left-branching	52	53%
Right-branching	28	29%

Table 6: Test Set distribution

Conceptual Association

As we have described in Chapter 4, we use the term *Conceptual Association* in this study to refer to association values computed between groups of words. We have used groups consisting of all categories from the Roget's II: The New Thesaurus (http://www.bartleby.com/62/). By assuming that all words within a group behave similarly, the parameter space can be built in terms of the groups rather than in terms of the words.

Given two thesaurus categories t_1 and t_2, there is a parameter which represents the degree of acceptability of the structure [n_1 n_2] where n_1 is a noun appearing in t_1 and n_2 appears in t_2. By the assumption that words within a group behave similarly, this is constant given the two categories.

Following Lauer and Dras (1944) we can formally write this parameter as $Pr(t_1 \rightarrow t_2)$ where the event $t_1 \rightarrow t_2$ denotes the modification of a noun in t_2 by a noun in t_1.

Training

To ensure that the test set is disjoint from the training data, all occurrences of the test noun compounds have been removed from the training corpus.

We are going to explore two types of training scheme. The first employs a pattern that follows Pustejovsky (1993) in counting the occurrences of subcomponents. A training instance is any sequence of four words $w_1w_2w_3w_4$ where $w_1, w_4 \notin N$ (N is a set of words that can be used only as nouns) and $w_2, w_3 \in N$. Let $count_p(w_1, w_2)$ be the number of times a sequence $w_1w_2w_3w_4$ occurs in the training corpus with $w_1, w_4 \notin N$.

The second type uses a window to collect training instances by observing how often a pair of nouns co-occur within some fixed number of words. In this work, a variety of window sizes are used.

In OntoNL we used a window to collect training instances by observing how often a pair of nouns co-occurs within some fixed number of words. For window size $n \geq 2$, let $count_n(w_1, w_2)$ be the number of times a sequence $n_1w_1 \ldots w_in_2$ occurs in the training corpus where $i \leq n-2$. The estimates are:

$$\mathrm{P}(t_1 \rightarrow t_2) = \frac{1}{\sum_{w_1 \in N, w_2 \in t_2} \frac{count_n(w_1, w_2)}{amb(w_1, w_2)}} \sum_{w_1 \in t_1, w_2 \in t_2} \frac{count_n(w_1, w_2)}{amb(w_1)amb(w_2)}$$

where *amb*(*w*) counts the number of categories w appears and N is a set of words that can only be used as nouns. The *amb*(*w*) has the effect of dividing the evidence from a training instance across all possible categories for the words. The first parameter of the multiplication is used to ensure that the parameters for a head noun sum to unity. After the calculation of the estimates, we continue by trying to make a right choice of all possible analyses for three word compounds, which are the counting of a right or a left branching analysis. So, for the adjacency model and a given compound of w_1, w_2, w_3 the estimation of the ratio is done by applying the equation

$$R_{adj} = \frac{\sum_{t_i \in cats(w_i)} P(t_1 \rightarrow t_2)}{\sum_{t_i \in cats(w_i)} P\left(t_2 \rightarrow t_3\right)}$$

for the dependency model and a given compound of w_1, w_2, w_3 the estimation of the ratio is done by applying the equation

$$R_{dep} = \frac{\sum_{t_i \in cats(w_i)} P(t_1 \rightarrow t_2) P(t_2 \rightarrow t_3)}{\sum_{t_i \in cats(w_i)} P(t_1 \rightarrow t_3) P(t_2 \rightarrow t_3)}$$

where t_1, t_2 and t_3 are conceptual categories in a taxonomy or thesaurus, and the nouns $w_1,\ldots,w_n$ are members of these categories. If the ratio is >1 then we conclude to a left-branching analysis. If it is <1 then a right branching analysis is chosen. If it is =1, the OntoNL analyzer, based on Lauer (Lauer, 1995) guesses left-branching, a rare case for conceptual association based on experimental results.

For a correct result we must sum over all possible categories for the words in the compound. In any case, the estimation of probabilities over concepts reduces the number of model parameters.

Results

In what follows, all evidence used to estimate the parameters of the model is collected in one pass over the corpus and stored in a fast access data structure. Evidence is gathered across the entire vocabulary, not just for those words necessary for analysing a particular test set. Once trained in this way, the program can quickly analyse any compound, restricted only by the lexicon and thesaurus. This demonstrates that the parsing strategy can be directly employed using currently available hardware in broad coverage natural language processing systems.

Six different training schemes have been used to estimate the parameters and each set of estimates used to analyse the test set under both the adjacency and the dependency model. The schemes used are the pattern that follows Pustejovsky (1993) in counting the occurrences of subcomponents and windowed training schemes with window widths of 2, 3, 4, 5 and 10 words.

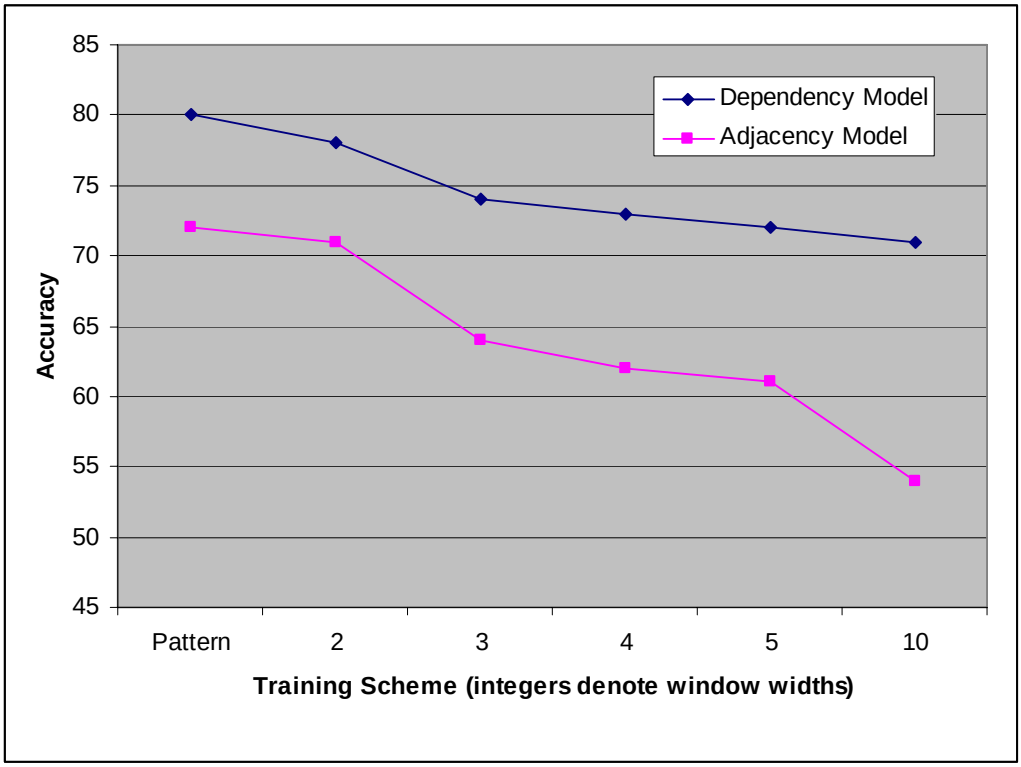


Figure 43: Accuracy of analysis of the test set under the dependency and the adjacency model for the pattern training scheme that follows Pustejovsky (1993) in counting the occurrences of subcomponents and for the windowed training schemes with window widths of 2, 3, 4, 5 and 10 words

The accuracy on the test set for all these experiments is shown in Figure 43. As can been seen, the OntoNL dependency model is more accurate than the OntoNL adjacency model. The proportion of cases in which the procedure was forced to guess, either because no data supported either analysis, is quite low. For the pattern and two-word window training schemes, the guess rate is less that 6% for both models. In the three-word window training scheme, the guess rate is less that 2%. For all larger windows, neither model is ever forced to guess.

In no case do any of the windowed training schemes outperform the pattern scheme. It seems that additional instances admitted by the windowed schemes are too noisy to make an improvement.

Lexical Association

To determine the difference made by conceptual association, the pattern training scheme has been retrained using lexical counts for both the dependency and adjacency model, but only for the words in the test set. Accuracy and guess rates are shown in figure 4. Conceptual association outperforms lexical association, presumably because of its ability to generalize (see Figure 44).

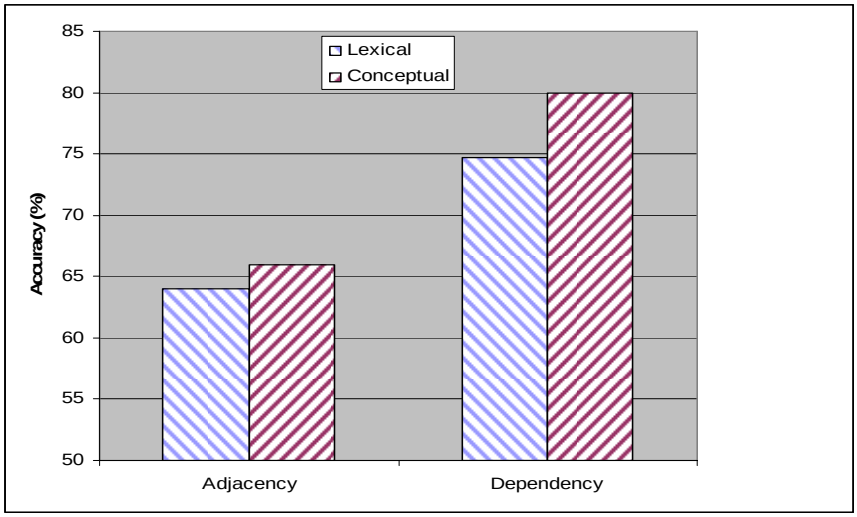


Figure 44: Accuracy of analysis of the test set under the dependency and the adjacency model for the pattern training scheme using lexical association and conceptual association

Using a Tagger

One problem with the training methods we presented previously is the restriction of training data to nouns in *N*. Many nouns, especially common ones, have verbal or adjectival usages that preclude them from being in *N*. Yet when they occur as nouns, they still provide useful training information that the current system ignores. To test whether using tagged data would make a difference, the freely available Stanford Log-Linear POS Tagger (http://nlp.stanford.edu/software/tagger.shtml) was applied to the corpus. Since no manually tagged training data is available for our corpus, the tagger's default rules were used.

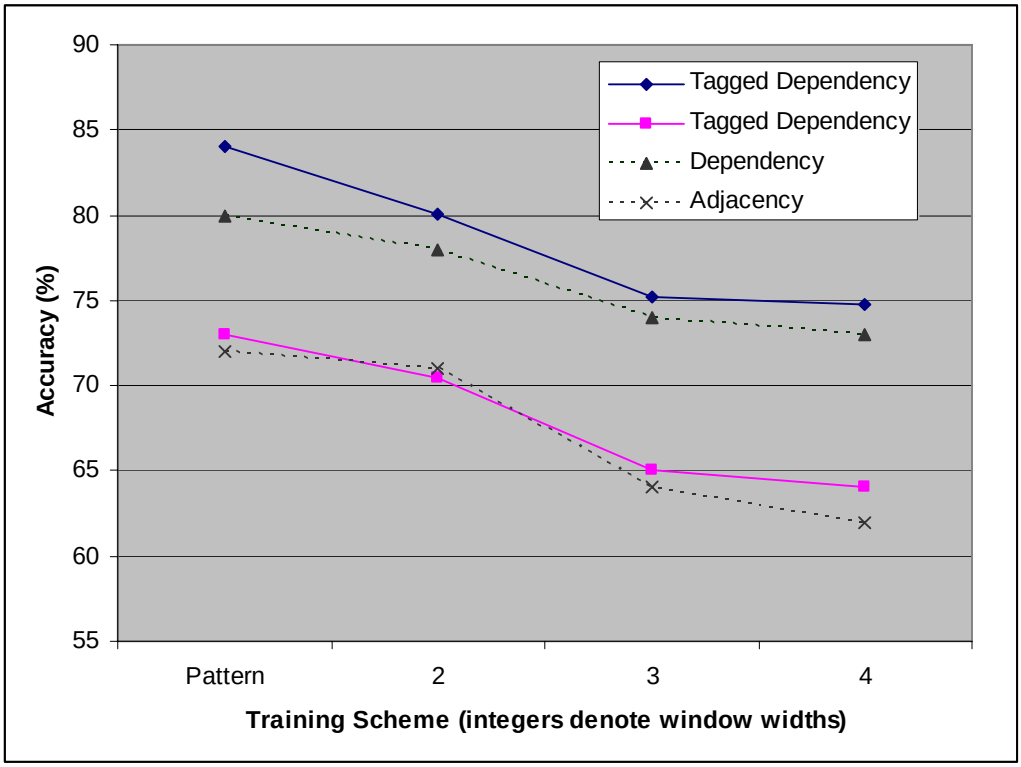


Figure 45: Accuracy of analyzing the test set using a tagged corpus under the dependency and the adjacency model for the pattern training scheme that follows Pustejovsky (1993) in counting the occurrences of subcomponents and for the windowed training schemes with window widths of 2, 3, and 4 words and comparison with the accuracy presented in figure 43.

Four training schemes have been used and the tuned analysis procedures applied to the test set. Figure 45 shows the resulting accuracy, with accuracy values from figure 43 displayed with dotted lines. If anything, admitting additional training data based on the tagger introduces more noise, reducing the accuracy. However, for the pattern training scheme an improvement was made to the dependency model, producing the highest overall accuracy of 84%.

Using Domain Ontologies

What we propose in this method is to use as corpus the nouns used for naming the concepts of the domain ontolog, plus their synonyms and the descriptions of the concepts inside the ontology. One problem with this approach is that in the absence of descriptions we only have as training corpus the names of the concepts of the ontology which is a very limited corpus with either excellent or very bad results. On the other hand when the

domain ontology used for the semantic disambiguation is also used for the noun compound bracketing mechanism there is no need to find relative to the domain corpuses each time we want to use the OntoNL.

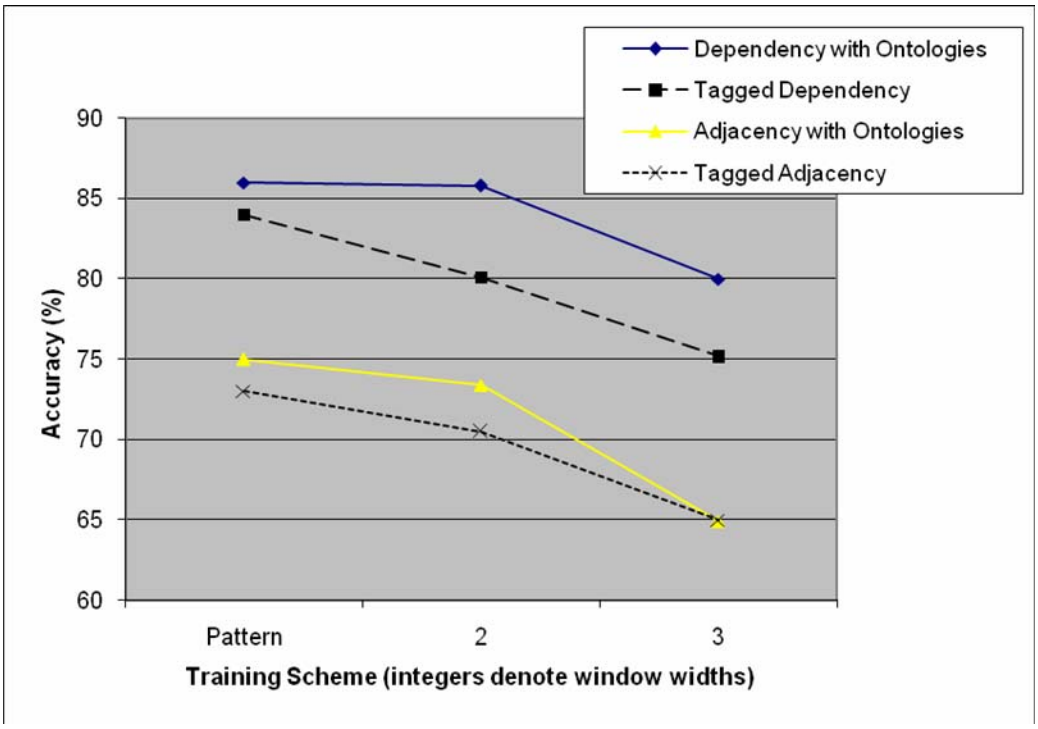


Figure 46: Accuracy of analyzing the test set using a tagged corpus and domain ontologies under the dependency and the adjacency model for the pattern training scheme that follows Pustejovsky (1993) in counting the occurrences of subcomponents and for the windowed training schemes with window widths of 2 and 3 words and comparison with the accuracy presented in figure 45.

Three training schemes have been used and the tuned analysis procedures applied to the test set. Figure 46 shows the resulting accuracy, with accuracy values from Figure 45 displayed with dotted lines. What we see is that the resulting accuracy is better in all cases and that the most significant improvement was in the dependency model with training scheme of window width 2 (85,8% from 80,1%).

SEMANTIC RELATEDNESS MEASURE

A. Comparison with human ratings of semantic relatedness

Chapter 7

To assess our relatedness measure's usefulness, we need to evaluate it against a "gold standard" of object relatedness. To that end we designed a detailed experiment in which human subjects were asked to assess the relatedness between pairs of objects. As Budanitsky and Hirst [Budanitsky et. al, 2006] found in a study comparing WordNet similarity measures human judgments give the best assessments of the "goodness" of a measure. We found that the experiment described in [Miller and Charles, 1991], which relies on human judgments, has become the benchmark in determining the similarity of words in NLP research (see [Budanitsky et. al, 2006,] [Jarmasz et. al, 2003], [Lin, 1998], [Resnik, 1995]). We reused their overall experimental design and adapted it to be usable for complex objects in an ontology as follows: First, we had to find a number of suitable object pairs from a number of ontologies. Then, we had to define an appropriate order in which those pairs were going to be presented to the subjects, who assessed the similarity of the pairs on a scale between zero (semantically unrelated) and one (highly related), according to their "relatedness of meaning".

The ontologies that we chose for our tests concern the domains of wine, pizza, the animal koala, soccer, people with pets, images and travel information and they are all OWL ontologies that can be found free in the web. The class hierarchies and the properties of the ontologies can be found in the APPENDIX. Note that the subjects' ability to relate to the ontology content is crucial for the success of the experiment. Lord [Lord et. al, 2003], for example, had to desist an evaluation with human subjects as experts in their application domain (biology) are difficult to find.

From the ontologies we have selected a number of concepts that we thought would be understandable to a general audience and combined them into pairs fulfilling the following criteria:

• At least two pairs should be in close vicinity in the ontology-graph.

• At least two pairs should be far apart in the ontology-graph.

• At least one pair should consist of a concept and its descendant/specialization.

The rest of the processes were paired in a way such that the processes' name, description, attributes, or properties featured some relatedness.

Chapter 7

We have obtained relatedness judgments from 25 human subjects, 10 from the computer science field that where shown the domain ontologies' structure and 15 from the liberal arts field that were used for the evaluation, for 85 pairs of concepts that we meet in the seven OWL domain ontologies for different domains (APPENDIX). The main goal is to compare the OntoNL sub-measures and the overall measure on how well they reflect human judgments. The subjects had the opportunity during the evaluation, to see the properties and the description (if any) of the concepts that they had to assess the relatedness.

The subjects were asked to assess the relatedness between two processes on a scale from 0 (semantically unrelated) to one (highly related). The users were asked to specify how they had made the assessment: 1. by concept name, 2. by concept description, 3. by concept properties, 4. a combination of 1-3, and 5. using other assessment method. This question captures in respect to which features of the object the relatedness was observed by the subjects – a notion that similarity researchers in the social sciences have found to be central [Gentner D. and Medina, J., 1998]. Finally, the subjects could add some comments on their assessment, that are crucial for determining the impact of influence of each relatedness measure that constitute the OntoNL semantic relatedness measurement.

Images Ontology

#	Pair		Humans LibArts	Humans CompScience	Overall	rel_{RS}	rel_{CD}	$rel_{PROPdirect}$	$rel_{PROPinverse}$
1	Video Text	Image	0,36	0,42	0,378	0	0,6	1	0,2857
2	Image Text	Image	0,71	0,68	0,701	0	0,8	1	0,425
3	Image	Mosaic	0,65	0,64	0,647	0,775	0,625	1	0,5857
4	Video Segment	Video Frame	0,54	0,48	0,522	0	0,5	1	0,425
5	Multimedia	Still Region	0	0,2	0,06	0	0	0,3	0,125
6	Image Text	Video Segment	0,18	0,04	0,138	0	0	0,3	0,2857
7	Video	Moving Region	0,77	0,68	0,743	0	0,725	1	0,725
8	Video	Still Region	0,29	0,32	0,299	0	0,3	0,3	0,325
9	Image	Moving Region	0,33	0,24	0,303	0	0,3	0,66	0,2857
10	Image	Still Region	0,72	0,74	0,726	0	0,775	1	0,625

Table 7: Human and computer ratings for the domain ontology Technical.owl

People Ontology

#	Pair		Humans LibArts	Humans CompScience	Overall	rel_{RS}	rel_{CD}	$rel_{PROPdirect}$	$rel_{PROPinverse}$
1	Person	Grass	0,08	0,1	0,086	0	0,2	0,25	0,05
2	Publication	Man	0,21	0,24	0,219	0,1435	0,25	1	0,2
3	Man	Vehicle	0,12	0,24	0,156	0,01	0,32	0,75	0,15
4	Person	Cat	0,28	0,36	0,304	0,1465	0,35	0,75	0,357
5	Person	Cow	0,28	0,36	0,304	0,2233	0,25	0,75	0,357
6	Vegeterian	Cow	0,62	0,84	0,686	0,6	0,9	1	0,428
7	Cow	Leaf	0,94	0,82	0,904	0,6	0,9	1	0,8
8	Cat Owner	Dog Liker	0,12	0,27	0,165	0	0,22	0,5	0,357
9	Newspaper	Old Lady	0,08	0,54	0,218	0	0,5	0,75	0,1875
10	Woman	Kid	0,75	0,88	0,789	0,425	0,825	1	0,5

Table 8: Human and computer ratings for the domain ontology People.owl

Koala Ontology

#	Pair		Humans LibArts	Humans CompScience	rel_{RS}	rel_{CD}	$rel_{PROPdirect}$	$rel_{PROPinverse}$
1	Animal	Forest	0,87	0,75	0,0	0,645	0,0	0,0
2	Student	Degree	0,94	0,75	0,166	0,925	0,0	0,0
3	Graduate Student	University	0,88	0,5	0,0	0,925	0,6	0,0
4	Gender	Degree	0,11	0,35	0,2	0,4	0,3	0,0
5	Gender	Rainforest	0,04	0	0,0	0,0	0,0	0,0
6	Koala	Dry Eucalypt Forest	0,84	0,75	0,0	0,0	0,0	0,0
7	Parent	Student	0,76	0,9	0,0	0,0	1,0	0,0
8	Parent	University	0,62	0,6	0,0	0,645	0,0	0,0
9	Student	Koala	0,3	0,5	0,0	0,0	1,0	0,0
10	Koala	Rain Forest	0,84	0,75	0,0	0,0	0,0	0,0

Table 9: Human and computer ratings for the domain ontology Koala.owl

Pizza Ontology

#	Pair		Humans LibArts	Humans CompScience	Overall	rel_{RS}	rel_{CD}	$rel_{PROPdirect}$	$rel_{PROPinverse}$
1	Margherita	Meat Topping	0,2	0,1	0,17	0	0	0,33	0,33
2	Vegeterian Pizza	Hot	0,43	0,68	0,505	0	0,72	0,625	0,33
3	Meaty Pizza	Meat Topping	0,98	0,8	0,926	0	0,825	1	0,4286
4	Fish Topping	Mild	0,5	0,6	0,53	0	0,662	0,625	0,33
5	FruttiDiMare	Mild	0,5	0,1	0,38	0	0,15	0,625	0,33
6	Non Vegeterian Pizza	Cajun	0,75	0,3	0,615	0	0,3	1	0,286
7	Ice Cream	Pizza	0	0	0	0	0	0,75	0,363
8	Ice Cream	Fruit Topping	0,68	0,1	0,506	0	0,2	0,825	0,2727
9	Country	Pizza	0,6	0,1	0,45	0	0	0,825	0,4
10	TobascoPepper Sauce	Hot	0,76	0,5	0,682	0	0,6	1	0,4286

Table 10: Human and computer ratings for the domain ontology Pizza.owl

Wine Ontology

Pair		Humans LibArts	Humans CompScience	Overall	rel_{RS}	rel_{CD}	$rel_{PROPdirect}$	$rel_{PROPinverse}$
Red Table Wine	Red	0,98	0,6	0,866	0,625	0,66	1	0,18
Dessert	Meal	0,68	0,65	0,671	0,33	0,663	1	0,125
Pinot	White	0,14	0,11	0,131	0,33	0	0,5	0,18
Dry Wine	Wine Sugar	0,04	0,7	0,238	0	0,6618	0,5	0,091
Table Wine	Winery	0,24	0,6	0,348	0,33	0,6618	0,5	0,091
Merlot	Dry	0,48	0,1	0,366	0,33	0	1	0,18
Ice Wine	White	0,78	0,4	0,666	0,33	0,3	1	0,18
Juice	Wine	0,18	0,04	0,138	0,0625	0	0,5	0,091
Meat	Red Table Wine	0,84	0,73	0,807	0,625	0,6618	1	0,25
Fruits	White Wine	0,92	0,73	0,863	0,625	0,7	1	0,25

Table 11: Human and computer ratings for the domain ontology Wine.owl

Travel Ontology

Pair		Humans LibArts	Humans CompScience	Overall	rel_{RS}	rel_{CD}	$rel_{PROPdirect}$	$rel_{PROPinverse}$
Accomondation Rating	City	0,78	0,56	0,714	0,3	0,678	0,7	0,33
Budget Accomondation	OneStar Rating	0,89	0,65	0,818	0,1	0,6	1	0,33
Hotel	Activity	0,98	0,99	0,983	0,1	0,711	1	0,33
Family Destination	Museums	0,85	0,88	0,859	0,3	0,9	1	0,66
Quiet Destination	Surfing	0,25	0,1	0,205	0,1	0	0,33	0
Bed and Breakfast	Sunbathing	0,11	0,05	0,092	0,1	0	0	0
Hotel	Quiet Destination	0,89	0,97	0,914	0,25	0,652	1	0,66
Luxury Hotel	Capital	0,91	0,97	0,928	0,6	0,758	0,75	
Capital	Safari	0,38	0,11	0,299	0	0,08	0,4285	0,33
BedandBreakfast	Yoga	0,08	0,21	0,119	0	0,38	0	0

Table 12: Human and computer ratings for the domain ontology Travel.owl

Soccer Ontology

#	Pair		Humans LibArts	Humans CompScience	Overall	rel_{RS}	rel_{CD}	rel_{PROP}	$rel_{PROPinverse}$
1	Player	Electronic Address Type	0,75	0,45	0,66	0	0,3333	1	0,017
2	Coach	Soccer Team	0,8	0,65	0,755	0	0,625	0,875	0,5
3	Player Object	Goal	0,9	0,75	0,855	0,48	0,58	1	0,2
4	Coach Object	Whistle	0,21	0,1	0,177	0,24	0,5416	0,1	0,017
5	Half Time	Referee	0,74	0,63	0,707	0,166	0,5948	0,666	0,5
6	Goal Post	Goal	0,98	0,6	0,866	0,48	0,7833	0,666	0,5
7	Net	Goalkeeper	0,87	0,68	0,813	0,224	0,7833	0,875	0,5
8	Goal Area	Forward	0,89	0,74	0,845	0,48	0,625	0,666	0,5
9	Penalty Mark	Penalty Period	0,92	0,6	0,824	0,8	0,5833	0,875	0,3
10	Soccer Field	SpectatorSeats	0,78	0,6	0,726	0	0,625	0,88	0,5
11	Score	Soccer Team	0,51	0,49	0,504	0,2	0,4945	0,5	0,017
12	Goal	Offside Kick	0,72	0,68	0,708	0,166	0,6629	0,66	0,8
13	Block	Goalkeeper Object	0,88	0,67	0,817	0,8	0,3201	0,88	0,3
14	Substitute	PlayerObject	0,63	0,51	0,594	0,53	0,5219	0,666	0,17
15	Tournament	SoccerTeamObject	0,6	0,48	0,564	0,21	0,4945	0,666	0,017
16	Flag	Game	0,18	0,12	0,162	0,21	0,5625	0,083	0,017
17	Applause	Yellow Card	0,5	0,52	0,506	0,63	0,6004	0,37	0,017
18	Chest	Doctor	0,43	0,08	0,325	0	0,1458	0,37	0,017
19	Penalty Kick	Red Card	0,84	0,78	0,822	0,08	0,7417	0,59	0,5
20	Offside Kick	Player Object	0,64	0,54	0,61	0,21	0,741	0,77	0,5
21	Pass	Spectator	0,14	0,08	0,122	0,08	0,1458	0,183	0,017

22	Dribble	Player Object	0,64	0,49	0,595	0,48	0,641	0,59	0,17
23	Foul Action	PlayerObject	0,62	0,51	0,587	0	0,341	0,63	0,17
24	Assistant Coach	Pre-Game Time	0,11	0,03	0,086	0	0,1875	0	0
25	Red Card	Referee Object	0,87	0,62	0,795	0	0,5	0,8	0,17

Table 13: Human and computer ratings for the domain ontology about Soccer

Our first objective was to investigate what are the values of the parameters f_1, f_2, w_1, w_2, w_3 for each ontology, and overall. Human subjects were used for the experiments. We observed that the optimal values of these parameters strongly depend on the ontology. Their optimal experimental values are shown in Table 14.

Ontology	rel_{PROP}		rel_{OntoNL}		
	f_1	f_2	w_1	w_2	w_3
Soccer Ontology	0,5	0,5	0,7	0,2	0,1
Wine Ontology	0,65	0,35	0,5	0,25	0,25
People Ontology	0,1	0,9	0,45	0,2	0,35
Pizza Ontology	0,65	0,35	0,5	0,27	0,23
Koala Ontology	0,99	0,01	0,25	0,65	0,1
Images Ontology	0,33	0,67	0,45	0,5	0,05
Travel Ontology	0,9	0,1	0,7	0,1	0,2

Table 14: The values of the relative weights f_1 and f_2 of eq. 26 and w_1 (for rel_{PROP}), w_2 (for rel_{CD}) and w_3 (for rel_{RS}) of eq. 36 for each one of the ontologies used for the specific experimentation

We observe that w_1 and f_1 are in general the most important of the weights, which implies that the number of common properties of two concepts is a significant factor in determining the relatedness. The conceptual distance measure (w_2) and the related senses measure (w_3) seem to have also significant impact, but in almost all ontologies (except the koala and images ontology for w2) the impact of each one of them was less than the common properties measure. Among these two measures the related senses measure (w_3) had a stronger impact than the conceptual distance measure (w_2) in two ontologies, while the conceptual distance measure (w_2) had a stronger impact in four ontologies. In Table 15 we present the humans (70% * relatedness measure value from the Liberal Arts Field subjects plus 30% * relatedness measure value from the Computer Science Field subjects) and the OntoNL measure ratings for each pair of the data set.

Using the optimal values for the parameters we studied how the computed relatedness measure among two concepts was correlated with the relatedness perceived by the human subjects. Table 16 shows the computed correlation coefficients between the system computed relatedness measure and the human subjects evaluated relatedness. In Table 16 the column **Pair** describes the pairs of concepts that are being tested for their relatedness, the column **Subjects** describes the semantic relatedness value that has been assigned by

the human subjects and the column **rel_{OntoNL}** contains the semantic relatedness value as it was computed by the OntoNL Semantic Relatedness Measure.

#	Pair from Image Ontology		Subjects	rel_{OntoNL}
1	Video Text	Image	0,378	0,3861386
2	Image Text	Image	0,701	0,6766375
3	Image	Mosaic	0,647	0,6763386
4	Video Segment	Video Frame	0,522	0,5266375
5	Multimedia	Still Region	0,06	0,0822375
6	Image Text	Video Segment	0,138	0,1306886
7	Video	Moving Region	0,743	0,7295875
8	Video	Still Region	0,299	0,2925375
9	Image	Moving Region	0,303	0,3341486
10	Image	Still Region	0,726	0,7244375

#	Pair from People Ontology		Subjects	rel_{OntoNL}
1	Person	Grass	0,086	0,0715
2	Publication	Man	0,219	0,226225
3	Man	Vehicle	0,156	0,162
4	Person	Cat	0,304	0,29961
5	Person	Cow	0,304	0,30649
6	Vegeterian	Cow	0,686	0,60834
7	Cow	Leaf	0,904	0,759
8	Cat Owner	Dog Liker	0,165	0,211085
9	Newspaper	Old Lady	0,218	0,2096875
10	Woman	Kid	0,789	0,56125

#	Pair from Koala Ontology		Subjects	rel_{OntoNL}
1	Animal	Forest	0,694	0,74525
2	Student	Degree	0,931	0,88185
3	Graduate Student	University	0,766	0,74325
4	Gender	Degree	0,182	0,34395
5	Gender	Rainforest	0,028	0
6	Koala	Dry Eucalypt Forest	0,785	0,70125
7	Parent	Student	0,742	0,76525
8	Parent	University	0,516	0,528
9	Student	Koala	0,36	0,3465
10	Koala	Rain Forest	0,813	0,35475

#	Pair from Pizza Ontology		Subjects	rel_{OntoNL}
1	Margherita	Meat Topping	0,17	0,198
2	Vegeterian Pizza	Hot	0,505	0,5013
3	Meaty Pizza	Meat Topping	0,926	0,730716
4	Fish Topping	Mild	0,53	0,4897
5	FruttiDiMare	Mild	0,38	0,3873
6	Non Vegeterian Pizza	Cajun	0,615	0,61716
7	Ice Cream	Pizza	0	0,42678
8	Ice Cream	Fruit Topping	0,506	0,501862
9	Country	Pizza	0,45	0,4695
10	TobascoPepper Sauce	Hot	0,682	0,685716

#	Pair from Travel Ontology		Subjects	rel_{OntoNL}
1	Accomondation Rating	City	0,714	0,6297
2	Budget Accomondation	OneStar Rating	0,818	0,7831
3	Hotel	Activity	0,983	0,8053
4	Family Destination	Museums	0,859	0,8862
5	Quiet Destination	Surfing	0,205	0,2179
6	Bed and Breakfast	Sunbathing	0,092	0,01
7	Hotel	Quiet Destination	0,914	0,8316
8	Luxury Hotel	Capital	0,928	0,6841
9	Capital	Safari	0,299	0,309055
10	BedandBreakfast	Yoga	0,119	0,076

#	Pair from Soccer Ontology		Subjects	rel_{OntoNL}
1	Player	Electronic Address Type	0,66	0,6978567
2	Coach	Soccer Team	0,755	0,71125
3	Player Object	Goal	0,855	0,808
4	Coach Object	Whistle	0,177	0,1965192

5	Half Time	Referee	0,707	0,5901407
6	Goal Post	Goal	0,866	0,65924
7	Net	Goalkeeper	0,813	0,76531
8	Goal Area	Forward	0,845	0,62758
9	Penalty Mark	Penalty Period	0,824	0,7689167
10	Soccer Field	SpectatorSeats	0,726	0,7144
11	Score	Soccer Team	0,504	0,435082
12	Goal	Offside Kick	0,708	0,6209728
13	Block	Goalkeeper Object	0,817	0,7194252
14	Substitute	PlayerObject	0,594	0,5888514
15	Tournament	SoccerTeamObject	0,564	0,540662
16	Flag	Game	0,162	0,18698
17	Applause	Yellow Card	0,506	0,4173628
18	Chest	Doctor	0,325	0,2634567
19	Penalty Kick	Red Card	0,822	0,56304
20	Offside Kick	Player Object	0,61	0,6893
21	Pass	Spectator	0,122	0,1536467
22	Dribble	Player Object	0,595	0,5598
23	Foul Action	PlayerObject	0,587	0,4769919
24	Assistant Coach	Pre-Game Time	0,086	0,0375
25	Red Card	Referee Object	0,795	0,6159

#	Pair from Wine Ontology		Subjects	rel_{OntoNL}
1	Red Table Wine	Red	0,866	0,67775
2	Dessert	Meal	0,671	0,595125
3	Pinot	White	0,131	0,2765
4	Dry Wine	Wine Sugar	0,238	0,343875
5	Table Wine	Winery	0,348	0,426375
6	Merlot	Dry	0,366	0,439
7	Ice Wine	White	0,666	0,514
8	Juice	Wine	0,138	0,19405
9	Meat	Red Table Wine	0,807	0,69045
10	Fruits	White Wine	0,863	0,7

Table 15: Human subjects and OntoNL measure ratings for the data set of different domains

Human Subjects Ratings				
Measure	rel_{PROP}	rel_{CD}	rel_{RS}	rel_{OntoNL}
Soccer Ontology	0,910	0,594	0,329	0,943
Wine Ontology	0,832	0,644	0,830	0,976
People Ontology	0,906	0,937	0,949	0,984
Pizza Ontology	0,657	0,77	-	0,863

Koala Ontology	0,492	0,846	0,285	0,857
Images Ontology	0,964	0,953	0,273	0,997
Travel Ontology	0,946	0,891	0,612	0,973

Table 16: The values of the coefficients of correlation between human ratings of relatedness and four computational measures; the three submeasures that constitute the OntoNL Semantic Relatedness Measure and the overall OntoNL measure with relative weights of Table 6

The results are satisfactory and show that the average OntoNL measure correlation for each ontology was almost always more than 0.9 and in 4 out of the 7 cases they were more than 0.95. The average correlation was 0.94.

From our research we observed that the subjects with computer science background had higher correlations with the system for the conceptual distance measure, while human subjects from liberal arts had higher correlations in general for the related properties measure. In all cases the calculated by the system weighted relatedness measure was higher correlated with the human subject evaluations than the correlations of the partial semantic measures (common properties, related senses, conceptual distance).

An observation mentioned above was the relatively large variability of the optimal weights for each ontology. We decided to experiment with the same set of weights for all the ontologies, to observe if the relatedness measures were drastically affected, and if they are still satisfactory. Table 17 shows the common set of weights used for all the experiments with all the ontologies.

OWL Domain Ontologies	rel_{PROP}		rel_{OntoNL}		
	f_1	f_2	w_1	w_2	w_3
	0,65	0,35	0,5	0,27	0,23

Table 17: The values of the relative weights f_1 and f_2 of rel_{PROP} and w_1 (for rel_{PROP}), w_2 (for rel_{CD}) and w_3 (for rel_{RS}) of the overall OntoNL Semantic Relatedness measure

Table 18 shows the correlations obtained between the systems computed values and the human subject computed values (second column). For comparison reasons the first column shows the correlations computed with different weights (copied from Table 16). Table 18 shows that the results obtained, as expected, are worse than the results obtained using different weights for each ontology. The average drop in correlation was 0.024, while the maximum drop in one ontology was 0.06. In this case (Koala Ontology) the

average correlation dropped below 0.8 (to 0.798). For this ontology however, even with its optimal weights the correlation was not very high (0.863).

Human Subjects Ratings		
Measure	rel_{OntoNL}	rel_{OntoNL}'
Soccer Ontology	0,943	0,918
Wine Ontology	0,976	0,974
People Ontology	0,984	0,966
Pizza Ontology	0,863	0,863
Koala Ontology	0,857	0,798
Images Ontology	0,997	0,973
Travel Ontology	0,973	0,935

Table 18: The values of the coefficients of correlation between human ratings of relatedness and four computational measures; the three submeasures that constitute the OntoNL Semantic Relatedness Measure and the overall OntoNL measure with relative weights of Table 8

Result Analysis

At the experimentation process the subjects were asked to specify how they had made the assessment: 1. by concept name, 2. by concept description, 3. by concept properties, 4. a combination of 1-3, and 5. using other assessment method. This question captured in respect to which features of the object the relatedness was observed by the subjects – a notion that similarity researchers in the social sciences have found to be central [Gentner D. and Medina, J., 1998]. Finally, the subjects could add some comments on their assessment, that were crucial for determining the impact of influence of each relatedness measure that constitute the OntoNL semantic relatedness measurement.

Parameters that we have found that affect the choice of weight are the following:

The language the ontology uses for its terminology. When ontologies are used directly from their source (web) a major factor of the rel_{RS} parameter's performance is the names that are used to describe the ontologies. If the names for the concepts and the logical relationships among the concepts used are near the "natural language" names the performance of the system is significantly better.

The number of the properties over the concepts. When the concepts of the ontology have a number of properties that specialize them over other concepts (the semantic network has a significantly greater number of edges over nodes) then the parameter

rel_{PROP} can participate with a great value of influence in the overall OntoNL semantic relatedness measure calculation.

The depth of the domain ontology. When the ontology is of a great depth then the conceptual distance needs to be assigned with a big relative weight because the information loss is significant over the inheritance.

Dealing with this uncertainty is crucial not only for the success application of the OntoNL Framework but in ontology engineering tasks such as domain modeling, ontology reasoning and concept mapping between ontologies.

To model this uncertainty that is dictated by the parameters discussed in the OntoNL Semantic Relatedness measure, we have developed a new probabilistic model for calculating the weight values of the measure.

To summarize, we have 5 weight values to calculate:

i. f_1 for rel_{OP1} (eq. 24)

ii. f_2 for rel_{OP2} (eq.25)

iii. w_1 for rel_{PROP} (eq. 26)

iv. w_2 for rel_{CD} (eq. 32)

v. w_3 for rel_{RS} (eq. 33)

The overall measure is

$$rel_{OntoNL}(c_1, c_2) = w_1 \times rel_{PROP}(c_1, c_2) + w_2 \times rel_{CD}(c_1, c_2) + w_3 \times rel_{RS}(c_1, c_2)$$

with $w_1 + w_2 + w_3 = 1, (w_1, w_2, w_3) > 0$ and $rel_{PROP}(c_1, c_2), rel_{CD}(c_1, c_2), rel_{RS}(c_1, c_2) \in [0,1]$

The goal that we set is to achieve the best cross correlation with the human subjects judgements. We cannot model the equation with this in mind because we do not want to experiment each time of use of the measure in different domains. So we need to develop a mechanism that will understand by the design where the ontology developer hid the semantics.

We first determine the features of the OWL Ontology structure that we essentially can state their impact in the OntoNL Semantic Relatedness Measure:

Feature 1

Let C be a set whose elements are called concepts or classes. Let $|C| \in N$ were $N := \{1,2,3,...\}$, be the number of all Classes of the OWL Domain Ontology.

Feature 2

Let P be a set whose elements are called Object Properties. Let $|P| \in N$ were $N := \{1,2,3,...\}$, be the number of all Object Properties of the OWL Domain Ontology.

Feature 3

Let H^C be a class hierarchy, a set of classes. H^C is a directed, transitive relation $H^C \subseteq C \times C$ which is also called class taxonomy. $H^C(C_s, C_i)$ is the set where C_s is a sub-class of C_i. The number of subclasses (C_s) for a class C_i is defined as $\left|H^C\left(C_s, C_i\right)\right|$

Feature 4

A specific kind of relations are attributes A. The function $att : A \rightarrow C$ with $range(A) := STRING$ relates concepts with literal values.

The values of these features can be computed univocally in each case of ontologies we used for the evaluation experiments.

The metrics we are proposing are not 'gold standard' measures of ontologies. Instead, the metrics are intended to evaluate certain aspects of ontologies and their potential for knowledge representation. Rather than describing an ontology as merely effective or ineffective, metrics describe a certain aspect of the ontology because, in most cases, the way the ontology is built is largely dependent on the domain in which it is designed. Ontologies modeling human activities (e.g., travel or terrorism) will have distinctly different characteristics from those modeling the natural (or physical) world (e.g. genomes or complex carbohydrates.

The category of metrics we are interested in is the schema metrics that evaluates ontology design and its potential for rich knowledge representation. Although we cannot know if the ontology design correctly models the knowledge, we can provide metrics that indicate the richness, width, depth, and inheritance of an ontology schema. The methodology for computing the values of these metrics can easily be integrated in the Ontology Processor module of figure 3.

The Ontology Metrics

Metric 1 (μ_1): Object Property Richness:

This metric reflects the richness of properties in an OWL ontology. An ontology that contains many object properties is richer than a taxonomy with only class-subclass relationships. The number of object properties that are defined for each class can indicate both the quality of ontology design and the amount of information pertaining to instance data. In general we assume that the more slots that are defined the more knowledge the ontology conveys.

Formally, the object property richness (*PR*) is defined as the average number of object properties per class. It is computed as the number of properties for all classes (P) divided by the number of classes (C).

$$PR = \frac{|P|}{|C|}$$

The result will be a real number representing the average number of object properties per class, which gives insight into how much knowledge about classes is in the schema. An ontology with a high value for the PR indicates that each class has a high number of object properties on the average, while a lower value might indicate that less information is provided about each class.

Metric 2 (μ_2): Inverse Object Property Richness

This metric may not be a popular metric for evaluating ontologies but it is crusial for the OntoNL Semantic Relatedness Measure. Formally, the inverse object property richness (PR_{inv}) is defined as the average number of inverse object properties per class (how many object properties of the class have also inverse properties). It is computed as the number of inverse properties for all classes (P_{inv}) divided by the number of classes (C).

$$PR_{inv} = \frac{|P_{inv}|}{|C|}$$

The result will be a real number representing the average number of inverse object properties per class.

Metric 3 (μ_3): Specificity Richness

This metric describes the specialization of information across different levels of the ontology's inheritance tree. This is a good indication of how well knowledge is grouped into different categories and subcategories in the ontology using Object Properties. An ontology that contains many relations other than class-subclass relations is richer than a taxonomy with only class-subclass relationships.

This metric can be measured for the whole schema or for a subtree of the schema.

Formally, the **specificity richness of the schema (SR)** is defined as the sum of all inner classes $\sum_{C_i \in C_{inner}}$ (all classes of the ontology except the leaf classes) of the number of properties of the subclass Cs of a class Ci ($P_{H^C(C_S,C_i)}$), minus the number of properties of the class Ci (P_{C_i}) divided by the number of properties of the subclass Cs ($P_{H^C(C_S,C_i)}$). This sum is divided by the number of the inner classes of the ontology.

$$SR = \frac{\sum_{C_i \in Cinner} \frac{\sum_{C_j \in H^C(C_S,C_i)} \frac{|P_{C_j}|}{|H^C(C_s,C_i)|} - |P_{C_i}|}{|P_{C_j}|}}{C_{inner}}$$

The result of the formula will be a real number representing the specialization of the ontology by moving vertically in the hierarchy. An ontology with a high SR would be an ontology that might reflect a very specialized type of knowledge that the ontology represents. while an ontology with a low SR would be an ontology that represents a wide range of general knowledge.

Metric 4 (μ_4): Inheritance Richness:

This metric describes the distribution of information across different levels of the ontology's inheritance tree or the fan-out of parent classes. This is a good indication of how well knowledge is grouped into different categories and subcategories in the ontology. This measure can distinguish a horizontal ontology from a vertical ontology or an ontology with different levels of specialization. A horizontal (or flat) ontology is an ontology that has a small number of subclasses. In contrast, a vertical ontology contains a large number of inheritance levels where classes have a small number of subclasses. This metric can be measured for the whole schema or for a subtree of the schema.

Formally, the inheritance richness of the schema (IR) is defined as the average number of subclasses per class.

$$IR = \frac{\sum_{C_i \in C} \left| H^C(C_S, C_i) \right|}{|C|}$$

The result of the formula will be a real number representing the average number of subclasses per class. An ontology with a low IR would be of a vertical nature, which might reflect a very detailed type of knowledge that the ontology represents. while an ontology with a high IR would be of a horizontal nature, which means that ontology represents a wide range of general knowledge.

Metric 5 (μ_5): Readability

This metric indicates the existence of human readable descriptions in the ontology, such as comments, labels, or captions. This metric can be a good indication if the ontology is going to be queried and the results listed to users.

Formally, the readability (R) of a class c is defined as the sum of the number attributes that are comments and the number of attributes that are labels the class has.

$$R = \left|A, A = rdfs:comment\right| + \left|A, A = rdfs:label\right|$$

The result of the formula will be an integer representing the availability of human-readable information for the instances of the current class.

If the readability is equal to zero, then we define the readability as the average number of classes with one-word string names per all the classes of the ontology.

$$R = \frac{\left|A, A = rdfs:one\_word\_ID\right|}{\left|A, A = rdfs:ID\right|}$$

We should recall here that for the success definition of the ontology metrics we used the feedback the users gave us at the evaluation process. It is under consideration the definition of more or more accurate ontology metrics. To proceed with this we will need extra experimentation data and knowledge of ontology engineering.

The OntoNL Semantic Relatedness Measure's Weight Value Calculation

We are going to use methodologies from Linear Programming field in order to compute the impact of each metric to the weights of the OntoNL Semantic Relatedness Measure. A *Linear Programming* problem is a special case of a *Mathematical Programming* problem. From an analytical perspective, a mathematical program tries to identify an *extreme* (i.e., minimum or maximum) point of a function $f(x_1, x_2, ..., x_n)$, which furthermore satisfies a set of constraints, e.g., $g(x_1, x_2, ..., x_n) \geq b$. Linear programming is the specialization of mathematical programming to the case where both, the function f - to be called the ***objective function*** - and the ***problem constraints*** are ***linear***.

The general form for a Linear Programming problem is as follows:

Objective Function:

$$\max/\min f(X_1, X_2, ..., X_n) := c_1X_1 + c_2X_2 + ... + c_nX_n$$

Technological Constraints:

$$a_{i1}X_1 + a_{i2}X_2 + ... + a_{in}X_n \left(\leq=\geq\right) b_i, i = 1, ..., m$$

Sign Restrictions:

$$\left(X_j \geq 0\right) or \left(X_j \leq 0\right) or \left(X_j urs\right), j = 1, ..., n$$

where ``urs" implies *unrestricted in sign*.

We are going to use the results that we have obtained empirically through experimentation as training data to determine the exact weight values of the metrics that we think that affect the parameters (f_1, f_2, w_1, w_2, w_3) of the OntoNL Semantic Relatedness Measurement. We observe that the optimal choice of f_1, f_2, w_1, w_2, w_3 is affected by the characteristics of the ontology structure and description and the ontology metrics defined above. We should recall here that for the success definition of the ontology metrics we also used the feedback the users gave us at the evaluation process. We can write that:

- $f_1 = f_1(\mu_1, \mu_3)$ (the influence parameter of the rel_{OP1} (see chapter 5)) to indicate that f_1 dependes on the ontology metrics μ_1 (object property richness) and μ_3 (specificity richness). This is because the ontology metric μ_1 describes the plurality of object properties of the domain ontology and the ontology metric μ_3 describes the specialization of information across different levels of the ontology's inheritance tree.

- $f_2 = f_2(\mu_1, \mu_2, \mu_3)$ (the influence parameter of the rel_{OP2} (see chapter 5)) is affected by the ontology metrics μ_1 (object property richness), μ_2 (inverse object property richness) and μ_3 (specificity richness)

- $w_1 = w_1(\mu_1, \mu_2, \mu_3)$ (the influence parameter of rel_{OP} (see chapter 5)) is affected by the ontology metrics μ_1 (object property richness), μ_2 (inverse object property richness) and μ_3 (specificity richness)

- $w_2 = w_2(\mu_1, \mu_3, \mu_4)$ (the influence parameter of rel_{CD} (see chapter 5)) is affected by the ontology metrics μ_1 (object property richness), μ_3 (specificity richness) and μ_4 (specificity richness)
- $w_3 = w_3(\mu_4, \mu_5)$ (the influence parameter of rel_{RS} (see chapter 5)) is affected by the ontology metrics μ_4 (specificity richness) and μ_5 (readability)

We want to determine the exact weight values (*c* values as they were presented in the objective function of the Linear Programming Methodology) of the metrics that affect the influence parameters (f_1, f_2, w_1, w_2, w_3) of the OntoNL Semantic Relatedness Measurement. To do that we have computed the ontology metrics (μ_1: object property richness, μ_2: inverse object property richness, $\mu3$: specificity richness, μ_4: specificity richness, μ_5: readability) for the 7 OWL domain ontologies that we have used for experimentation. Then we defined the objective functions to represent the problem as a linear programming problem. Since we assume a linear dependency of the parameters f_1, f_2, w_1, w_2, w_3 from the ontology metrics we can write:

$$f_1 \equiv f_1(\mu_1, \mu_3) \coloneqq c_{11} \times \mu_1 + c_{13} \times \mu_3 - e_1$$

$$f_2 \equiv f_2(\mu_1, \mu_2, \mu_3) \coloneqq c_{21} \times \mu_1 + c_{22} \times \mu_2 + c_{23} \times \mu_3 - e_2$$

$$w_1 \equiv w_1(\mu_1, \mu_2, \mu_3) \coloneqq c_{31} \times \mu_1 + c_{32} \times \mu_2 + c_{33} \times \mu_3 - e_3$$

$$w_2 \equiv w_2(\mu_1, \mu_3, \mu_4) \coloneqq c_{41} \times \mu_1 + c_{43} \times \mu_3 + c_{44} \times \mu_4 - e_4$$

$$w_3 \equiv w_3(\mu_4, \mu_5) \coloneqq c_{54} \times \mu_4 + c_{55} \times \mu_5 - e_5$$

In these equations c_{ij} are constants and e_i's are error values. We will compute the values of the parameters c_{ij} and e_i so that the values computed for f_1, f_2, w_1, w_2, w_3 for each one of the ontologies that we will use for training is minimized. As training ontologies we will use the ones that we described above (and shown in the APPENDIX). For each one of these ontologies we have calculated the values of μ_1, μ_2, μ_3, μ_4 and μ_5. We also used as values for f_1, f_2, w_1, w_2, w_3 the values that gave the maximum correlations for the concept

relatedness in the user experiments (table 14). The seven OWL Domain Ontologies that were used for experimentation were (see APPENDIX for more details):

1.The Soccer Ontology
2. The Wine Ontology
3. The People Ontology
4. The Pizza Ontology
5. The Koala Ontology
6. The Images Ontology
7. The Travel Ontology

This gave us a system of 175 equations. We used a Linear Solver to compute the different *c* values (the weight values of the ontology metrics that affect the influence parameters of the OntoNL Semantic Relatedness Measure) and the errors *e* between the influence parameters value computed by the linear solver for each of the ontologies and the influence parameter value computed manually to achieve the best correlation with the humans judgements, in order to minimize the objective functions to zero. By using this methodology, when an OWL Domain Ontology enters the OntoNL Framework automatically by calculating the values of the metrics and by multiplying them with the corresponding *c* values we will get the values of the influence parameters of the OntoNL Semantic Relatedness Measure.

The results of the linear programming procedure for each influence parameter of the OntoNL Semantic Relatedness Measure are presented in Tables 19-23.

[The weight values definition problem for f_1.xls]		
Name	**Original Value**	**Final Value**
c1	**0**	**0,758196161**
c3	**0**	**0,435593623**
e11	0	0,05333553
e12	0	-0,074884789
e13	0	0,130046084
e14	0	-0,086183898
e15	0	-0,002707919
e16	0	-0,004941855
e17	0	-0,014663153

Table 19: The values of the parameters that influence the metrics used for calculation of the weight value f_1 of the OntoNL Semantic Relatedness Measure and the minimized errors for each one of the seven ontologies used for experimentation.

In table 19 we see the results that we have obtained from the linear solver for the objective function $f_1(\mu_1, \mu_3) := c_1 \times \mu_1 + c_3 \times \mu_3 - e_1$, were c_1 and c_3 values are the weight values that we will use to multiply the computed ontology metrics μ_1 and μ_3 respectively in order to define the influence parameter f_1 of a domain ontology we want to process. The e11-e17 values are the deviations from the human judgements for each one of the seven ontologies used for experimentation.

[The weight values definition problem for f_2.xls]		
Name	**Original Value**	**Final Value**
c1	**0**	**0,049345079**
c2	**0**	**0,840100697**
c3	**0**	**0,047572988**
e21	0	0,136529522
e22	0	0,022606066
e23	0	-0,070313878
e24	0	0,022606066
e25	0	0,058533106
e26	0	-0,148934617
e27	0	-0,021026264

Table 20: The values of the parameters that influence the metrics used for calculation of the weight value f_2 of the OntoNL Semantic Relatedness Measure and the minimized errors for each one of the seven ontologies used for experimentation.

In table 20 we see the results that we have obtained from the linear solver for the objective function $f_2(\mu_1, \mu_2, \mu_3) := c_1 \times \mu_1 + c_2 \times \mu_2 + c_3 \times \mu_3 - e_2$, were c_1, c_2 and c_3 values are the weight values that we will use to multiple the computed ontology metrics μ_1, μ_2 and μ_3 respectively in order to define the influence parameter f_2 of a domain ontology we want to process. The e21-e27 values are the deviations from the human judgements for each one of the seven ontologies used for experimentation.

[The weight values definition problem for w_1.xls]		
Name	**Original Value**	**Final Value**
c1	**0**	**0,549347616**
c2	**0**	**0,310262499**
c3	**0**	**0,26487096**
e31	0	0,06518881
e32	0	-0,066029702
e33	0	-0,117953097
e34	0	-0,066029702
e35	0	0,021163584
e36	0	-0,267102916

e37	0	-0,061115641

Table 21: The values of the parameters that influence the metrics used for calculation of the weight value w_1 of the OntoNL Semantic Relatedness Measure and the minimized errors for each one of the seven ontologies used for experimentation.

In table 21 we see the results that we have obtained from the linear solver for the objective function $w_1(\mu_1, \mu_2, \mu_3) := c_1 \times \mu_1 + c_2 \times \mu_2 + c_3 \times \mu_3 - e_3$, were c_1, c_2 and c_3 values are the weight values that we will use to multiple the computed ontology metrics metrics μ_1, μ_2 and μ_3 respectively in order to define the influence parameter w_1 of a domain ontology we want to process. The e31-e37 values are the deviations from the human judgements for each one of the seven ontologies used for experimentation.

[The weight values definition problem for w_2.xls]		
Name	**Original Value**	**Final Value**
c1	**0**	**0,363197897**
c3	**0**	**0,046685606**
c4	**0**	**0,245670362**
e41	0	0,08867345
e42	0	0,016971459
e43	0	-0,043044046
e44	0	0,009254977
e45	0	-0,212717517
e46	0	-0,066789378
e47	0	0,207651054

Table 22: The values of the parameters that influence the metrics used for calculation of the weight value w_2 of the OntoNL Semantic Relatedness Measure and the minimized errors for each one of the seven ontologies used for experimentation.

In table 22 we see the results that we have obtained from the linear solver for the objective function $w_2(\mu_1, \mu_3, \mu_4) := c_1 \times \mu_1 + c_3 \times \mu_3 + c_4 \times \mu_4 - e_4$, were c_1, c_3 and c_4 values are the weight values that we will use to multiple the computed ontology metrics metrics μ_1, μ_3 and μ_4 respectively in order to define the influence parameter w_2 of a domain ontology we want to process. The e41-e47 values are the deviations from the human judgements for each one of the seven ontologies used for experimentation.

[The weight values definition problem for w_3.xls]		
Name	**Original Value**	**Final Value**
c4	**0**	**0,278023071**
c5	**0**	**0,315776889**
e51	0	0,016872301
e52	0	-0,056071167
e53	0	-0,098853789

e54	0	-0,037958858
e55	0	0,064273687
e56	0	0,149284072
e57	0	-0,037546245

Table 23: The values of the parameters that influence the metrics used for calculation of the weight value w_3 of the OntoNL Semantic Relatedness Measure and the minimized errors for each one of the seven ontologies used for experimentation.

In table 23 we see the results that we have obtained from the linear solver for the objective function $w_3(\mu_4, \mu_5) := c_4 \times \mu_4 + c_5 \times \mu_5 - e_5$, were c_4 and c_5 values are the weight values that we will use to multiple the computed ontology metrics metrics μ_4 and μ_5 respectively in order to define the influence parameter w_3 of a domain ontology we want to process. The e51-e57 values are the deviations from the human judgements for each one of the seven ontologies used for experimentation.

From the results presented in the Tables 19-23 we can see the most important deviations from the empirical results we obtained by experiment with the human subjects. For the factor f_1 we get the largest deviation for the ontology *People* since it is an ontology with a small number of Object Properties in comparison to the Classes that it has.

For the factor f_2 we get the largest deviations for the ontologies *Soccer* and *Images* because of the little number of the inverse Object Properties for the *Soccer* Ontology and the lack of Specificity Richness as it was defined earlier in the Metrics for the *Images* ontology.

For the factor w_1 we get the largest deviations for the ontologies *People* and *Images* because of the reasons that influence the bad performance in the calculation of the values of f_1 and f_2.

For the factor w_2 we get the largest deviations for the ontologies *Koala* and *Travel* because they are quite flat as domain ontologies, they do not have a large Inheritance Richness as it was defined in the Metrics definition.

For the factor w_3 we get the largest deviation for the ontologies *Images* because it does not have descriptions, like comments and labels and because the names of the classes are mainly two word strings, so the methodology of finding related senses using the WordNet did not have good results.

In conclusion, the methodology of linear programming helped the determination of an automatic way for calculating the influence parameters (f_1, f_2, w_1, w_2, w_3) of the OntoNL Semantic Relatedness Measure. The methodology showed that with the correct definition of ontology metrics we get realistic results for the relatedness of concepts of a domain ontology. The deviations from the human judgements were expected if we confront the ontology metrics and the ontologies we used for experimentation. The methodology we used to define those ontology metrics was based on the feedback of the users we used for the experimentation. By using a more systematic way of extracting the knowledge and experience of the users maybe could lead to a more accurate definition of ontology metrics with even better results in comparison with human judgements.

B. An application-based evaluation of measures of relatedness

We have performed an application-based evaluation of the OntoNL Semantic Relatedness Measure. The application used the OWL Ontology for the domain of soccer (http://lamia.ced.tuc.gr/ontologies/AV_MDS03/soccer), because it is a large and very specific ontology. Also, the context of the ontology is familiar with the users.

We first asked the users to submit requests. We gathered a total of 20 requests concerning a disambiguation type 1[1] and 40 requests concerning a disambiguation type 2[2] of the algorithm, after eliminating any duplicates. We distinguished the types of expressions based on the OntoNL Language Model in 3 different types:

1. Subject Part
2. Subject Part – Conjuctive/Disjunctive/Plain Verb Phrase
3. Subject Part – Verb – Conjuctive/Disjunctive/Plain Object Part

We have presented to the human subjects, the resulted concepts related to the subject concept of their request. The users replied the ranking position of their correct response in

[1] The query contains generally keywords that can be resolved by using only the ontology repository (ontological structures and semantics)

[2] One of the subject or object part of the language model cannot be disambiguated by using the ontology repository

mind and this experiment was conducted twice. Since our results are a ranked list, we use a scoring metric based on the inverse rank of our results, similar to the idea of Mean and Total Reciprocal Rank scores described in [Radev et al, 2002], which are used widely in evaluation for information retrieval systems with ranked results. Hence our *precision* and *recall* are defined as:

$$PRECISION = \frac{\sum \frac{1}{ranking}}{\#requests}$$

$$RECALL = \frac{n(accepted\_ranking)}{\#requests}$$

The *precision* is depended on the ranking position of the correct related concept to the subject concept of the request. The *recall* is depended on the number of the related concepts the algorithm returns. In Table 24 we present the precision and recall scores we obtained for the two most complex datasets of request types for the disambiguation type (2).

DataSet	Precision	Recall (n = 3)	Recall (n =5)	Recall (n =8)
Subject Part – Conjuctive/Disjunctive/Plain Verb Phrase (15 requests)	49%	60%	86,7%	100%
Subject Part – Verb – Conjuctive/Disjunctive/Plain Object Part (25 requests)	39,7%	52%	76%	92%
Total	44%	55%	80%	95%

Table 24: Quality metrics for the first iteration

What we see is that overall we gain more than 50% of the correct matches in the first three hits and that the requests of type Subject Part – Conjuctive/Disjunctive/Plain Verb Phrase had better precision and recall than the requests of type Subject Part – Verb – Conjuctive/Disjunctive/Plain Object Part requests. This is because we use the verbs in this application to disambiguate in a more sufficient way the RelationTypes modeled in the OWL Domain Ontology for Soccer that is based on the MPEG-7 (see Chapter 6).

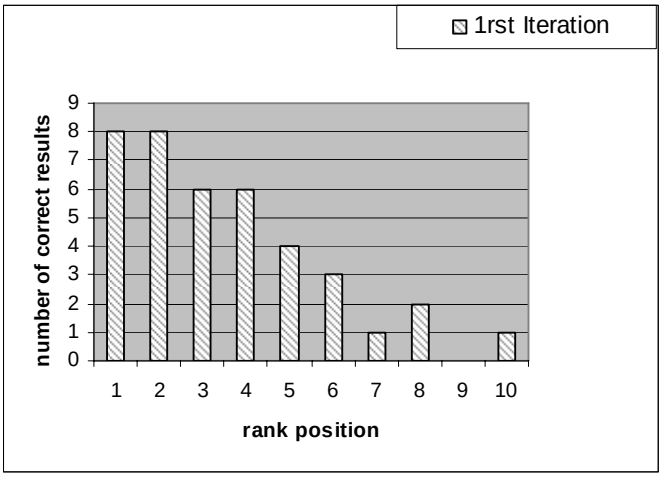


Figure 47: The precision of the OntoNL measure to the user input for the requests of disambiguation type (2)

DataSet	Precision	Recall (n = 3)	Recall (n =5)	Recall (n =7)
Subject Part – Conjuctive/Disjunctive/Plain Verb Phrase (10 requests)	47%	70%	90%	100%
Subject Part – Verb – Conjuctive/Disjunctive/Plain Object Part (10 requests)	46,1%	60%	100%	-
Total	46,63%	65%	90%	100%

Table 25: Quality metrics for the second iteration

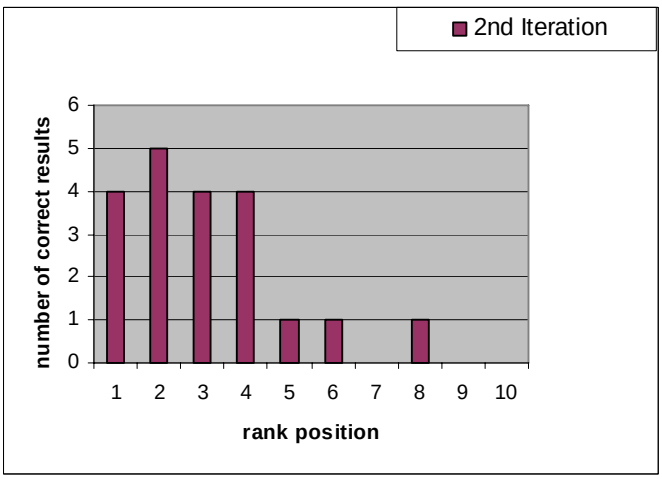


Figure 48: The precision of the OntoNL measure to the user input for the requests of disambiguation type (2) for a second iteration

After this experiment we asked the users to submit new requests and we once again gathered 20 requests of disambiguation type (2). In **Table 25** we present the precision and recall scores we obtained for the two most complex datasets of request types for the disambiguation type (2) and for a second iteration of the experiment.

What we see here is that in total we gain a 65% of the correct matches in the first 3 results of the OntoNL Disambiguation Procedure an a 90% in the first 5 results. The overall conclusion that derives is that in a second iteration of tests the performance was better because of the familiarity of the users using the system increased. In more details, the request type Subject Part – Conjuctive/Disjunctive/Plain Verb Phrase has a better precision but the request type Subject Part – Verb – Conjuctive/Disjunctive/Plain Object Part has a better recall.

We also present the overall satisfaction of users with respect to the effectiveness of the results compared against a keyword-based search (Figure 49). Overall, the performance decreases a little as the complexity of the language model increases, but as shown in Figure 49 , we get the correct results sooner and faster against a keyword-based search.

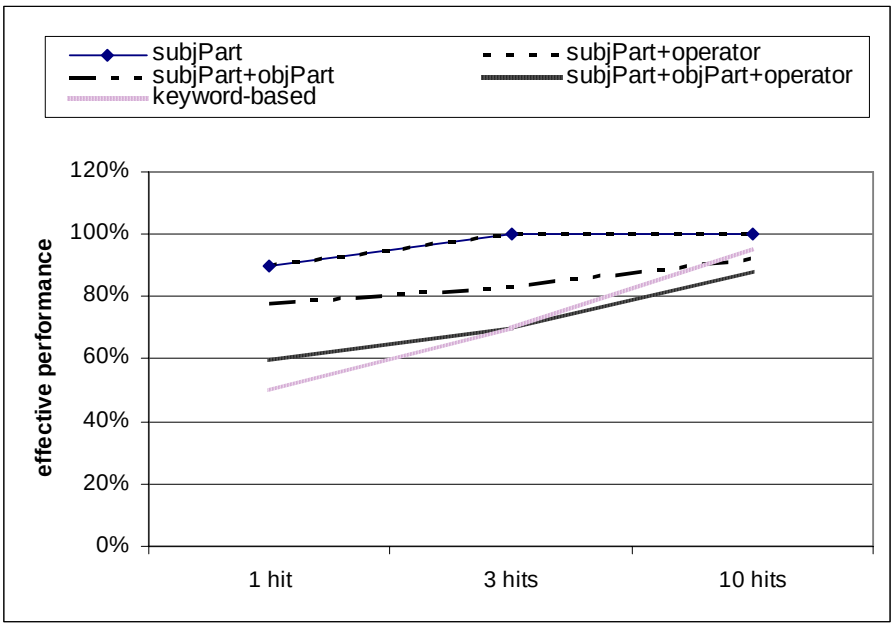


Figure 49. The effectiveness of the NL2DL in the domain of soccer against a keyword-based search

Summary

In this section we have presented the OntoNL Evaluation Framework that is based on three types of evaluation: the Adequacy, the Diagnostic and the Performance Evaluation. After the description of these types we presented the evaluation results for each one of them.

The adequacy evaluation addressed the evaluation of the usability of the interfaces according to a defined set of heuristics. The graphical user interface that was evaluated is the application of the OntoNL Framework in the domain of soccer as it was described in the implementation chapter and is presented in figure 36. This evaluation had as a starting point the ten heuristics of Nielsen [Nielsen, 1994].

The Diagnostic Evaluation was about testing the range of possible sentences that the OntoNL system can parse and disambiguate linguistically. It was conducted by system developers and it referred to the successful parsing of natural language expressions and to different categories of grammatical relations combinations that need to be disambiguated. The qualitative evaluation concerns measuring the effectiveness of the noun compound bracketing mechanism, the semantic relatedness measure and an application-based evaluation of measures of relatedness.

We evaluated our parser against the Minipar [Lin, 1998] and the Link Parser [Sleator and Temperlay, 1993].The results were satisfactory and provided a basis for the parser improvement. We also evaluated the OntoNL dependency model against the OntoNL adjacency model for noun compound bracketing and conclude that the adjacency model is superior and was used in the final model.

We presented the evaluation of the OntoNL Semantic Relatedness Measurement firstly by a comparison with human ratings and secondly by using an application-based evaluation. We concluded to the parameters that affect the choice of the weight value of each one of the sub-measures developed to comprise the OntoNL measure and we used the evaluation empirical results and Linear Programming to define the values of these weights by defining ontology metrics that influence the weights of the OntoNL measure.

Chapter 8

Conclusions

In this thesis we presented the design and implementation of the OntoNL Framework, a natural language interface generator for knowledge repositories, as well as a natural language system for interactions with multimedia repositories which was built using the OntoNL Framework.

It is well known that a problem with the natural language interfaces to information repositories is the ambiguities of the requests, which may lead to lengthy clarification dialogues. Due to the complexity of natural language, reliable natural language understanding is an unaccomplished goal in spite of years of work in fields like Artificial Intelligence, Computational Linguistics and other. The natural language understanding could be approached by applying methods for consulting knowledge sources such as domain ontologies. Ontologies are usually expressed in a formal knowledge representation language so that detailed, accurate, consistent, sound, and meaningful distinctions can be made among the classes (general concepts), properties (those concepts may have), and the relations that exist among these concepts. A module dealing with ontologies can perform automated reasoning using the ontologies, and thus provide advanced services to intelligent applications such as: conceptual/semantic search and retrieval, software agents, decision support, speech and natural language understanding and knowledge management.

The methodology that we have developed is reusable, domain independent and works with input only the OWL ontology that was used as a reference schema for constructing a knowledge repository. The methodology depends on a semantic relatedness measure that we have developed for domain ontologies that concludes to semantic ranking. The semantic ranking is a methodology for ranking related concepts based on their commonality, related senses, conceptual distance, specificity and semantic relations. This

procedure concludes to the natural language representation for information retrieval using an ontology query language, the SPARQL. The SPARQL queries are ranked based on the semantic relatedness measure value that is also used for the automatic construction of the queries.

This methodology is integrated in the OntoNL Framework, a natural language interface generator to knowledge repositories. We have presented the OntoNL Framework for building natural language interfaces to semantic repositories, as well as the implementation of a natural language interaction interface for semantic multimedia repositories which was built using the OntoNL Framework. The application of the OntoNL Framework addresses a semantic multimedia repository with digital audiovisual content of soccer events and metadata concerning soccer in general, has been developed and demonstrated in the 2nd and 3rd Annual Review of the DELOS II EU Network of Excellence (IST 507618) (http://www.delos.info/).

The OntoNL Framework implements a software platform that automates to a large degree the construction of natural language interfaces for knowledge repositories. To achieve the applicability and reusability of the OntoNL Framework in many different applications and domains, the supporting software is independent on the domain ontologies.

The software components of the OntoNL Framework address uniformly a range of problems in sentence analysis each of which traditionally had required a separate mechanism. A single architecture handles both syntactic and semantic analysis, handles ambiguities at both the general and the domain specific environment. At the same time, the Framework has been designed in a way to avoid dependencies with the information repository so that it becomes reusable in different applications with different domain semantics.

The OntoNL Software Engineering Framework has two major objectives. The first is to minimize the cost of building natural language interfaces to information systems by providing reusable software components that can be used in different application domains and knowledge bases, and adapted with a small cost to a new environment. The second is to do semantic processing, exploiting domain ontologies in order to reduce ambiguities in

a particular domain. The output of a natural language request is a ranked set of queries in an ontology query language.

The Framework in a particular application environment has to be supplied with domain ontologies (encoded in OWL) which are used for semantic processing. The user input in an application environment is natural language requests and WH-questions (who, were, what, etc.). The output for a particular input NL query is a set of one or more weighted disambiguated to the specific domain queries, encoded in SPARQL. We choose SPARQL as the query language to represent the natural language queries since SPARQL is defined in terms of the W3C's RDF data model and will work for any data source that can be mapped into RDF. If the environment uses a different type of repository than OWL-SPARQL, a module has to be implemented that does the mapping from the SPARQL encoded queries to the schema and query language that the environment uses (Relational Schema-SQL, XML Schema-XQUERY, etc). Since this transformation is Schema dependent it is not automated within the Framework software.

The main components of the OntoNL provide Linguistic Analysis and Ontology Processing for Semantic Disambiguation. The Linguistic Analysis includes components for POS tagging, Noun Compound Bracketing, Grammatical Relations Discovery, and Synonym and Sense Discovery. To perform its functions it uses input from the WordNet which provides information about word synonyms and the part of speech that a word is (verb, noun, etc.). The Semantic Disambiguation Module of the OntoNL is responsible for domain specific disambiguation and result ranking. The language model used in OntoNL supports both the Linguistic Analyzer and the Ontology Processor.

Disambiguation in natural language processing is used to eliminate the possible senses that can be assigned to a word in the discourse, and associate a sense which is distinguishable from other meanings. However, WordNet gives only generic categories of senses and not domain specific. Thus it is clear that much better semantic disambiguation can be done when domain knowledge is available in the form of ontologies. The purpose of the OntoNL Semantic Disambiguation Module is to use information of the OntoNL Ontology Processor in the OntoNL Framework, in order to do semantic disambiguation of the natural language queries. The input in the Ontology Processor is OWL Ontologies and

instances of the language model produced by the Syntactic Analyzer. The output is disambiguated sentences expressed as queries in SPARQL, or in the case that complete disambiguation is not possible, a set of ranked SPARQL queries.

In particular, the common types of ambiguity encountered in the OntoNL Framework are:

1. The natural language expression contains general keywords that can be resolved by using only the ontology repository (ontological structures and semantics).
2. One of the subject or object part of the language model contains terms that cannot be disambiguated by using the ontology repository.
3. Neither the subject nor the object part contains terms disambiguated by using the ontological structures.

Figure 49 shows the general steps of the semantic disambiguation algorithm used in OntoNL using UML Activity Diagram notation. The approach is general for any OWL DL or Full domain ontology.

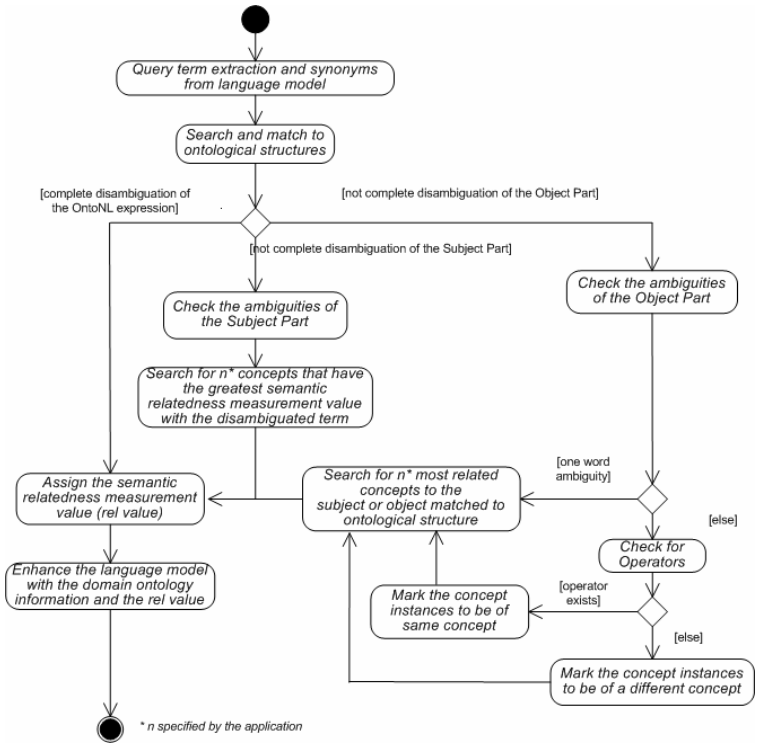


Figure 50. The OntoNL Semantic Disambiguation procedure

The input to the algorithm are instances of the language model, which include terms extracted from the natural language input, their synonyms, and their tagging according to the language model constructs. The algorithm searches to see if there is a correspondence between the naming of the language model instance and the ontological structures. If there is a complete match, a Relatedness Value measure is assigned with value 1 to indicate the complete relevance of the sentence with the specific domain. If the disambiguation is not complete (either in the Subject Part or the Object Part) the algorithm checks for the number of the terms that show ambiguity. If the ambiguity is in theSubject Part then the algorithm checks for a number specified by the application of ontology concepts that have the greatest relatedness value with the disambiguated term of the request. If there is only one term with an ambiguity in the Object Part then the algorithm checks and retrieve the output of the OntoNL Ontologies Processor for a number, specified by the application, of the most related concepts to the concept that comprise the subject or the object part (if the ambiguity is in the object or the subject part respectively) of the expression. If in a part of the expression are more than one terms with ambiguities the algorithm checks for operators (or/and). In the existence of an operator the algorithm considers the terms to be concept instances of the same concept of the domain ontology. In the absence of an operator the algorithm considers the terms to be concept instances of a different ontology concept. Then the algorithm searches for a number, specified by the application, of the most related concepts to the concept that found a correspondence to the ontological structures and assigns the relatedness measure, already calculated by the OntoNL Ontologies Processor. The last activity of the algorithm is to enhance the Ontology Structure class of the OntoNL Language Model (see figure 15) with the corresponding ontology concepts to natural language terms in the class attribute and with the relatedness measurement value the value attribute.

When a query cannot be disambiguated completely from the OntoNL Semantic Disambiguation procedure, OntoNL uses a Semantic Relatedness Measure to suggest weighted possible interpretations of the user request. . To that purpose, OntoNL borrows and expands ideas from the research of Semantic Relatedness of concepts in semantic networks. The Relatedness Matrix contains a weight of relatedness (Relatedness Measure)

between any two concepts. Intuitively, tightly interrelated concepts or clusters of concepts in the ontology are more likely to be the object of the user natural language interactions.

The relatedness measure depends on the semantic relations defined by properties in OWL. Properties can be used to state relationships between individuals (named ObjectProperties) or from individuals to data values (named DatatypeProperties). Based on the semantic relations when we detect that a source concept-class is immediately related via an ObjectProperty with the target concept, the relatedness value is set to 1

The algorithm also takes into account the semantic relation of OWL:EquivalentClass. The class that is OWL:EquivalentClass with a source class has a similarity (not relatedness) value 1. In our computations, the classes related to the source class of the ontology are also related with the same value to the equivalent class.

In all other cases the relatedness value computation is based on the following factors: the commonality (based on the semantic relations and the conceptual distance) and the related senses.

The commonality depends on the amount of the common information two concepts share. The commonality measure has two factors: The position of the concepts relatively to the position of their most specific common subsumer (how far is their common father) and the reciprocity of their properties (if the connecting OWL ObjectProperties have also inverse properties). The position of the concepts relatively to the position of their common subsumer will be examined by the conceptual distance and the specificity measurement.

After the syntactic and semantic disambiguation, we have concluded to the subject of the query, specialized by additional description that forms the object part or possible object parts of the query. We need a formal way to represent the query, a standardized query language that will meet the specification of the ontology language (OWL) and will be easily mapped to various forms of repository constructions. Although we could in principle use an internal representation of the preprocessed NL interactions, we opted to use a representation that is near to the languages used in the Semantic Web, so that when the repository is based on OWL or RDF to be able to directly use it to access the repository. We choose SPARQL as the query language to represent the natural language

queries since SPARQL is defined in terms of the W3C's RDF data model and will work for any data source that can be mapped into RDF.

To provide an automatic construction of SPARQL queries we need at any point to define the path that leads from the subject part to the object part of the natural language expression by taking into account the constraints that are declared from the keywords and the relatedness value between the related classes of the ontology. In the OntoNL Framework the edges linking the classes of the ontology graph are the objectProperties of the OWL syntax and the weight values are specified by the relatedness measure calculation.

In Chapter 6 we described the implementation of the OntoNL Framework and of an application of the OntoNL Framework that addresses a semantic multimedia repository with digital audiovisual content of soccer events and metadata concerning soccer in general. The overall architecture is shown in figure 50. The OntoNL expects domain ontology expressed in OWL. The reference ontologies we used is an application of the DS-MIRF ontological infrastructure (Tsinaraki et alii 2004) and the WordNet for the syntactic analysis. The repository for accessing the instances is the DS-MIRF Metadata Repository (Tsinaraki et alii 2006).

The OntoNL Component provides the NL Ontology API and the NL Query API for communication. The NL Query API contains functions to input a natural language query and after the disambiguation outputs a number of weighted SPARQL queries, based on the structures of the ontologies used for the disambiguation. It implements functions for the data transfer between the Framework and the repository. The NL Ontology API consists of the total of functions used for manipulating the ontologies that interfere with the system.

The DS-MIRF OntoNL Manager provides the OntoNL component with the ontologies for the disambiguation and the natural language expression for disambiguation. It is also responsible for retrieving the user request, communicate with the repository, manage the results, rank them based on any existing User Profile information and presented them to the front end the user uses for interactions.

The output of the OntoNL is weighted SPARQL queries. To interface with DS-MIRF we had to develop mappings of the SPARQL to the retrieval language of DS-MIRF which intern uses XQuery to access semantic MPEG-7 multimedia content from the XML DBMS.

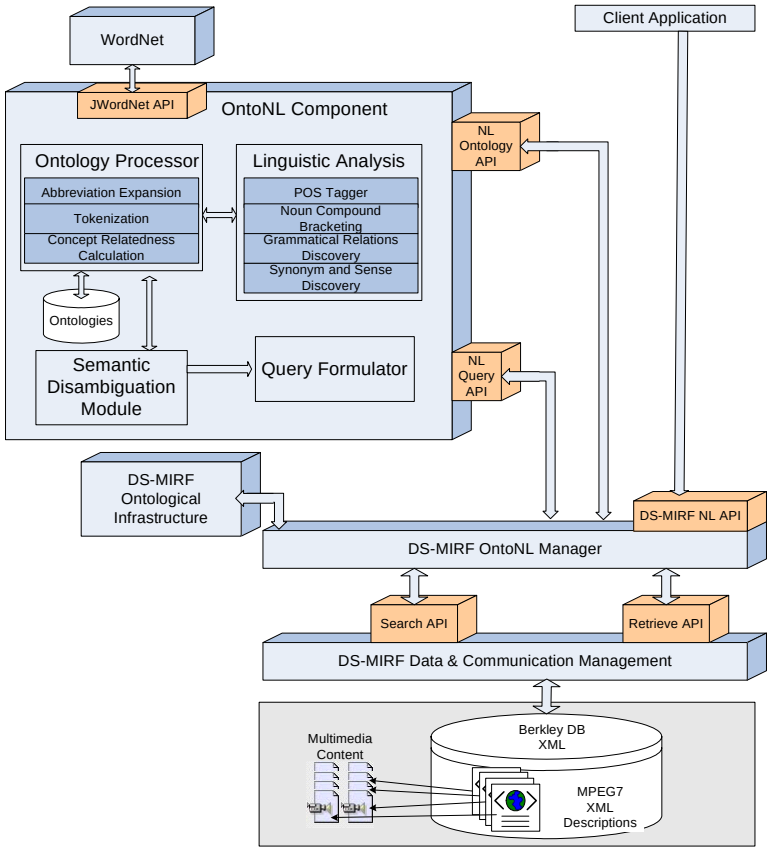


Figure 51. NL2DL Infrastructure

A complete evaluation framework has been designed for the OntoNL generator. As far as it concerns the OntoNL Semantic Relatedness Measure evaluation, the framework takes into account a large number of parameters regarding the characteristics of the ontologies involved and the types of users. We have focused our attention to the performance experimentation in a generic way utilizing readily available ontologies in the web, not carefully constructed by hand ontologies. As we discussed in the previous section the three factors w_1, w_2 and w_3, of the overall OntoNL measure help of balancing among the three sub-measure depending on the application ontology.

In order to assess the impact of each of the sub-measures we needed to evaluate it against a "gold standard" of object relatedness. To that end we designed a detailed experiment in which human subjects were asked to assess the relatedness between two objects. Budanitsky and Hirst (2006) based on their study have found that comparing WordNet similarity measures with human judgments give the best assessments of the "goodness" of a measure. We have obtained relatedness judgments from 25 human subjects, 10 from the computer science field that where shown the domain ontologies' structure and 15 from the liberal arts field that were used for the evaluation, for 85 pairs of concepts that we meet in the seven OWL domain ontologies for different domains (APPENDIX).

The results we obtained were very satisfactory and showed the effectiveness of the OntoNL Framework and especially the OntoNL Semantic Disambiguation Procedure. The results showed that the average OntoNL measure correlation for each ontology was almost always more than 0.9 and in 4 out of the 7 cases they were more than 0.95. The average correlation was 0.94. The human subjects also evaluated the relatedness of the concepts based on the semantic measure that we have developed (common properties, related senses, and conceptual distance). The correlations of their evaluations with the system computed measures were shown in Table 16, and were also satisfactory. From our research we observed that the subjects with computer science background had higher correlations with the system for the conceptual distance measure, while human subjects from liberal arts had higher correlations in general for the related properties measure. In all cases the calculated by the system weighted relatedness measure was higher correlated with the human subject evaluations than the correlations of the partial semantic measures (common properties, related senses, conceptual distance). We wanted to bound their values and provide the complete measurement that would show good results regardless of the OWL ontology used. We first determined the features of the OWL Ontology structure that effect the OntoNL Semantic Relatedness Measure.

An observation was the relatively large variability of the optimal weights for each ontology. We decided to experiment with the same set of weights for all the ontologies, to observe if the relatedness measures were drastically affected, and if they are still satisfactory. The results obtained, as expected, were worse than the results obtained using

different weights for each ontology. The average drop in correlation was 0.024, while the maximum drop in one ontology was 0.06. In this case (Koala Ontology) the average correlation dropped below 0.8 (to 0.798). For this ontology however, even with its optimal weights the correlation was not very high (0.863).

To deal with the uncertainty of the different values of the weights according to the application domain ontology we used a methodology from Linear Programming field and we defined a number of Ontology Metrics so to compute the parameters of the ontologies that define the values of the weights of the OntoNL Semantic Relatedness Measuure. In our case we wanted to minimize the deviation of the results that we have obtained empirically through experimentation with the calculated values of the decision variables extracted from the Linear Solver. The results were satisfactory and showed that with more research in determining metrics of influence of these decision variables we can obtain more accurate results.

After this step we continued with an application-based evaluation of the OntoNL measure. We chose to use for the application, the OWL Ontology for the domain of soccer (http://lamia.ced.tuc.gr/ontologies/AV_MDS03/soccer), because it is a big and very specific ontology. Also, the context of the ontology is familiar with the users.

As far as it concerns the evaluation of the natural language query representation to a query language for retrieval of metadata, the experiments tested if the language model's components where successfully mapped to ontological structures (figure 47) and if the semantic relatedness measure resulted in satisfactory matches (figure 48) for the domain of soccer in a question answering system for the DS-MIRF Metadata Repository. We also presented the overall satisfaction of users with respect to the effectiveness of the results compared against a keyword-based search (figure 49). The conclusion that derived is that in a second iteration of tests the users expressed a higher satisfaction because their familiarity of using the system increased. The results that concerned ontological structures and semantics (figures 47 and 48) were strongly dependent on the form of the specific ontology. Overall, the performance decreases a little as the complexity of the language model increases, but as shown in figure 49, we get the correct results sooner and faster against a keyword-based search.

Chapter 8

The research and development that was conducted in order to complete this thesis has been published in the following conferences and journals:

1. A. Karanastasi, S. Christodoulakis: "The OntoNL Semantic Relatedness Measure for OWL Ontologies", in the Proceedings of the Second IEEE International Conference on Digital Information Management (IEEE ICDIM '07), 28-31 October 2007, Lyon, France
2. A. Karanastasi, S. Christodoulakis: "Semantic Processing of Natural Language Queries in the OntoNL Framework", in the Proceedings of the IEEE International Conference on Semantic Computing (IEEE ICSC), 17-19 September 2007, Irvine, CA
3. A. Karanastasi, S. Christodoulakis: "Ontology-Driven Semantic Ranking for Natural Language Disambiguation in the OntoNL Framework", in the Proceedings of the 4th European Semantic Web Conference (ESWC), 3-7 June 2007, Innsbruck, Austria
4. A. Karanastasi, A. Zwtos, S. Christodoulakis: "The OntoNL Framework for Natural Language Interface Generation and a Domain-specific Application", in Proceedings of the DELOS Conference on Digital Libraries, 13-14 February, Tirrenia, Pisa, Italy 2007
5. A. Karanastasi, A. Zwtos, S. Christodoulakis: "User Interactions with Multimedia Repositories using Natural Language Interfaces: an Architectural Framework and its Implementation", in the Journal of Digital Information Management (JDIM), Volume 4 Issue 4, December 2006
6. A. Karanastasi, A. Zwtos, S. Christodoulakis: "User Interactions with Multimedia Repositories using Natural Language Interfaces: an Architectural Framework and its Implementation", in Proceedings of the Fourth Special Workshop on Multimedia Semantics (WMS06), June 19-21, 2006
7. S. Christodoulakis, A. Karanastasi, J. Koehler, K. Biatov, T. Catarci, S. Kimani: "Natural Language and Speech Interfaces to Knowledge Repositories", Poster on the 9th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2005), September 2005, Vienna, Austria

8. A. Karanastasi, S. Christodoulakis: “OntoNL: An Ontology-based Natural Language Interface Generator for Multimedia Repositories”, in Proceedings of the Seventh International Workshop of the EU Network of Excellence DELOS on AUDIO-VISUAL CONTENT AND INFORMATION VISUALIZATION IN DIGITAL LIBRARIES (AVIVDiLib'05), May 2005
9. A. Karanastasi, F. Kazasis, S. Christodoulakis: “A Natural Language Model and a System for Managing TV-Anytime Information in Mobile Environments”, ACM/Verlag Personal and Ubiquitous Computing Journal, Volume 4, 2005
10. A. Karanastasi, F. Kazasis, S. Christodoulakis: "A Natural Language Model for Managing TV-Anytime Information in Mobile Environments", In Proceedings of the International Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS), Riga, Latvia, 7 - 8 June, 2004
11. A. Karanastasi, F. Kazasis, S. Christodoulakis: "A Natural Language Model and a System for Managing TV-Anytime Information from Mobile Devices", In Proceedings of the 9th International Conference on Applications of Natural Language to Information Systems (NLDB), Manchester, United Kingdom, June 2004
12. F.G. Kazasis, N. Moumoutzis, N. Pappas, A. Karanastasi, S.Christodoulakis: "Designing Ubiquitous Personalized TV-Anytime Services", In the proceedings of the International Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS), lagenfurt/Velden, Austria, June 2003

The OntoNL component has been developed and with applications has been demonstrated in the 2nd and 3rd Annual Review of the DELOS II EU Network of Excellence (IST 507618) (http://www.delos.info/). The developer of the OntoNL component is Alexandros Zotos, graduate student of the Electronic and Computer Engineering Department of the Technical University of Crete and member of the Laboratory of Distributed Multimedia Information Systems and Applications. The application of the OntoNL in the domain of soccer (OntoNL2DS-MIRF) has been evaluated by an evaluation team of three HCI experts, Prof. Tiziana Catarci, Dr. Yael Dubinsky and Dr. Stephen Kimani, from the Department of Computer and Systems Science (Dipartimento di Informatica e Sistemistica) of the University of Rome "La Sapienza".

Chapter 8

Future Directions

Based on a research that was conducted in 2001 by the Global Reach, 55% of the Internet users are non-English speakers. but the 80% of the Internet and Digital Library resources are in English (Bian, Chen, 2000). In order to allow unrestricted access to these data, the availability of language processing tools, i.e. multilingual information retrieval, multilingual display, multilingual text generation, translation memories, terminological databases, lexicon servers and machine translation systems, is a prerequisite.

In the global economy, information systems are no longer utilized by users in a single geographical region but all over the world. Information can be generated, stored, processed, and accessed in several different languages. All of this reveals the importance of research in multilingual information systems.

There are several essential components in multilingual information systems. These components are namely multilingual resources, machine translation, cross-lingual information retrieval, multilingual information extraction and summarization, and user evaluations and studies.

Multilingual resources include corpora, lexicons, and ontology. Parallel and comparable corpora are important for generating a statistical translation model to overcome the limitations of a manually generated dictionary. In addition, annotated corpora and lexicons have been widely used for many natural language processing tasks. Unfortunately, the development of these resources requires much human intervention. Ontology is an inventory of concepts organized insome internal structuring principle, which is important in organizing and managing information.

Machine translation has over 50 years of history. It is defined as an automated process to transform written text from one language to another. One approach is to convert the source text into an *abstract semantic representation*. This semantic representation will be used for producing the translated text in the target language. Another approach will be to mainly base on a *statistical model* for word translations and word re-orderings. The model

parameters can be learned from a large parallel corpus. Recently, research on translating named entities has become popular because it is useful in different information access applications for which named entities play an important role. Automatic generation of transliteration rules is also actively explored. Multilingual information retrieval is defined as the process that takes queries in any language, searches a collection of objects—including text, images, sound clips—and returns the most relevant objects. It involves several major tasks, namely, query translation, indexing, and retrieval methods. One can employ common information retrieval techniques for conducting indexing and retrieval. Query translation is performed in a separate effort. Another approach is to develop a *framework* that can deal with all these tasks in a more integrated manner. This is a vision that can be approached by developing methodologies as the one's described above and integrating them in the OntoNL Framework.

REFERENCES

AGIRRE, E., RIGAU, G. 1997. A proposal for word sense disambiguation using conceptual distance. In Recent Advances in Natural Language Processing: Selected Papers from RANLP'95, volume 136 of Amsterdam Sudies in the Theory and History of Linguistic Science: Current Issues in Linguistic Theory, chapter 2, pages 161-173. John Benjamins Publishing Company, Amsterdam/Phildadelphia, 1997.

AKT ONTOLOGY, http://akt.open.ac.uk/ocml/domains/akt-support-ontology/

ALLEN, J.F. 1994 Natural Language Understanding, Benjamin Cummings, 1987, Second Edition, 1994.

ALSHAWI.1992 The Core Language Engine. MIT Press, Cambridge, Massachusetts.

ALSHAWI,, H., CARTER, D., CROUCH, R., PULMAN, S., RAYNER, M., SMITH, A. CLARE 1992 – A Contextual Reasoning and Cooperative Response Framework for the Core Language Engine. Final report, SRI International, December 1992.

ANDERSON, J. R 1976. Language, Memory, and Thought. Lawrence Erlbaum and Associates, Hillsdale, New Jersey.

ANDERSON, J. R. 1983. A Spreading Activation Theory of Memory. Journal of Verbal Learning and Verbal Behavior,22(3), 261-95.

ASKJEEVES, http://askjeeves.com/, 2000.

BERK, L., 1999. English Syntax. From Word to Discourse, Oxford University Press.

BERGER, A. et al., 2000 Bridging the lexical chasm: Statistical approaches to answer-finding, in Proceedings of SIGIR, 2000.

BERNERS-LEE, T., HENDLER, J., LASSILA, O. 2001. The Semantic Web. Scientific American, 284(5)

BLACKBURN, S. 1994. The Oxford Dictionary of Philosophy, Oxford University Press, Oxford, UK.

BOBROW, et al. 1977. GUS, a frame driven dialog system, Artificial Intelligence, 8:155-173.

BREESE, J. S., HECKERMAN, D., and KADIE, C. 1998. Empirical Analysis of Predictive Algorithms for Collaborative Filtering, in Proc. of the 14th Conf. on Uncertainty in Artificial Intelligence, July, 1998.

BRESNAN, J. 2001. Lexical-Functional Syntax. Blackwell, Oxford.

BRICKLEY, D., GUHA, R. V. 2004 (eds.), "RDF Vocabulary Description Language 1.0: RDF Schema", W3C Recommendation, 10 Feb. 2004. (http://www.w3.org/TR/rdf-schema)

BRILL, E., et al. 2002 Data Intensive Question Answering", Proceedings of TREC-10, NIST.

BROWN, G. YULE, G., Discourse Analysis, Cambridge, 1983.

BURKE, R., et al., “Question Answering from Frequently-Asked Question files: experiences with the FAQ Finder system”, Technical Report, University of Chicago Computer Science Department, June 1997

CALLAN, J., SMEATON, A. 2003. Personalization and recommender systems in digital li-braries. Technical report, DELOS-NSF Workshop on Personalization and Recom-mender Systems in Digital Libraries Further Contributors: Beaulieu M., Borlund P., Brusilovsky P., Chalmers M.,Lynch C.,Riedl J., Smyth B., Straccia, U., Toms E.

CARROLL, J., MINNEN, G., BRISCOE, T. 1999. Corpus annotation for parser evaluation. In Proceedings of the EACL workshop on Linguistically Interpreted Corpora (LINC).

CHARNIAK, E., 2000. A Maximum-entropy-inspired parser. In Proceedings of NAACL-2000.

CHEN, H., LYNCH, K. J. 1992. Automatic construction of networks of concepts characterizing document databases. IEEE Transactions on Systems, Man and Cybernetics, 22(5), 885-902.

CHEN, H., NG, T. 1995. An Algorithmic Approach to Concept Exploration in a Large Knowledge Network (Automatic Thesaurus Consultation); Symbolic Branch-and-Bound vs. Connectionist Hopfield Net Activation. Journal of the American Society for Information Science 46(5):348-369.

CHIA, C. 2002. The personalization challenge in public libraries: perspectives and prospects. Bertelsmann Foundation, Gütersloh 2002.

CHOMSKY, N. 1981. Lectures on Government and Binding. Foris, Dordrecht.

CHOUEKA, Y., LUSIGNAN, S. 1985. Disambiguation by short contexts. Computers and the Humanities, 19, 147-158.

CIMIANO, P., HAASE, P., SURE, Y., VÖLKER, J., AND WANG, Y. 2006. Question answering on top of the BT digital library. In Proceedings of the 15th international Conference on World Wide Web (Edinburgh, Scotland, May 23 - 26, 2006). WWW '06. ACM Press, New York, NY, 861-862.

CRESTANI, F. 1997. Application of Spreading Activation Techniques in Information Retrieval. Artificial Intelligence Review, 11(6): 453-482.

CRYSTAL, D. 1991. A Dictionary of Linguistics and Phonetics, 3rd ed. Blackwell, Oxford, UK

CODD, E.F. 1974. Seven Steps to RENDEZVOUS with the Casual User. In J. Kimbie and K. Koffeman, editors, Data Base Management. North-Holland Publishers, 1974.

CODD, E.F. 1970 A Relational Model for Large Shared Data Banks. Communications of the ACM, 13(6):377–387, 1970.

COHEN, P., and KJELDSEN, R. 1987. Information Retrieval by Constrained Spreading Activation on Semantic Networks. Information Processing and Management, 23(4):255-268.

COLLINS, A., LOFTUS, E. 1975. A spreading activation theory of semantic processing. Psychological Review, 82, 407-428.

COLLINS, M. 1999. Head-Driven Statistical Models for Natural Language Parsing. PhD Thesis, University of Pensilvania.

DAVIDSON, D. 1969. Truth and meaning. In J. W. Davis et al., editors, Philosophical, pages 1--20. Hingham, 1969.

DAVIDSON, D 1973. In defense of Convention T. In H. Leblanc, editor, Truth, Syntax and Modality, pages 76--85. North Holland, 1973

DAVIES, J., WEEKS, R., and KROHN, U. 2002. QuizRDF: SearchTechnology for the Semantic Web. In Proceedings of the WWW2002 workshop on RDF & Semantic Web Applications, Hawaii, USA, 2002.

DELGADO, J., ISHII, N. 1999. Memory-Based Weighted-Majority Prediction for Recommender Systems, ACM SIGIR'99 Workshop on Recommender Systems.

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY. 1989. Proceedings of the Second DARPA Speech and Natural Language Workshop, Cape Cod, Massachusetts, October 1989

DOYLE, P. 1997. Natural language. AI Qual Summary. http://www.cs.dartmouth.edu/%7Ebrd/Teaching/AI/Lectures/Summaries/natlang.html

DYER, M. G. 1983 In-depth Understanding, MIT Press

FALOUTSOS, C. 1996. Searching Multimedia Databases by Content, KluwerAcademic Publishers, 1996

FALLSIDE, D. 2001. "XML Schema Part 0: Primer", W3C Recommendation, (http://www.w3.org/TR/xmlschema-0/)

FROOGLE. http://froogle.google.com

GADAMER, H. G. 1960 Wahrheit und Methode, Tuebingen.

GALE, W.A.; CHURCH, K. W., YAROWSKY, David 1993. "A method for disambiguating word senses in a large corpus." Computers and the Humanities, 26, 415-439.

GENTNER, D., MEDINA, J. 1998. Similarity and the Development of Rules. Cognition 65, 1998: pp. 263-297.

GINZBURG, J. 1995"Resolving questions I", in Linguistics and Philosophy, Vol. 18(5), 459-527, 1995a.

GINZBURG, J. 1995 "Resolving questions II", in Linguistics and Philosophy, Vol. 18(6), 567-609, 1995b.

GOLDMAN, S. A., WARMUTH, M. K., 1993 Learning Binary Relations Using Weighted Majority Voting, ACM COLT 1993, USA, pp453-462.

GOUGENHEIM, G., MICHEA, R. 1961. “Sur la détermination du sens d'un mot au moyen du contexte.” La Traduction Automatique, 2(1), 16-17.

GRAESSER, A.C, FRANKLIN, S. P. 1990. QUEST: a cognitive model of question answering”, Discourse Processes, 13, 279-303.

GREEN, et al., “BASEBALL: an automatic question answerer”, Proceedings of the Western Joint Computer Conference, 1961.

GRICE, H. P. 1967 Logic and Conversation. William James Lectures, Harvard University, 1967 (Reprinted in Grice 1989).

GRICE, H. P. 1975. Logic and conversation. In P. Cole, editor, Speech Acts, Syntax and Semantics, Vol III: Speech Acts. Academic Press, New York, 1975.

GROSZ. B. J. 1983 TEAM: A Transportable Natural-Language Interface System. In Proceedings of the 1st Conference on Applied Natural Language Processing, Santa Monica, California, pages 39–45.

GUARINO, N. 1999. ONTOSEEK: Content-based access to the Web, IEEE Intelligent Systems, pp. 70-80.

GUHA R., McCOOL, R., MILLER, E. 2003. Semantic Search. Proceedings of the WWW2003, Budapest.

HUNTER, J. 2001 Adding Multimedia to the Semantic Web - Building an MPEG-7 Ontology, International Semantic Web Working Symposium (SWWS), Stanford, July 30 - August 1, 2001.

HEARST, M. A. 1991. Noun homograph disambiguation using local context in large corpora. Proceedings of the 7th Annual Conf. of the University of Waterloo Centre for the New OED and Text Research, Oxford, United Kingdom, 1-19.

HENDRIX, G., SACERDOTI, E., SAGALOWICZ, D., SLOCUM, J. 1978 Developing a Natural Language Interface to Complex Data. ACM Transactions on Database Systems, 3(2):105–147, 1978.

HIRST, G., ST-ONGE, D. 1998 Lexical chains as representations of context for the detection and correction of malapropisms. In Christiane Fellbaum, editor, WordNet: An Electronic Lexical Database, chapter 13, pages 305-332. The MIT Press, Cambridge, MA, 1998.

HIZ, H. (ed.), Questions, Reidel, Holland, 1978.

IDE, N. VERONIS, G. 1998. Word Sense Disambiguation: the State of the Art. In Computational Linguistics, 4(1), 1-40, 1998

ISO/IEC: 15938-3:2001: Information Technology – Multimedia content description interface – Part 3 visual (2001) Version 1

JAIN, A.K., DUBES, R.C. 1988. Algorithms for Clustering Data, Prentice-Hall , 1988, ISBN 0-13-022278-X

JAIN, A.K., MURPHY, M. N., FLYNN, P.J. 1999. Data Clustering: A Review, ACM Comp. Surveys, Vol. 31, No. 3, Sept. 99.

JACCARD, P. 1912. The Distribution of the Flora in the Alpine Zone. *The New Phytologist,* 11(2):37-50

JARMASZ, M., SZPAKOWICZ, S. 2003. Roget's Thesaurus and Semantic Similarity. International Conference on Recent Advances in Natural Language Processing (RANLP2003). Borovets, Bulgaria, 2003.

JIANG, J. J., CONRATH, D. W. 1997. Semantic Similarity based on Corpus Statistics and Lexical Taxonomy. In Proceedings of International Conference on Research in Computational Linguistics.

JOHNSON, T.1985. Natural Language Computing: The Commercial Applications. Ovum Ltd., London.

KAZASIS, F.G., MOUMOUTZIS, N., PAPPAS, N., KARANASTASI, A., CHRISTODOULAKIS, S. 2003. Designing Ubiquitous Personalized TV-Anytime Services, In the proceedings of the International Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS), lagenfurt/Velden, Austria, June 2003

KAPLAN, A. 1950. An experimental study of ambiguity and context. Mimeographed, 18pp, November 1950. [Published as: Kaplan, Abraham (1955). "An experimental study of ambiguity and context." Mechanical Translation, 2(2), 39-46.]

KARANASTASI, A., CHRISTODOULAKIS, S. 2007. The OntoNL Semantic Relatedness Measure for OWL Ontologies", in the Proceedings of the Second IEEE International Conference on Digital Information Management (IEEE ICDIM '07), 28-31 October 2007, Lyon, France

KARANASTASI, A., CHRISTODOULAKIS, S. 2007 Semantic Processing of Natural Language Queries in the OntoNL Framework, in the Proceedings of the IEEE International Conference on Semantic Computing (IEEE ICSC), 17-19 September 2007, Irvine, CA

KARANASTASI, A., CHRISTODOULAKIS, S. 2007 Ontology-Driven Semantic Ranking for Natural Language Disambiguation in the OntoNL Framework, in the Proceedings of the 4th European Semantic Web Conference (ESWC), 3-7 June 2007, Innsbruck, Austria

KARANASTASI, A., ZOTOS, A. CHRISTODOULAKIS, S. 2007 The OntoNL Framework for Natural Language Interface Generation and a Domain-specific Application, in Proceedings of the DELOS Conference on Digital Libraries, 13-14 February, Tirrenia, Pisa, Italy 2007

KARANASTASI, A., ZOTOS, A. CHRISTODOULAKIS, S. 2006. User Interactions with Multimedia Repositories using Natural Language Interfaces: an Architectural Framework and its Implementation", in the Journal of Digital Information Management (JDIM), Volume 4 Issue 4, December 2006

KARANASTASI, A., ZOTOS, A. CHRISTODOULAKIS, S. 2006 User Interactions with Multimedia Repositories using Natural Language Interfaces: an Architectural Framework and its Implementation, in Proceedings of the Fourth Special Workshop on Multimedia Semantics (WMS06), June 19-21, 2006

KARANASTASI, A., CHRISTODOULAKIS S., KOEHLER, J., BIATOV, K., CATARCI, T., KIMANI, S. 2005. Natural Language and Speech Interfaces to Knowledge Repositories, Poster on the 9th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2005), September 2005, Vienna, Austria

KARANASTASI, A., CHRISTODOULAKIS, S. 2005 OntoNL: An Ontology-based Natural Language Interface Generator for Multimedia Repositories, in Proceedings of the Seventh International Workshop of the EU Network of Excellence DELOS on AUDIO-VISUAL CONTENT AND INFORMATION VISUALIZATION IN DIGITAL LIBRARIES (AVIVDiLib'05), May 2005

KARANASTASI, A., KAZASIS, F., CHRISTODOULAKIS, S. 2005. A Natural Language Model and a System for Managing TV-Anytime Information in Mobile Environments, Special Issue on ACM Verlag International Journal of Personal and Ubiquitous Computing, Volume9, Number 5, 262-272

KARANASTASI, A., KAZASIS, F., CHRISTODOULAKIS, 2004 A Natural Language Model for Managing TV-Anytime Information in Mobile Environments, In Proceedings of the International Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS), Riga, Latvia, 7 - 8 June, 2004

KARANASTASI, A., KAZASIS, F., CHRISTODOULAKIS, S. 2004. A Natural Language Model for Managing TV-Anytime Information from Mobile Devices. In Proceedings of the 9th International Conference on Applications of Natural Language to Information Systems (NLDB), Manchester, UK, 2004

KELLER, F., LAPATA, M. 2003. Using the Web to obtain frequencies for unseen bigrams. Computational Linguistics, 29:459–484.

KLEIN, D., MANNING C.D. 2003 Accurate Unlexicalized Parsing. In proceedings of the 41st meeting of he association for computational linguistics.

KOUTSOUDAS, A. K. KORFHAGE, R. 1956. “M.T. and the problem of multiple meaning.” Mechanical Translation, 2(2), 46-51.

LAUER, M. 1995. Designing Statistical Language Learners:Experiments on Noun Compounds. Ph.D. thesis, Department of Computing Macquarie University NSW 2109 Australia.

LEACOCK, C., CHODOROW, M. 1998. Combining Local Context and WordNet Similarity for Word Sense Identification. In WordNet: An Electronic Lexical Database. The MIT Press.

LEDOUX, K., GORDON, P. C., CAMBLIN, C. C., & SWAAB, T. Y. 2006. Coreference and lexical repetition: Mechanisms of discourse integration. Memory & Cognition. Cognitive

Neurology/Neuropsychology, Johns Hopkins University, 1629 Thames Street, Suite 350, Baltimore, MD 21231

LEE, J. H., LEE, Y. J., 1993 Information Retrieval based on Conceptual Distance in IS-A Hierarchies, Journal of Documentation 49(2):188-207

LEHNERT, W. G. 1978. The Process of Question Answering, New Jersey.

LEHNERT, W. G. 1986. A conceptual theory of question answering. In B. J. Grosz, K. Sparck Jones, and B. L. Webber, editors, Natural Language Processing, pages 651–657. Kaufmann, Los Altos, CA.

LEVINSON, S. C. 1983. *Pragmatics*. Cambridge University Press, 1983.

LIN, D. 1998. An Information-Theoretic Definition of Similarity. In Proceedings of the 15th International Conference on Machine Learning, Madison, WI

LORD, P., STEVENS, R., BRASS, A., GOBLE, C. 2003 Investigating semantic similarity measures across the Gene Ontology: the relationship between sequence and annotation. Bioinformatics, 19, 1275--1283

LYONS, J., 1995. Linguistic Semantics: An introduction, Cambridge University Press, Cambridge, UK.

MARCUS, M. 1980. A Theory of Syntactic Recognition for Natural Language. MIT Press.

MASTERMAN, M. 1961. Semantic message detection for machine translation, using an interlingua. 1961 International Conference on Machine Translation of Languages and Applied Language Analysis, Her Majesty's Stationery Office, London, 1962, 437-475.

MacCARTNEY, B., De MARNEFFE, M.C., MANNING, C., 2006. Generating Typed Dependency Parses from Phrase Structure Parses, to appear at LREC-06

McCARTHY, J., 1987. Generality in Artificial Intelligence, Communication of the ASM. Vol. 30, No. 12, pp,1030-1035.

McGUINNESS, D. L.,VAN HARMELEN, F. (eds.). 2004 OWL Web Ontology Language: Overview. W3C Recommendation, 10 Feb. 2004. (http://www.w3.org/TR/owl-features).

McKEOWN, K., KUKICH, K., SHAW, J. 1994 Practical issues in automatic documentation generation. In ANLP, pages 7--14.

MANOLA, F., MILLES, E. (eds.), 2004 RDF Primer. W3C Recommendation, 10 Feb. 2004. (http://www.w3.org/TR/rdf-primer)

MELTON, J. 2006 SQL, XQuery, and SPARQL: What's Wrong With This Picture? XTech 2006: "Building Web 2.0" — 16-19 May 2006, Amsterdam, The Netherlands

MEYER, D. E., SCHVANEVELDT, R. W. 1971. Facilitation in recognizing pairs ofwords: Evidence of a dependence between retrieval operations. Journal of Experimental Psychology, 90(2), 227-34.

MILLER, G., BECKWITH, R., FELLBAUM, C., GROSS, D. and MILLER, K.J. 1990. Introduction to WordNet: an on-line lexical database.'International Journal of Lexicography, 3(4), 235 – 244

MILLER, G., BECKWITH, R., FELLBAUM, C., GROSS, D., MILLER, K. 1993. Five papers on wordnet, Technical report, Stanford University

MILLER, G. A. , CHARLES, W. G. 1991. Contextual Correlates of Semantic Similarity. Language and Cognitive Processes 6, 1-28.

MILLER, P. H., 2000. Strong Generative Capacity: The Semantics of Linguistic Formalism. Number 46 in Lecture Notes. CSLI Publications, Stanford, CA.

MINNEN, G., CARROLL, J., PEARCE, D. 2001. Applied morphological processing of English, Natural Language Engineering, 7(3). 207-223

MOLDOVAN, D., et al. 2003. LCC Tools for Question Answering", in Proceedings of TREC-11, NIST.

NAKOV, P., SCHWARTZ, A., WOLF, B., HEARST, M. 2005. Scaling up BioNLP: Application of a text annotation architecture to noun compound bracketing. In Proceedings of SIG BioLINK.

NG, H. T., LEE, H. B. 1996. Integrating multiple knowledge sources to disambiguate word sense: An examplarbased approach. Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics, 24-27 June 1996, University of California, Santa Cruz, California, 40-47.

NIELSEN, J. 1994. Heuristic Evaluation, In Nielsen, J. and Mack, R. L. (Eds.), Usability Inspection Methods. John Wiley and Sons, New York, pp. 25-62.

O'HARA, K., ALANI, H;, SHADBOLT, N. 2002. Identifying Communities of Practices: Analyzing Ontologies as Networks to Support Community Recognition, IFIP-WCC 2002, Montreal, Canada

OLEARY D. 1998. Using AI in knowledge management: Knowledge bases and ontologies. IEEE Intelligent Systems, May 1998.

OWL WEB ONTOLOGY LANGUAGE REFERENCE, http://www.w3.org/TR/owl-ref/

PEARL, J. 1988. Probabilistic Reasoning in Intelligent Systems: Neworkd of Plausible Inference. Morgan Kaufman, San Mateo, CA.

PIERA, C. 1995. On Compounding in English and in Spanish." Evolution and Revolution in Linguistic Theory. Campos, H. and P.Kempchinsky, (eds). Washington: Georgetown University Press.

PRAGER, J., CHU-CARROLL, J., CZUBA, K. 2002. Use of WordNet Hypernyms for Answering What-Is Questions, Proceedings of TREC-10, NIST.

PUSTEJOVSKY, J., ANICK, P., BERGLER, S. 1993. Lexical semantic techniques for corpus analysis. Computational Linguistics, 19(2):331–358.

QUILLIAN, M. R. 1968. Semantic Memory. In Minsky, M. (Ed.), Semantic Information Processing. MIT Press, Cambridge, MA.

QUIRK, R., GREENBAUM, S. 1973. A University Grammar of English. Longman.

RADEV, D., Qi, H., Wu, H., Fan, W. 2002 Evaluating Web-based Question Ansering Systems. Proceedings of LREC, 2002

RATNAPARKHI, A. 1996 A Maximum Entropy Model for Part-of-Speech Tagging. In Proceedings of the 1st Conference on Empirical Methods in Natural Language Processing, EMNLP, Pennsylvania, May, 1996

RESNIK, P. 1989. Access to Multiple Underlying Systems in JANUS. BBN report 7142, Bolt Beranek and Newman Inc., Cambridge, Massachusetts, September 1989.

RESNIK, P., HEARST, M. 1993. Structural Ambiguity and Conceptual Relations. In Proceedings of the Workshop on Very Large Corpora:Academic and Industrial Perspectives, June 22, Ohio State University, pp58-64

RESNIK, P. 1993. Selection and information: a class-based approach to lexical relationships. Ph.D. thesis, University of Pennsylvania, UMI Order No. GAX94-13894.

RESNIK, P. 1995. Using information content to evaluate semantic similarity in a taxonomy. In Proceedings of the l4th International Joint Conference on Artificial Intelligence (1JCAI- 95), pages 448-453.

RDF: Resource Description Framework (RDF): Concepts and Abstract Syntax, Recommendation, World Wide Web Consortium, 2004-02-10; http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

RITCHEY, T. 1998. General Morphological Analysis: A general method for non-quantified modeling

ROCHA, C., SCHWABE, D., POGGI DE ARAGAO, P. 2004. A hybrid approach for searching in the semantic web. In Proceedings of the 13th International World Wide Web Conference, New York, USA, May 2004

SALEMBIER, P. 2001 MPEG-7 Multimedia Description Schemes, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 11, No. 6, 2001

SALTON, G., BUCKLEY, C. 1988. Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management,* 24(5):513-523

SALTON, G. 1989 Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer, Addison-Wesley, 1989

SCHA, R.J.H. 1977 Philips Question Answering System PHILIQA1. In SIGART Newsletter, no.61. ACM, New York, February 1977.

SCHANK, R. C., ABELSON, R. P. 1977. Scripts, Plans, Goals and Understanding, New Jersey.

SHETH, A., BERTRAM, C., AVANT, D., HAMMOND, B., KOCHUT, K., WARKE, Y. 2002. Managing Semantic Content for the Web. IEEE Internet Computing 6(4): 80-87, 2002

SIMMONS, R. F. 1965. Answering English questions by computer: a survey, in Communications of the ACM, 8(1):53-70

SIMMONS, R. F., 1973 Semantic Networks: computation and use for understanding English sentences", in Schank, R. C. and Colby, K. M., Computer Models of Thought and Language, San Francisco.

SLEATOR, D., TEMPERLAY, D. 1993. Parsing English with a link grammar. In Third International Workshop on Parsing Technologies.

SMEATON, A. F., BERRUT, C. 1995. Running TREC-4 experiments: A chronological report of query expansion experiments carried out as part of TREC-4. In Proceedings of The Fourth Text Retrieval Conference (TREC-4). NIST special publication.

SPARCK JONES, K. 1972 Exhaustivity and Specificity, Journal of Documentation, Volume 28 Number 1 1972 pp 11-21.

SPARCK JONES, K. 1983. Compound Noun Interpretation Problems. Computer Speech Processing. In Fallside, F. and Woods, W. A., Prentice-Hall, NJ. pp 363-81.

SPERBER, D., WILSON, D. 1986 Relevance: Communication and Cognition. Blackwell, Oxford and Harvard University Press, Cambridge MA.

STAIRMAND, M. A. 1997. Textual context analysis for information retrieval. In Proceedings of the 20th Annual International A CM- SIGIR Conference on Research and Development in Information Retrieval, pages 140-147.

STEINBACH, M., KARYPIS, G., KUMAR, V. 2000 A Comparison of Document Clustering Techniques, In KDD Workshop on Text Mining.

THE PENN TREEBANK PROJECT, www.cis.upenn.edu/~treebank/

THE SITE OF THE TV-ANYTIME FORUM, http://www.tv-anytime.org

THOMPSON, B. H., THOMPSON, F. B. 1983 Introducing ASK, A Simple Knowledgeable System. In Proceedings of the 1st Conference on Applied Natural Language Processing, Santa Monica, California, pages 17–24, 1983.

TOUTANOVA, K., KLEIN, D., MANNING, C. D., SINGER, Y. 2003 Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network, In Proceedings of Human Language Technology, Edmonton, Canada, 2003.

TOUTANOVA, K., MANNING, C. D. 2000 Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. In Proceedings of EMNLP, Hong-Kong, October, 2000.

TSINARAKI, C., FATOUROU, E., CHRISTODOULAKIS, S. 2003. An Ontology-Driven Framework for the Management of Semantic Metadata describing Audiovisual Information. In Proceedings of CAiSE, Velden, Austria, 2003, pp 340-356

TSINARAKI, C., POLYDOROS, P., CHRISTODOULAKIS, S. 2004. Integration of OWL Ontologies in MPEG-7 and TV-Anytime Compliant Semantic Indexing. In Proceedings of CAiSE 2004: 398-413.

TSINARAKI, C., POLYDOROS, P., CHRISTODOULAKIS, S. 2005. GraphOnto: A Component and a User Interface for the Definition and Use of Ontologies in Multimedia Information Systems, In Proceedings of AVIVDiLib 2005, Cortona, Italy, April 2005

TSINARAKI, C., CHRISTODOULAKIS, S. 2006. A Multimedia User Preference Model that supports Semantics and its application to MPEG 7/21, In Proceedings of MMM 2006, Beijing, China, 4-6 January 2006.

YATES, B., NETO, B. 1999. Modern Information Retrieval. ACM Press, New Cork, USA, 1999.

VARGAS-VERA, M., MOTTA, E. 2004. AQUA – Ontology-based Question Answering System. Third International Mexican Conference on Artificial Intelligence (MICAI-2004), Lecture Notes in Computer Science 2972 Springer Verlag, 2004.

VOORHEES, E. M. 1993. Using wordnet to disambiguate word senses for text retrieval. In Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, pages 171-180.

VOORHEES, E. M. 1994. Query expansion using lexical-semantic relations. In Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, pages 61-69.

WALTZ, D.L. 1978 An English Language Question Answering System for a Large Relational Database. Communications of the ACM, 21(7):526–539, July 1978.

WARREN, D, PEREIRA, F. 1982 An Efficient Easily Adaptable System for Interpreting Natural Language Queries. Computational Linguistics, 8(3-4):110–122, July-December 1982.

WEAVER, W. 1949. Translation. Mimeographed, 12 pp., July 15, 1949. Reprinted in Locke, William N. and Booth, A. Donald (1955) (Eds.), Machine translation of languages. John Wiley & Sons, New York, 15-23.

WEIBE, J., MAPLES, J., DUAN, L., BRUCE, R. 1997. Experience in WordNet sense tagging in the Wall Street Journal. ACL-SIGLEX Workshop “Tagging Text with Lexical Semantics: Why, What, and How?” April 4-5, 1997, Washington, D.C., 8-11.

WEI, M. 1993 An analysis of word relatedness correlation measures. Master's thesis, University of Western Ontario, London, Ontario, May 1993.

WINOGRAD, T. 1972. Understanding Natural Language, NY Academic Press.

WILKS, Y., STEVENSON, M. 1996. The grammar of sense: Is word sense tagging much more than part-of-speech tagging? Technical Report CS-96-05, University of SHEFFIELD, Sheffield, United Kingdom.

WILSON, D., SPERBER, D. 1986 On defining 'relevance', in Grandy and Warner (eds.), Philosophical grounds of rationality, Oxford.

WOODS, W.A. 1968 Procedural Semantics for a Question-Answering Machine. In Proceedings of the Fall Joint Computer Conference, pages 457–471, New York, NY, 1968. AFIPS.

WOODS, W. A., KAPLAN, R. M., WEBBER, B. N. 1972 The Lunar Sciences Natural Language Information System: Final Report. BBN Report 2378, Bolt Beranek and Newman Inc., Cambridge, Massachusetts.

XDM: XQuery 1.0 and XPath 2.0 Data Model (XDM), Candidate Recommendation, World Wide Web Consortium, 2005-11-03; http://www.w3.org/TR/2005/CR-xpath-datamodel-20051103/

APPENDIX

In what follows we present the owl ontologies that were used for the evaluation tests. The figures are taken from the Protégé ontology editor and the GraphOnto ontology editor.

Images Ontology

An ontology for Images, image regions (SVG), videos, frames, segments, and what they depict. Currently the default ontology for images in MINDSWAP's Photostuff tool.

Namespace: http://www.mindswap.org/~glapizco/technical.owl#
Location: http://www.mindswap.org/~glapizco/technical.owl

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY technical.owl "http://www.mindswap.org/~glapizco/technical.owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&technical.owl;"
        xmlns:owl="&owl;"
        xmlns:rdf="&rdf;"
        xmlns:rdfs="&rdfs;">

<!-- Ontology Information -->
  <owl:Ontology rdf:about=""/>

<!-- Classes -->
  <owl:Class rdf:about="#DepictedThing"
           rdfs:label="Depicted Thing">
    <owl:equivalentClass>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://xmlns.com/foaf/0.1/depiction"/>
        <owl:someValuesFrom rdf:resource="&owl;Thing"/>
      </owl:Restriction>
    </owl:equivalentClass>
  </owl:Class>

  <owl:Class rdf:about="#Image"
           rdfs:comment="The class of images"
           rdfs:label="Image">
    <rdfs:subClassOf rdf:resource="#MultimediaContent"/>
    <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1/Image"/>
  </owl:Class>

  <owl:Class rdf:about="#ImagePart"
           rdfs:label="Image Part">
    <rdfs:comment>2D spatial regions of an image or video frame</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Segment"/>
  </owl:Class>

  <owl:Class rdf:about="#ImageText"
           rdfs:label="Image Text">
    <rdfs:comment>Spatial regions of an image or video frame that correspond to text or
```

```
captions</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#StillRegion"/>
  </owl:Class>

  <owl:Class rdf:about="#Mosaic"
             rdfs:label="Mosaic">
    <rdfs:comment>Mosaic or panaoramic view of a video segment</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#StillRegion"/>
  </owl:Class>

  <owl:Class rdf:about="#MovingRegion"
             rdfs:label="Moving Region">
    <rdfs:comment>2D spatio-temporal regions of video data</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Segment"/>
  </owl:Class>

  <owl:Class rdf:about="#Multimedia"
             rdfs:comment="The class of multimedia resources"
             rdfs:label="Multimedia">
    <rdfs:subClassOf rdf:resource="#MultimediaContent"/>
  </owl:Class>

  <owl:Class rdf:about="#MultimediaContent"
             rdfs:comment="The class of multimedia data"
             rdfs:label="Multimedia Content">
    <rdfs:subClassOf rdf:resource="http://www.mindswap.org/~glapizco/simpleABC.owl#Actuality"/>
  </owl:Class>

  <owl:Class rdf:about="#Segment"
             rdfs:label="Segment">
    <rdfs:comment>The class of  fragments of multimedia content</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#MultimediaContent"/>
  </owl:Class>

  <owl:Class rdf:about="#StillRegion"
             rdfs:label="Still Region">
    <rdfs:comment>2D spatial regions of an image or video frame</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Segment"/>
  </owl:Class>

  <owl:Class rdf:about="#Video"
             rdfs:comment="The class of videos"
             rdfs:label="Video">
    <rdfs:subClassOf rdf:resource="#MultimediaContent"/>
  </owl:Class>

  <owl:Class rdf:about="#VideoFrame"
             rdfs:comment="Frame of a video"
             rdfs:label="VideoFrame">
    <rdfs:subClassOf rdf:resource="#MultimediaContent"/>
  </owl:Class>

  <owl:Class rdf:about="#VideoSegment"
             rdfs:label="Video Segment">
    <rdfs:comment>Temporal intervals or segments of video data</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Segment"/>
    <rdfs:subClassOf rdf:resource="#Video"/>
  </owl:Class>

  <owl:Class rdf:about="#VideoSegmentsOrStillRegions"
             rdfs:label="VideoSegmentsOrStillRegions">
    <rdf:type rdf:resource="&owl;Thing"/>
    <owl:unionOf rdf:datatype="&rdf;XMLLiteral">
      &lt;owl:Class rdf:about="#VideoSegment">&lt;/owl:Class>
```

```
    &lt;owl:Class rdf:about="#StillRegion"&gt;&lt;/owl:Class>
   </owl:unionOf>
  </owl:Class>

  <owl:Class rdf:about="#VideoText"
          rdfs:label="Video Text">
   <rdfs:comment>Spatio-temporal regions of video data that correspond to text or captions</rdfs:comment>
   <rdfs:subClassOf rdf:resource="#MovingRegion"/>
  </owl:Class>

  <owl:Class rdf:about="http://www.mindswap.org/~glapizco/simpleABC.owl#Actuality"/>
  <owl:Class rdf:about="http://xmlns.com/foaf/0.1/Image"/>

 <!-- Annotation Properties -->
  <owl:AnnotationProperty rdf:about="&rdfs;comment"/>
  <owl:AnnotationProperty rdf:about="&rdfs;label"/>

 <!-- Datatype Properties -->
  <owl:DatatypeProperty rdf:about="#endFrame"
                  rdfs:label="endFrame">
   <rdfs:domain rdf:resource="#VideoSegment"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#frameNumber"
                  rdfs:label="frameNumber">
   <rdfs:domain rdf:resource="#VideoFrame"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasDurationSeconds"
                  rdfs:label="hasDurationSeconds">
   <rdfs:domain rdf:resource="#Video"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasTotalFrames"
                  rdfs:label="hasTotalFrames">
   <rdfs:domain rdf:resource="#Video"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#startFrame"
                  rdfs:label="startFrame">
   <rdfs:domain rdf:resource="#VideoSegment"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#svgOutline"
                  rdfs:label="svgOutline">
   <rdfs:domain rdf:resource="#ImagePart"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="&owl;unionOf"/>

 <!-- Object Properties -->
  <owl:ObjectProperty rdf:about="#depiction"
                rdfs:label="depiction">
   <rdfs:subPropertyOf rdf:resource="http://xmlns.com/foaf/0.1/depiction"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#depicts"
                  rdfs:label="depicts">
   <owl:inverseOf rdf:resource="#depiction"/>
   <rdfs:subPropertyOf rdf:resource="http://xmlns.com/foaf/0.1/depicts"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#descriptor">
   <rdf:type rdf:resource="&owl;Thing"/>
```

```
  <rdfs:domain rdf:resource="#MultimediaContent"/>
 </owl:ObjectProperty>

 <owl:ObjectProperty rdf:about="#frameOf"
                     rdfs:label="frameOf">
  <rdfs:domain rdf:resource="#VideoFrame"/>
  <rdfs:range rdf:resource="#Video"/>
 </owl:ObjectProperty>

 <owl:ObjectProperty rdf:about="#hasRegion"
                     rdfs:label="hasRegion">
  <rdfs:domain rdf:resource="#MultimediaContent"/>
  <rdfs:range rdf:resource="#ImagePart"/>
 </owl:ObjectProperty>

 <owl:ObjectProperty rdf:about="#hasSegment"
                     rdfs:label="hasSegment">
  <rdfs:domain rdf:resource="#Video"/>
  <rdfs:range rdf:resource="#VideoSegment"/>
 </owl:ObjectProperty>

 <owl:ObjectProperty rdf:about="#regionOf"
                     rdfs:label="regionOf">
  <rdfs:domain rdf:resource="#ImagePart"/>
  <rdfs:range rdf:resource="#MultimediaContent"/>
  <owl:inverseOf rdf:resource="#hasRegion"/>
 </owl:ObjectProperty>

 <owl:ObjectProperty rdf:about="#segmentOf"
                     rdfs:label="segmentOf">
  <rdfs:domain rdf:resource="#VideoSegment"/>
  <rdfs:range rdf:resource="#Video"/>
  <owl:inverseOf rdf:resource="#hasSegment"/>
 </owl:ObjectProperty>

 <owl:ObjectProperty rdf:about="#visualDescriptor">
  <rdf:type rdf:resource="&owl;Thing"/>
  <rdfs:comment>Descriptor - applicable to images, videos, video segments, still regions and moving
regions.</rdfs:comment>
  <owl:subPropertyOf rdf:resource="#descriptor"/>
 </owl:ObjectProperty>

 <owl:ObjectProperty rdf:about="&owl;subPropertyOf"/>
</rdf:RDF>
```

People Ontology

An ontology about people and information about their pets.

Namespace: http://owl.man.ac.uk/2005/07/sssw/people.html
Location: http://owl.man.ac.uk/2005/07/sssw/people.html

```
<?xml version="1.0" ?>
- <rdf:RDF xmlns="file:/Users/seanb/Desktop/Cercedilla2005/hands-on/people.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:ns0="http://owl.man.ac.uk/2005/07/sssw/people#" xml:base="file:/Users/seanb/Desktop/Cercedilla2005/hands-
    on/people.owl">
  <owl:Ontology rdf:about="file:/Users/seanb/Desktop/Cercedilla2005/hands-on/people.owl" />
  <owl:Class rdf:about="http://www.w3.org/2002/07/owl#Thing" />
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#haulage_worker">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
- <owl:equivalentClass>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#works_for" />
    </owl:onProperty>
- <owl:someValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#part_of" />
    </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#haulage_company" />
    </owl:someValuesFrom>
    </owl:Restriction>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#haulage_company" />
    </owl:unionOf>
    </owl:Class>
    </owl:someValuesFrom>
    </owl:Restriction>
    </owl:equivalentClass>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">haulage worker</rdfs:label>
    </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#vehicle">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">vehicle</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
    </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#man">
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#adult" />
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person" />
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#male" />
    </owl:intersectionOf>
    </owl:Class>
    </owl:equivalentClass>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">man</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
    </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#cat_owner">
- <owl:equivalentClass>
```

```
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person" />
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#cat" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#has_pet" />
      </owl:onProperty>
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">cat owner</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#animal">
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#eats" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">animal</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#sheep">
  <rdfs:subClassOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#animal" />
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">sheep</rdfs:label>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:allValuesFrom>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#grass" />
      </owl:allValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#eats" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#woman">
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#adult" />
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#female" />
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person" />
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">woman</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#tabloid">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A newspaper. Tabloids are usually thought of
      as more "down-market" than broadsheets.</rdfs:comment>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">tabloid</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#broadsheet" />
      </owl:disjointWith>
```

```
- <rdfs:subClassOf>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#newspaper" />
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#grownup">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">grownup</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#adult" />
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person" />
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#male">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">male</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The class of all male things.</rdfs:comment>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#lorry_driver">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">lorry driver</rdfs:label>
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person" />
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#lorry" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#drives" />
      </owl:onProperty>
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#cow">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Cows are naturally
        vegetarians.</rdfs:comment>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">cow</rdfs:label>
- <rdfs:subClassOf>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#vegetarian" />
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#quality_broadsheet">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
- <rdfs:subClassOf>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#broadsheet" />
      </rdfs:subClassOf>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">quality broadsheet</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#plant">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">plant</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#animal_lover">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Someone who really likes
        animals</rdfs:comment>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">animal lover</rdfs:label>
- <owl:equivalentClass>
- <owl:Class>
```

<owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person" />
<owl:Restriction>
<owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">3</owl:minCardinality>
<owl:onProperty>
<owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#has_pet" />
</owl:onProperty>
</owl:Restriction>
</owl:intersectionOf>
</owl:Class>
</owl:equivalentClass>
</owl:Class>
<owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#haulage_truck_driver">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<owl:equivalentClass>
<owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person" />
<owl:Restriction>
<owl:someValuesFrom>
<owl:Restriction>
<owl:someValuesFrom>
<owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#haulage_company" />
</owl:someValuesFrom>
<owl:onProperty>
<owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#part_of" />
</owl:onProperty>
</owl:Restriction>
</owl:someValuesFrom>
<owl:onProperty>
<owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#works_for" />
</owl:onProperty>
</owl:Restriction>
<owl:Restriction>
<owl:someValuesFrom>
<owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#truck" />
</owl:someValuesFrom>
<owl:onProperty>
<owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#drives" />
</owl:onProperty>
</owl:Restriction>
</owl:intersectionOf>
</owl:Class>
</owl:equivalentClass>
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">haulage truck driver</rdfs:label>
</owl:Class>
<owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#bus_company">
<rdfs:subClassOf>
<owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#company" />
</rdfs:subClassOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">bus company</rdfs:label>
</owl:Class>
<owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#old_lady">
<owl:equivalentClass>
<owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#female" />
<owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person" />
<owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#elderly" />
</owl:intersectionOf>
</owl:Class>
</owl:equivalentClass>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />

```
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">old lady</rdfs:label>
- <rdfs:subClassOf>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
- <owl:Restriction>
- <owl:allValuesFrom>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#cat" />
      </owl:allValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#has_pet" />
      </owl:onProperty>
      </owl:Restriction>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#has_pet" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#animal" />
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#cat">
  <rdfs:subClassOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#animal" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">cat</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#dog" />
      </owl:disjointWith>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#bicycle">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">bicycle</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A human propelled vehicle, with two
      wheels</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#vehicle" />
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#giraffe">
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:allValuesFrom>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#leaf" />
      </owl:allValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#eats" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:subClassOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#animal" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">giraffe</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#van">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">van</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#vehicle" />
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#lorry">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">lorry</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#vehicle" />
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </owl:Class>
- <owl:Class>
- <owl:disjointWith>
- <owl:Class>
```

```
- <owl:unionOf rdf:parseType="Collection">
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#animal" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#part_of" />
      </owl:onProperty>
      </owl:Restriction>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#animal" />
      </owl:unionOf>
      </owl:Class>
      </owl:disjointWith>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#plant" />
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#part_of" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#plant" />
      </owl:Restriction>
      </owl:unionOf>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#cat_liker">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">cat liker</rdfs:label>
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person" />
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#likes" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#cat" />
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#pet">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">pet</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
- <owl:equivalentClass>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#is_pet_of" />
      </owl:onProperty>
      </owl:Restriction>
      </owl:equivalentClass>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#broadsheet">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A newspaper. Broadsheets are usually
        considered to be more "high-brow" than tabloids.</rdfs:comment>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">broadsheet</rdfs:label>
- <rdfs:subClassOf>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#newspaper" />
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#dog_liker">
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person" />
- <owl:Restriction>
```

```
- <owl:someValuesFrom>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#dog" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#likes" />
      </owl:onProperty>
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">dog liker</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#driver">
- <rdfs:subClassOf>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#adult" />
      </rdfs:subClassOf>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">driver</rdfs:label>
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person" />
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#vehicle" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#drives" />
      </owl:onProperty>
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#newspaper">
- <rdfs:subClassOf>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#tabloid" />
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#broadsheet" />
      </owl:unionOf>
      </owl:Class>
      </rdfs:subClassOf>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">newspaper</rdfs:label>
- <rdfs:subClassOf>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#publication" />
      </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">All newspapers are either broadsheets or
      tabloids.</rdfs:comment>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#red_top">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">red top</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:subClassOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#tabloid" />
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#magazine">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
- <rdfs:subClassOf>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#publication" />
      </rdfs:subClassOf>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">magazine</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#young">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">young</rdfs:label>
- <owl:disjointWith>
```

```
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#adult" />
      </owl:disjointWith>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#bus_driver">
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person" />
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#bus" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#drives" />
      </owl:onProperty>
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">bus driver</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Someone who drives a bus.</rdfs:comment>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#van_driver">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">van driver</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person" />
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#drives" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#van" />
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">person</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#animal" />
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#tiger">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">tiger</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:subClassOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#animal" />
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#kid">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">kid</rdfs:label>
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#young" />
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person" />
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#dog">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">dog</rdfs:label>
```

<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
<owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#bone" />
</owl:someValuesFrom>
- <owl:onProperty>
<owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#eats" />
</owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#duck">
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">duck</rdfs:label>
<rdfs:subClassOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#animal" />
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
</owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#haulage_company">
- <rdfs:subClassOf>
<owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#company" />
</rdfs:subClassOf>
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">haulage company</rdfs:label>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
</owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#grass">
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">grass</rdfs:label>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:subClassOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#plant" />
</owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#brain">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">brain</rdfs:label>
</owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#tree">
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">tree</rdfs:label>
<rdfs:subClassOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#plant" />
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
</owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#bone">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">bone</rdfs:label>
</owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#publication">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">publication</rdfs:label>
</owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#female">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">female</rdfs:label>
</owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#white_thing">
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">white thing</rdfs:label>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
</owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#mad_cow">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A mad cow is a cow that has been eating the brains of sheep.</rdfs:comment>
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">mad cow</rdfs:label>
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#cow" />
- <owl:Restriction>
- <owl:someValuesFrom>

```
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#sheep" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#part_of" />
      </owl:onProperty>
      </owl:Restriction>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#brain" />
      </owl:intersectionOf>
      </owl:Class>
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#eats" />
      </owl:onProperty>
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#truck">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">truck</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#vehicle" />
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#bus">
  <rdfs:subClassOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#vehicle" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">bus</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#pet_owner">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person" />
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#has_pet" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#animal" />
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">pet owner</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#white_van_man">
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#man" />
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#drives" />
      </owl:onProperty>
- <owl:someValuesFrom>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#van" />
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#white_thing" />
      </owl:intersectionOf>
      </owl:Class>
      </owl:someValuesFrom>
```

```
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">white van man</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A white van man is a man who drives a white
      van.</rdfs:comment>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#reads" />
      </owl:onProperty>
  <owl:allValuesFrom rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#tabloid" />
      </owl:Restriction>
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#elderly">
- <rdfs:subClassOf>
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#adult" />
      </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">elderly</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#dog_owner">
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#person" />
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#has_pet" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#dog" />
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">dog owner</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#car">
  <rdfs:subClassOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#vehicle" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">car</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#leaf">
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#part_of" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#tree" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">leaf</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#company">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">company</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#vegetarian">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A vegetarian is defined as an animal that eats
      no other animals, or parts of animals.</rdfs:comment>
```

```
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#animal" />
- <owl:Restriction>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:complementOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#part_of" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#animal" />
      </owl:Restriction>
      </owl:complementOf>
      </owl:Class>
      </owl:allValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#eats" />
      </owl:onProperty>
      </owl:Restriction>
- <owl:Restriction>
- <owl:allValuesFrom>
- <owl:Class>
  <owl:complementOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#animal" />
      </owl:Class>
      </owl:allValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#eats" />
      </owl:onProperty>
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">vegetarian</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#adult">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">adult</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Things that are adult.</rdfs:comment>
      </owl:Class>
- <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#reads">
  <rdfs:range rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#publication" />
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">reads</rdfs:label>
      </owl:ObjectProperty>
- <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#drives">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">drives</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </owl:ObjectProperty>
- <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#has_mother">
  <rdfs:range rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#woman" />
- <rdfs:subPropertyOf>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#has_parent" />
      </rdfs:subPropertyOf>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">has_mother</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </owl:ObjectProperty>
- <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#has_pet">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">has_pet</rdfs:label>
- <rdfs:subPropertyOf>
  <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#likes" />
      </rdfs:subPropertyOf>
  <rdfs:domain rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#person" />
```

<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Anyone that has a pet must like that pet.</rdfs:comment>
<rdfs:range rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#animal" />
- <owl:inverseOf>
<owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#is_pet_of" />
</owl:inverseOf>
</owl:ObjectProperty>
- <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#works_for">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">works_for</rdfs:label>
</owl:ObjectProperty>
- <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#part_of">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">part_of</rdfs:label>
- <owl:inverseOf>
<owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#has_part" />
</owl:inverseOf>
</owl:ObjectProperty>
- <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#has_child">
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">has_child</rdfs:label>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
</owl:ObjectProperty>
- <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#eats">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
- <owl:inverseOf>
<owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#eaten_by" />
</owl:inverseOf>
<rdfs:domain rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#animal" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">eats</rdfs:label>
</owl:ObjectProperty>
- <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#is_pet_of">
<owl:inverseOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#has_pet" />
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">is_pet_of</rdfs:label>
</owl:ObjectProperty>
- <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#likes">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">likes</rdfs:label>
</owl:ObjectProperty>
- <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#has_father">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">has_father</rdfs:label>
<rdfs:range rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#man" />
- <rdfs:subPropertyOf>
<owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#has_parent" />
</rdfs:subPropertyOf>
</owl:ObjectProperty>
- <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#has_parent">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">has_parent</rdfs:label>
</owl:ObjectProperty>
- <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#eaten_by">
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">eaten_by</rdfs:label>
<owl:inverseOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#eats" />
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
</owl:ObjectProperty>
- <owl:ObjectProperty rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#has_part">
<owl:inverseOf rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#part_of" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">has_part</rdfs:label>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
</owl:ObjectProperty>
- <ns0:elderly rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Minnie">
- <ns0:has_pet>
- <owl:Thing rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Tom">

```
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Tom</rdfs:label>
    </owl:Thing>
    </ns0:has_pet>
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Minnie</rdfs:label>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdf:type rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#female" />
    </ns0:elderly>
- <ns0:duck rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Huey">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Huey</rdfs:label>
    </ns0:duck>
- <ns0:van rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Q123_ABC">
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Q123 ABC</rdfs:label>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A white van</rdfs:comment>
<rdf:type rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#white_thing" />
    </ns0:van>
- <ns0:broadsheet rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#The_Times">
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The Times</rdfs:label>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
    </ns0:broadsheet>
- <rdf:Description rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Joe">
- <ns0:has_pet>
- <ns0:dog rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Fido">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Fido</rdfs:label>
    </ns0:dog>
    </ns0:has_pet>
- <rdf:type>
- <owl:Restriction>
<owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:maxCardinality>
<owl:onProperty rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#has_pet" />
    </owl:Restriction>
    </rdf:type>
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Joe</rdfs:label>
<rdf:type rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#person" />
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
    </rdf:Description>
- <ns0:male rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Mick">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Mick is male and drives a white van. Due to
        the axiom concerning drivers, we know that Mick must be a man, and is therefore a white van man. The axiom about the
        reading material of a white van man then allows us to infer things about the Daily Mirror.</rdfs:comment>
<ns0:drives rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#Q123_ABC" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Mick</rdfs:label>
- <ns0:reads>
- <owl:Thing rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Daily_Mirror">
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Daily Mirror</rdfs:label>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The paper read by Mick (a white van
        man).</rdfs:comment>
    </owl:Thing>
    </ns0:reads>
    </ns0:male>
- <ns0:cow rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Flossie">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Flossie</rdfs:label>
    </ns0:cow>
- <ns0:duck rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Dewey">
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Dewey</rdfs:label>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
    </ns0:duck>
- <owl:Thing rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Pete">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Pete</rdfs:label>
    </owl:Thing>
```

```
- <ns0:tiger rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Fluffy">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Fluffy</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </ns0:tiger>
- <ns0:dog rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Rex">
  <ns0:is_pet_of rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#Mick" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Rex</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </ns0:dog>
- <ns0:duck rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Louie">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Louie</rdfs:label>
      </ns0:duck>
- <ns0:tabloid rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#The_Sun">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The Sun</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </ns0:tabloid>
- <owl:Thing rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Spike">
  <ns0:is_pet_of rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#Pete" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Spike</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </owl:Thing>
- <ns0:person rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Walt">
  <ns0:has_pet rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#Huey" />
  <ns0:has_pet rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#Louie" />
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Walt</rdfs:label>
  <ns0:has_pet rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#Dewey" />
      </ns0:person>
- <ns0:person rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Fred">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Fred</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
- <ns0:has_pet>
- <ns0:cat rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Tibbs">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Tibbs</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
      </ns0:cat>
      </ns0:has_pet>
      </ns0:person>
- <owl:AllDifferent>
- <owl:distinctMembers rdf:parseType="Collection">
  <ns0:duck rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Dewey" />
  <ns0:dog rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Fido" />
  <ns0:cow rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Flossie" />
  <ns0:tiger rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Fluffy" />
  <ns0:person rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Fred" />
  <ns0:duck rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Huey" />
  <rdf:Description rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Joe" />
- <ns0:person rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Kevin">
  <ns0:has_pet rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#Fluffy" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Kevin</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <ns0:has_pet rdf:resource="http://owl.man.ac.uk/2005/07/sssw/people#Flossie" />
      </ns0:person>
  <ns0:duck rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Louie" />
  <ns0:male rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Mick" />
  <ns0:elderly rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Minnie" />
  <ns0:van rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Q123_ABC" />
  <ns0:dog rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Rex" />
- <ns0:broadsheet rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#The_Guardian">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The Guardian</rdfs:label>
      </ns0:broadsheet>
  <ns0:tabloid rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#The_Sun" />
```

```
<ns0:broadsheet rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#The_Times" />
<ns0:cat rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Tibbs" />
<ns0:person rdf:about="http://owl.man.ac.uk/2005/07/sssw/people#Walt" />
    </owl:distinctMembers>
    </owl:AllDifferent>
    </rdf:RDF>
```

Koala Ontology

Namespace: http://protege.stanford.edu/plugins/owl/owl-library/koala.owl

Location: http://protege.stanford.edu/plugins/owl/owl-library/koala.owl


```
<?xml version="1.0" ?>
- <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns="http://protege.stanford.edu/plugins/owl/owl-library/koala.owl#"
    xml:base="http://protege.stanford.edu/plugins/owl/owl-library/koala.owl">
  <owl:Ontology rdf:about="" />
- <owl:Class rdf:ID="Female">
- <owl:equivalentClass>
- <owl:Restriction>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasGender" />
    </owl:onProperty>
- <owl:hasValue>
  <Gender rdf:ID="female" />
    </owl:hasValue>
    </owl:Restriction>
    </owl:equivalentClass>
    </owl:Class>
- <owl:Class rdf:ID="Marsupials">
- <owl:disjointWith>
  <owl:Class rdf:about="#Person" />
    </owl:disjointWith>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#Animal" />
    </rdfs:subClassOf>
    </owl:Class>
- <owl:Class rdf:ID="Student">
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Person" />
- <owl:Restriction>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#isHardWorking" />
    </owl:onProperty>
  <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">true</owl:hasValue>
    </owl:Restriction>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#University" />
    </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasHabitat" />
    </owl:onProperty>
    </owl:Restriction>
    </owl:intersectionOf>
    </owl:Class>
    </owl:equivalentClass>
    </owl:Class>
- <owl:Class rdf:ID="KoalaWithPhD">
  <owl:versionInfo>1.2</owl:versionInfo>
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
- <owl:Restriction>
```

```
- <owl:hasValue>
  <Degree rdf:ID="PhD" />
      </owl:hasValue>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasDegree" />
      </owl:onProperty>
      </owl:Restriction>
  <owl:Class rdf:about="#Koala" />
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
      </owl:Class>
- <owl:Class rdf:ID="University">
- <rdfs:subClassOf>
  <owl:Class rdf:ID="Habitat" />
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:ID="Koala">
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">false</owl:hasValue>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#isHardWorking" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#DryEucalyptForest" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasHabitat" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="#Marsupials" />
      </owl:Class>
- <owl:Class rdf:ID="Animal">
  <rdfs:seeAlso>Male</rdfs:seeAlso>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasHabitat" />
      </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:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasGender" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:versionInfo>1.1</owl:versionInfo>
      </owl:Class>
- <owl:Class rdf:ID="Forest">
  <rdfs:subClassOf rdf:resource="#Habitat" />
      </owl:Class>
- <owl:Class rdf:ID="Rainforest">
  <rdfs:subClassOf rdf:resource="#Forest" />
      </owl:Class>
```

<owl:Class rdf:ID="GraduateStudent">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>
<owl:ObjectProperty rdf:about="#hasDegree" />
</owl:onProperty>
<owl:someValuesFrom>
<owl:Class>
<owl:oneOf rdf:parseType="Collection">
<Degree rdf:ID="BA" />
<Degree rdf:ID="BS" />
</owl:oneOf>
</owl:Class>
</owl:someValuesFrom>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#Student" />
</owl:Class>
<owl:Class rdf:ID="Parent">
<owl:equivalentClass>
<owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#Animal" />
<owl:Restriction>
<owl:onProperty>
<owl:ObjectProperty rdf:about="#hasChildren" />
</owl:onProperty>
<owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:minCardinality>
</owl:Restriction>
</owl:intersectionOf>
</owl:Class>
</owl:equivalentClass>
<rdfs:subClassOf rdf:resource="#Animal" />
</owl:Class>
<owl:Class rdf:ID="DryEucalyptForest">
<rdfs:subClassOf rdf:resource="#Forest" />
</owl:Class>
<owl:Class rdf:ID="Quokka">
<rdfs:subClassOf>
<owl:Restriction>
<owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">true</owl:hasValue>
<owl:onProperty>
<owl:FunctionalProperty rdf:about="#isHardWorking" />
</owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#Marsupials" />
</owl:Class>
<owl:Class rdf:ID="TasmanianDevil">
<rdfs:subClassOf rdf:resource="#Marsupials" />
</owl:Class>
<owl:Class rdf:ID="MaleStudentWith3Daughters">
<owl:equivalentClass>
<owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#Student" />
<owl:Restriction>
<owl:onProperty>
<owl:FunctionalProperty rdf:about="#hasGender" />
</owl:onProperty>
<owl:hasValue>
<Gender rdf:ID="male" />
</owl:hasValue>
</owl:Restriction>

```
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasChildren" />
      </owl:onProperty>
  <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">3</owl:cardinality>
      </owl:Restriction>
- <owl:Restriction>
  <owl:allValuesFrom rdf:resource="#Female" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasChildren" />
      </owl:onProperty>
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
      </owl:Class>
  <owl:Class rdf:ID="Degree" />
- <owl:Class rdf:ID="Male">
- <owl:equivalentClass>
- <owl:Restriction>
  <owl:hasValue rdf:resource="#male" />
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasGender" />
      </owl:onProperty>
      </owl:Restriction>
      </owl:equivalentClass>
      </owl:Class>
  <owl:Class rdf:ID="Gender" />
- <owl:Class rdf:ID="Person">
  <rdfs:subClassOf rdf:resource="#Animal" />
  <owl:disjointWith rdf:resource="#Marsupials" />
      </owl:Class>
- <owl:ObjectProperty rdf:ID="hasHabitat">
  <rdfs:range rdf:resource="#Habitat" />
  <rdfs:domain rdf:resource="#Animal" />
      </owl:ObjectProperty>
- <owl:ObjectProperty rdf:ID="hasDegree">
  <rdfs:domain rdf:resource="#Person" />
  <rdfs:range rdf:resource="#Degree" />
      </owl:ObjectProperty>
- <owl:ObjectProperty rdf:ID="hasChildren">
  <rdfs:range rdf:resource="#Animal" />
  <rdfs:domain rdf:resource="#Animal" />
      </owl:ObjectProperty>
- <owl:FunctionalProperty rdf:ID="hasGender">
  <rdfs:range rdf:resource="#Gender" />
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty" />
  <rdfs:domain rdf:resource="#Animal" />
      </owl:FunctionalProperty>
- <owl:FunctionalProperty rdf:ID="isHardWorking">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean" />
  <rdfs:domain rdf:resource="#Person" />
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty" />
      </owl:FunctionalProperty>
  <Degree rdf:ID="MA" />
      </rdf:RDF>
```

Pizza Ontology

Namespace: http://www.co-ode.org/ontologies/pizza/2005/05/16/pizza.owl

Location: http://www.co-ode.org/ontologies/pizza/2005/05/16/pizza.owl


```
<?xml version="1.0" ?>
- <rdf:RDF xmlns="http://www.co-ode.org/ontologies/pizza/2005/05/16/pizza.owl#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
    syntax-ns#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-
    schema#" xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:dc="http://purl.org/dc/elements/1.1/" xml:base="http://www.co-ode.org/ontologies/pizza/2005/05/16/pizza.owl">
- <owl:Ontology rdf:about="">
  <owl:imports rdf:resource="http://protege.stanford.edu/plugins/owl/protege" />
  <owl:versionInfo rdf:datatype="http://www.w3.org/2001/XMLSchema#string">version 1.2</owl:versionInfo>
  <rdfs:comment xml:lang="en">A "final stage" that contains all constructs required for the various versions of the Pizza
    Tutorial run by Manchester</rdfs:comment>
  <protege:defaultLanguage rdf:datatype="http://www.w3.org/2001/XMLSchema#string">en</protege:defaultLanguage>
    </owl:Ontology>
- <owl:Class rdf:ID="TomatoTopping">
- <owl:disjointWith>
  <owl:Class rdf:ID="SpinachTopping" />
    </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="RocketTopping" />
    </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="OliveTopping" />
    </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="CaperTopping" />
    </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="PetitPoisTopping" />
    </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:ID="Mild" />
    </owl:someValuesFrom>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:ID="hasSpiciness" />
    </owl:onProperty>
    </owl:Restriction>
    </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:ID="ArtichokeTopping" />
    </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="LeekTopping" />
    </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="PepperTopping" />
    </owl:disjointWith>
  <rdfs:label xml:lang="pt">CoberturaDeTomate</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:ID="OnionTopping" />
    </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="GarlicTopping" />
    </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="AsparagusTopping" />
```

```
        </owl:disjointWith>
- <rdfs:subClassOf>
  <owl:Class rdf:ID="VegetableTopping" />
        </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:ID="MushroomTopping" />
        </owl:disjointWith>
        </owl:Class>
- <owl:Class rdf:ID="Rosa">
- <owl:disjointWith>
  <owl:Class rdf:ID="Napoletana" />
        </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="LaReine" />
        </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="Cajun" />
        </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:ID="hasTopping" />
        </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
        </owl:Restriction>
        </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:ID="Capricciosa" />
        </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="AmericanHot" />
        </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="FruttiDiMare" />
        </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="QuattroFormaggi" />
        </owl:disjointWith>
  <rdfs:label xml:lang="pt">Rosa</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:ID="Siciliana" />
        </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="FourSeasons" />
        </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
        </owl:onProperty>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:ID="GorgonzolaTopping" />
  <owl:Class rdf:ID="MozzarellaTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
        </owl:unionOf>
        </owl:Class>
        </owl:allValuesFrom>
        </owl:Restriction>
        </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:ID="Giardiniera" />
        </owl:disjointWith>
```

```
- <owl:disjointWith>
  <owl:Class rdf:ID="Margherita" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#GorgonzolaTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:ID="Fiorentina" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="UnclosedPizza" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="Veneziana" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="Soho" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="Caprina" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="SloppyGiuseppe" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="PolloAdAstra" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="Mushroom" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:ID="American" />
      </owl:disjointWith>
- <rdfs:subClassOf>
  <owl:Class rdf:ID="NamedPizza" />
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:ID="PrinceCarlo" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="Parmense" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:ID="SlicedTomatoTopping">
  <rdfs:label xml:lang="pt">CoberturaDeTomateFatiado</rdfs:label>
  <rdfs:subClassOf rdf:resource="#TomatoTopping" />
- <rdfs:subClassOf>
```

```
- <owl:Restriction>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Mild" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:ID="SundriedTomatoTopping" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#FruttiDiMare">
- <owl:disjointWith>
  <owl:Class rdf:about="#Siciliana" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Napoletana" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#FourSeasons" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#GarlicTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#QuattroFormaggi" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#GarlicTopping" />
  <owl:Class rdf:ID="MixedSeafoodTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#American" />
      </owl:disjointWith>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#NamedPizza" />
      </rdfs:subClassOf>
```

```
- <owl:disjointWith>
  <owl:Class rdf:about="#Capricciosa" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MixedSeafoodTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Veneziana" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Mushroom" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PolloAdAstra" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Margherita" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#UnclosedPizza" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">FrutosDoMar</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#SloppyGiuseppe" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Giardiniera" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#LaReine" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Parmense" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Rosa" />
- <owl:disjointWith>
  <owl:Class rdf:about="#PrinceCarlo" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Cajun" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:ID="PizzaTopping">
```

```
- <rdfs:subClassOf>
  <owl:Class rdf:ID="DomainConcept" />
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:ID="IceCream" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="PizzaBase" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">CoberturaDaPizza</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:ID="Pizza" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#American">
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#QuattroFormaggi" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#UnclosedPizza" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">Americana</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#Capricciosa" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#LaReine" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Parmense" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PrinceCarlo" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#SloppyGiuseppe" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Giardiniera" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Mushroom" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
  <owl:disjointWith rdf:resource="#Rosa" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Napoletana" />
      </owl:disjointWith>
```

```
- <owl:disjointWith>
  <owl:Class rdf:about="#PolloAdAstra" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Siciliana" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
      </owl:disjointWith>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#NamedPizza" />
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Cajun" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:ID="PeperoniSausageTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#PeperoniSausageTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Margherita" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#FourSeasons" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Veneziana" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#SpinachTopping">
  <owl:disjointWith rdf:resource="#TomatoTopping" />
```

```
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Mild" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#PetitPoisTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#RocketTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#ArtichokeTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#GarlicTopping" />
      </owl:disjointWith>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#VegetableTopping" />
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#LeekTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#MushroomTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#AsparagusTopping" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">CoberturaDeEspinafre</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#CaperTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#OliveTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#OnionTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PepperTopping" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#Pizza">
  <owl:disjointWith rdf:resource="#PizzaTopping" />
  <rdfs:label xml:lang="en">Pizza</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#IceCream" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:InverseFunctionalProperty rdf:ID="hasBase" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#PizzaBase" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
```

```
- <owl:disjointWith>
  <owl:Class rdf:about="#PizzaBase" />
      </owl:disjointWith>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#DomainConcept" />
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:about="#MushroomTopping">
- <owl:disjointWith>
  <owl:Class rdf:about="#LeekTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#SpinachTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#OnionTopping" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Mild" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#PepperTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#TomatoTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#AsparagusTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#RocketTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#GarlicTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#ArtichokeTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#OliveTopping" />
      </owl:disjointWith>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#VegetableTopping" />
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#CaperTopping" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">CoberturaDeCogumelo</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#PetitPoisTopping" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:ID="HotSpicedBeefTopping">
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:ID="Hot" />
      </owl:someValuesFrom>
```

```
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:ID="ChickenTopping" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">CoberturaDeBifePicante</rdfs:label>
- <rdfs:subClassOf>
  <owl:Class rdf:ID="MeatTopping" />
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:ID="HamTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PeperoniSausageTopping" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#PizzaBase">
- <owl:disjointWith>
  <owl:Class rdf:about="#IceCream" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Pizza" />
  <owl:disjointWith rdf:resource="#PizzaTopping" />
  <rdfs:label xml:lang="pt">BaseDaPizza</rdfs:label>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#DomainConcept" />
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:ID="ValuePartition">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A ValuePartition is a pattern that describes a
      restricted set of classes from which a property can be associated. The parent class is used in restrictions, and the
      covering axiom means that only members of the subclasses may be used as values. The possible subclasses cannot be
      extended without updating the ValuePartition class.</rdfs:comment>
  <rdfs:label xml:lang="pt">ValorDaParticao</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#DomainConcept" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#GarlicTopping">
  <rdfs:label xml:lang="pt">CoberturaDeAlho</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#CaperTopping" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:ID="Medium" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#LeekTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#MushroomTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#OnionTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PetitPoisTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#SpinachTopping" />
  <owl:disjointWith rdf:resource="#TomatoTopping" />
```

```
- <owl:disjointWith>
  <owl:Class rdf:about="#ArtichokeTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#AsparagusTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PepperTopping" />
      </owl:disjointWith>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#VegetableTopping" />
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#RocketTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#OliveTopping" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#Parmense">
- <owl:disjointWith>
  <owl:Class rdf:about="#Giardiniera" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#LaReine" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#American" />
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Margherita" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:ID="ParmesanTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Veneziana" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#UnclosedPizza" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#FourSeasons" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PolloAdAstra" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#QuattroFormaggi" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Mushroom" />
```

```
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Siciliana" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">Parmense</rdfs:label>
  <owl:disjointWith rdf:resource="#Rosa" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#AsparagusTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Capricciosa" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#NamedPizza" />
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#TomatoTopping" />
  <owl:Class rdf:about="#AsparagusTopping" />
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#ParmesanTopping" />
  <owl:Class rdf:about="#HamTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
```

```
- <owl:disjointWith>
  <owl:Class rdf:about="#PrinceCarlo" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Napoletana" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Cajun" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#SloppyGiuseppe" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#HamTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:about="#Cajun">
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Siciliana" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Napoletana" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PolloAdAstra" />
      </owl:disjointWith>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#NamedPizza" />
      </rdfs:subClassOf>
  <rdfs:label xml:lang="pt">Cajun</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#QuattroFormaggi" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:ID="PrawnsTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#American" />
- <owl:disjointWith>
  <owl:Class rdf:about="#SloppyGiuseppe" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Rosa" />
- <owl:disjointWith>
  <owl:Class rdf:about="#UnclosedPizza" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PrinceCarlo" />
      </owl:disjointWith>
```

```
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:ID="TobascoPepperSauce" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Mushroom" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#LaReine" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Capricciosa" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Giardiniera" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#OnionTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Veneziana" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#OnionTopping" />
  <owl:Class rdf:ID="PeperonataTopping" />
  <owl:Class rdf:about="#PrawnsTopping" />
  <owl:Class rdf:about="#TobascoPepperSauce" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
```

```
    </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
    </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
    </owl:disjointWith>
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
- <owl:disjointWith>
  <owl:Class rdf:about="#FourSeasons" />
    </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Margherita" />
    </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#PeperonataTopping" />
    </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
    </owl:onProperty>
    </owl:Restriction>
    </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
    </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
    </owl:someValuesFrom>
    </owl:Restriction>
    </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Parmense" />
    </owl:Class>
- <owl:Class rdf:ID="GoatsCheeseTopping">
- <rdfs:subClassOf>
  <owl:Class rdf:ID="CheeseTopping" />
    </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
    </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Mild" />
    </owl:someValuesFrom>
    </owl:Restriction>
    </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#MozzarellaTopping" />
    </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#GorgonzolaTopping" />
    </owl:disjointWith>
  <rdfs:label xml:lang="pt">CoberturaDeQueijoDeCabra</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#ParmesanTopping" />
    </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="FourCheesesTopping" />
    </owl:disjointWith>
    </owl:Class>
```

```
- <owl:Class rdf:about="#Napoletana">
  <owl:disjointWith rdf:resource="#Cajun" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Mushroom" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PolloAdAstra" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">Napoletana</rdfs:label>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:ID="AnchoviesTopping" />
  <owl:Class rdf:about="#CaperTopping" />
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#OliveTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#LaReine" />
      </owl:disjointWith>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#NamedPizza" />
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#UnclosedPizza" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#FourSeasons" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PrinceCarlo" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Rosa" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#OliveTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
```

```
- <owl:disjointWith>
  <owl:Class rdf:about="#Margherita" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Parmense" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#AnchoviesTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#SloppyGiuseppe" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Capricciosa" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#CaperTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Veneziana" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Siciliana" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Giardiniera" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#American" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#QuattroFormaggi" />
      </owl:disjointWith>
      </owl:Class>
```

```
- <owl:Class rdf:about="#VegetableTopping">
- <owl:disjointWith>
  <owl:Class rdf:ID="FishTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="SauceTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#MeatTopping" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">CoberturaDeVegetais</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:ID="HerbSpiceTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:ID="FruitTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#CheeseTopping" />
      </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#PizzaTopping" />
- <owl:disjointWith>
  <owl:Class rdf:ID="NutTopping" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:ID="JalapenoPepperTopping">
- <owl:disjointWith>
  <owl:Class rdf:ID="SweetPepperTopping" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Hot" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:ID="GreenPepperTopping" />
      </owl:disjointWith>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#PepperTopping" />
      </rdfs:subClassOf>
  <rdfs:label xml:lang="pt">CoberturaDeJalapeno</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#PeperonataTopping" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#FishTopping">
  <rdfs:subClassOf rdf:resource="#PizzaTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#MeatTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#VegetableTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#HerbSpiceTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#FruitTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#NutTopping" />
```

```
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#CheeseTopping" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">CoberturaDePeixe</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#SauceTopping" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Mild" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:about="#AnchoviesTopping">
  <rdfs:label xml:lang="pt">CoberturaDeAnchovies</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#PrawnsTopping" />
      </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#FishTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#MixedSeafoodTopping" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:ID="VegetarianPizzaEquivalent1">
  <rdfs:label xml:lang="pt">PizzaVegetarianaEquivalente1</rdfs:label>
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Pizza" />
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:allValuesFrom>
  <owl:Class rdf:ID="VegetarianTopping" />
      </owl:allValuesFrom>
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:comment xml:lang="en">Any pizza that only has vegetarian toppings or no toppings is a VegetarianPizzaEquiv1.
      Should be inferred to be equivalent to VegetarianPizzaEquiv2. Not equivalent to VegetarianPizza because PizzaTopping
      is not covering</rdfs:comment>
      </owl:Class>
- <owl:Class rdf:about="#AsparagusTopping">
- <owl:disjointWith>
  <owl:Class rdf:about="#ArtichokeTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#LeekTopping" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Mild" />
```

```
        </owl:someValuesFrom>
        </owl:Restriction>
        </rdfs:subClassOf>
    <rdfs:label xml:lang="pt">CoberturaDeAspargos</rdfs:label>
  - <owl:disjointWith>
    <owl:Class rdf:about="#RocketTopping" />
        </owl:disjointWith>
  - <owl:disjointWith>
    <owl:Class rdf:about="#OnionTopping" />
        </owl:disjointWith>
    <rdfs:subClassOf rdf:resource="#VegetableTopping" />
    <owl:disjointWith rdf:resource="#SpinachTopping" />
  - <owl:disjointWith>
    <owl:Class rdf:about="#PepperTopping" />
        </owl:disjointWith>
  - <owl:disjointWith>
    <owl:Class rdf:about="#CaperTopping" />
        </owl:disjointWith>
  - <owl:disjointWith>
    <owl:Class rdf:about="#PetitPoisTopping" />
        </owl:disjointWith>
    <owl:disjointWith rdf:resource="#MushroomTopping" />
    <owl:disjointWith rdf:resource="#TomatoTopping" />
  - <owl:disjointWith>
    <owl:Class rdf:about="#OliveTopping" />
        </owl:disjointWith>
    <owl:disjointWith rdf:resource="#GarlicTopping" />
        </owl:Class>
  - <owl:Class rdf:ID="SultanaTopping">
  - <rdfs:subClassOf>
  - <owl:Restriction>
  - <owl:onProperty>
    <owl:FunctionalProperty rdf:about="#hasSpiciness" />
        </owl:onProperty>
  - <owl:someValuesFrom>
    <owl:Class rdf:about="#Medium" />
        </owl:someValuesFrom>
        </owl:Restriction>
        </rdfs:subClassOf>
    <rdfs:label xml:lang="pt">CoberturaSultana</rdfs:label>
  - <rdfs:subClassOf>
    <owl:Class rdf:about="#FruitTopping" />
        </rdfs:subClassOf>
        </owl:Class>
  - <owl:Class rdf:about="#GorgonzolaTopping">
  - <owl:disjointWith>
    <owl:Class rdf:about="#ParmesanTopping" />
        </owl:disjointWith>
  - <rdfs:subClassOf>
    <owl:Class rdf:about="#CheeseTopping" />
        </rdfs:subClassOf>
  - <owl:disjointWith>
    <owl:Class rdf:about="#MozzarellaTopping" />
        </owl:disjointWith>
  - <rdfs:subClassOf>
  - <owl:Restriction>
  - <owl:onProperty>
    <owl:FunctionalProperty rdf:about="#hasSpiciness" />
        </owl:onProperty>
  - <owl:someValuesFrom>
    <owl:Class rdf:about="#Mild" />
        </owl:someValuesFrom>
        </owl:Restriction>
        </rdfs:subClassOf>
```

```
- <owl:disjointWith>
  <owl:Class rdf:about="#FourCheesesTopping" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">CoberturaDeGorgonzola</rdfs:label>
  <owl:disjointWith rdf:resource="#GoatsCheeseTopping" />
      </owl:Class>
- <owl:Class rdf:about="#NutTopping">
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Mild" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <rdfs:label xml:lang="pt">CoberturaDeCastanha</rdfs:label>
  <owl:disjointWith rdf:resource="#VegetableTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#MeatTopping" />
      </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#PizzaTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#FruitTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#SauceTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#HerbSpiceTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#FishTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#CheeseTopping" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:ID="HotGreenPepperTopping">
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Hot" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <rdfs:label xml:lang="pt">CoberturaDePimentaoVerdePicante</rdfs:label>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#GreenPepperTopping" />
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:about="#ChickenTopping">
- <owl:disjointWith>
  <owl:Class rdf:about="#PeperoniSausageTopping" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Mild" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
```

```
        </owl:onProperty>
        </owl:Restriction>
        </rdfs:subClassOf>
    <rdfs:label xml:lang="pt">CoberturaDeFrango</rdfs:label>
    <owl:disjointWith rdf:resource="#HotSpicedBeefTopping" />
  - <rdfs:subClassOf>
    <owl:Class rdf:about="#MeatTopping" />
        </rdfs:subClassOf>
  - <owl:disjointWith>
    <owl:Class rdf:about="#HamTopping" />
        </owl:disjointWith>
        </owl:Class>
  - <owl:Class rdf:ID="Spiciness">
    <rdfs:label xml:lang="pt">Tempero</rdfs:label>
    <rdfs:comment xml:lang="en">A ValuePartition that describes only values from Hot, Medium or Mild. NB Subclasses can
          themselves be divided up into further partitions.</rdfs:comment>
  - <owl:equivalentClass>
  - <owl:Class>
  - <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Hot" />
    <owl:Class rdf:about="#Medium" />
    <owl:Class rdf:about="#Mild" />
        </owl:unionOf>
        </owl:Class>
        </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#ValuePartition" />
        </owl:Class>
  - <owl:Class rdf:about="#PrawnsTopping">
  - <owl:disjointWith>
    <owl:Class rdf:about="#MixedSeafoodTopping" />
        </owl:disjointWith>
    <rdfs:label xml:lang="pt">CoberturaDeCamarao</rdfs:label>
    <rdfs:subClassOf rdf:resource="#FishTopping" />
    <owl:disjointWith rdf:resource="#AnchoviesTopping" />
        </owl:Class>
  - <owl:Class rdf:ID="CheeseyPizza">
  - <owl:equivalentClass>
  - <owl:Class>
  - <owl:intersectionOf rdf:parseType="Collection">
  - <owl:Restriction>
  - <owl:onProperty>
    <owl:ObjectProperty rdf:about="#hasTopping" />
        </owl:onProperty>
  - <owl:someValuesFrom>
    <owl:Class rdf:about="#CheeseTopping" />
        </owl:someValuesFrom>
        </owl:Restriction>
    <owl:Class rdf:about="#Pizza" />
        </owl:intersectionOf>
        </owl:Class>
        </owl:equivalentClass>
    <rdfs:label xml:lang="pt">PizzaComQueijo</rdfs:label>
    <rdfs:comment xml:lang="en">Any pizza that has at least 1 cheese topping.</rdfs:comment>
        </owl:Class>
  - <owl:Class rdf:ID="SpicyTopping">
    <rdfs:label xml:lang="pt">CoberturaTemperada</rdfs:label>
    <rdfs:comment xml:lang="en">Any pizza topping that has spiciness Hot</rdfs:comment>
  - <owl:equivalentClass>
  - <owl:Class>
  - <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#PizzaTopping" />
  - <owl:Restriction>
  - <owl:onProperty>
    <owl:FunctionalProperty rdf:about="#hasSpiciness" />
```

```
        </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Hot" />
        </owl:someValuesFrom>
        </owl:Restriction>
        </owl:intersectionOf>
        </owl:Class>
        </owl:equivalentClass>
        </owl:Class>
- <owl:Class rdf:about="#ArtichokeTopping">
- <owl:disjointWith>
  <owl:Class rdf:about="#OliveTopping" />
        </owl:disjointWith>
  <owl:disjointWith rdf:resource="#GarlicTopping" />
  <owl:disjointWith rdf:resource="#TomatoTopping" />
  <owl:disjointWith rdf:resource="#AsparagusTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#CaperTopping" />
        </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#RocketTopping" />
        </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#VegetableTopping" />
  <owl:disjointWith rdf:resource="#MushroomTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#PepperTopping" />
        </owl:disjointWith>
  <rdfs:label xml:lang="pt">CoberturaDeArtichoke</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#OnionTopping" />
        </owl:disjointWith>
  <owl:disjointWith rdf:resource="#SpinachTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#LeekTopping" />
        </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Mild" />
        </owl:someValuesFrom>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
        </owl:onProperty>
        </owl:Restriction>
        </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#PetitPoisTopping" />
        </owl:disjointWith>
        </owl:Class>
- <owl:Class rdf:ID="CajunSpiceTopping">
- <owl:disjointWith>
  <owl:Class rdf:ID="RosemaryTopping" />
        </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Hot" />
        </owl:someValuesFrom>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
        </owl:onProperty>
        </owl:Restriction>
        </rdfs:subClassOf>
- <rdfs:subClassOf>
```

```
  <owl:Class rdf:about="#HerbSpiceTopping" />
      </rdfs:subClassOf>
  <rdfs:label xml:lang="pt">CoberturaDeCajun</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="#FruitTopping">
- <owl:disjointWith>
  <owl:Class rdf:about="#CheeseTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#HerbSpiceTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#SauceTopping" />
      </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#PizzaTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#MeatTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#VegetableTopping" />
  <owl:disjointWith rdf:resource="#FishTopping" />
  <owl:disjointWith rdf:resource="#NutTopping" />
  <rdfs:label xml:lang="pt">CoberturaDeFrutas</rdfs:label>
      </owl:Class>
- <owl:Class rdf:ID="ParmaHamTopping">
  <rdfs:label xml:lang="pt">CoberturaDePrezuntoParma</rdfs:label>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#HamTopping" />
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Mild" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:ID="RedOnionTopping">
  <rdfs:label xml:lang="pt">CoberturaDeCebolaVermelha</rdfs:label>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#OnionTopping" />
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:ID="NonVegetarianPizza">
- <owl:disjointWith>
  <owl:Class rdf:ID="VegetarianPizza" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">PizzaNaoVegetariana</rdfs:label>
  <rdfs:comment xml:lang="en">Any Pizza that is not a VegetarianPizza</rdfs:comment>
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
- <owl:Class>
- <owl:complementOf>
  <owl:Class rdf:about="#VegetarianPizza" />
      </owl:complementOf>
      </owl:Class>
  <owl:Class rdf:about="#Pizza" />
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
      </owl:Class>
```

```
- <owl:Class rdf:about="#RosemaryTopping">
  <rdfs:label xml:lang="pt">CoberturaRosemary</rdfs:label>
  <owl:disjointWith rdf:resource="#CajunSpiceTopping" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Mild" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#HerbSpiceTopping" />
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:about="#NamedPizza">
  <rdfs:subClassOf rdf:resource="#Pizza" />
  <rdfs:comment xml:lang="en">A pizza that can be found on a pizza menu</rdfs:comment>
  <rdfs:label xml:lang="pt">PizzaComUmNome</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="#Capricciosa">
- <owl:disjointWith>
  <owl:Class rdf:about="#FourSeasons" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#QuattroFormaggi" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#TomatoTopping" />
  <owl:Class rdf:about="#HamTopping" />
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#AnchoviesTopping" />
  <owl:Class rdf:about="#OliveTopping" />
  <owl:Class rdf:about="#PeperonataTopping" />
  <owl:Class rdf:about="#CaperTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Parmense" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Mushroom" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#AnchoviesTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
```

```
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#OliveTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#HamTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#PeperonataTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Napoletana" />
- <owl:disjointWith>
  <owl:Class rdf:about="#PrinceCarlo" />
      </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Giardiniera" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#CaperTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#SloppyGiuseppe" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#UnclosedPizza" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
```

```
- <owl:disjointWith>
  <owl:Class rdf:about="#LaReine" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
- <owl:disjointWith>
  <owl:Class rdf:about="#PolloAdAstra" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Rosa" />
  <owl:disjointWith rdf:resource="#American" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Veneziana" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Cajun" />
  <rdfs:label xml:lang="pt">Capricciosa</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#Siciliana" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Margherita" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#Siciliana">
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#ArtichokeTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Capricciosa" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
```

```
  <owl:disjointWith rdf:resource="#Cajun" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#UnclosedPizza" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#AnchoviesTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Napoletana" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Margherita" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
- <owl:disjointWith>
  <owl:Class rdf:about="#FourSeasons" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Mushroom" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#American" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#OliveTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PrinceCarlo" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#HamTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Rosa" />
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
- <rdfs:subClassOf>
```

```
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#AnchoviesTopping" />
  <owl:Class rdf:about="#ArtichokeTopping" />
  <owl:Class rdf:about="#GarlicTopping" />
  <owl:Class rdf:about="#HamTopping" />
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#OliveTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Veneziana" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#LaReine" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Giardiniera" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Parmense" />
- <owl:disjointWith>
  <owl:Class rdf:about="#PolloAdAstra" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#SloppyGiuseppe" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">Siciliana</rdfs:label>
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#GarlicTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#QuattroFormaggi" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#Hot">
- <owl:disjointWith>
  <owl:Class rdf:about="#Mild" />
      </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#Spiciness" />
  <rdfs:label xml:lang="pt">Picante</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="#GreenPepperTopping">
- <owl:disjointWith>
  <owl:Class rdf:about="#PeperonataTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#JalapenoPepperTopping" />
  <rdfs:label xml:lang="pt">CoberturaDePimentaoVerde</rdfs:label>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#PepperTopping" />
      </rdfs:subClassOf>
```

```
- <owl:disjointWith>
  <owl:Class rdf:about="#SweetPepperTopping" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:ID="ThinAndCrispyBase">
  <rdfs:subClassOf rdf:resource="#PizzaBase" />
- <owl:disjointWith>
  <owl:Class rdf:ID="DeepPanBase" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">BaseFinaEQuebradica</rdfs:label>
      </owl:Class>
- <owl:Class rdf:ID="VegetarianPizzaEquivalent2">
  <rdfs:comment xml:lang="en">An alternative to VegetarianPizzaEquiv1 that does not require a definition of
        VegetarianTopping. Perhaps more difficult to maintain. Not equivalent to VegetarianPizza</rdfs:comment>
  <rdfs:label xml:lang="pt">PizzaVegetarianaEquivalente2</rdfs:label>
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Pizza" />
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#FruitTopping" />
  <owl:Class rdf:about="#HerbSpiceTopping" />
  <owl:Class rdf:about="#NutTopping" />
  <owl:Class rdf:about="#SauceTopping" />
  <owl:Class rdf:about="#VegetableTopping" />
  <owl:Class rdf:about="#CheeseTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
      </owl:Class>
- <owl:Class rdf:about="#Veneziana">
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#SultanaTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Parmense" />
  <owl:disjointWith rdf:resource="#Rosa" />
- <owl:disjointWith>
  <owl:Class rdf:about="#FourSeasons" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#QuattroFormaggi" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Cajun" />
- <owl:disjointWith>
  <owl:Class rdf:about="#PolloAdAstra" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PrinceCarlo" />
      </owl:disjointWith>
```

```
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#CaperTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:ID="PineKernels" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#OliveTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#CaperTopping" />
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#OliveTopping" />
  <owl:Class rdf:about="#OnionTopping" />
  <owl:Class rdf:about="#PineKernels" />
  <owl:Class rdf:about="#SultanaTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
```

```
    </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Capricciosa" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Mushroom" />
    </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Napoletana" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Margherita" />
    </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
    </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#OnionTopping" />
    </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
    </owl:onProperty>
    </owl:Restriction>
    </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
- <owl:disjointWith>
  <owl:Class rdf:about="#LaReine" />
    </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Giardiniera" />
    </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
    </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
    </owl:someValuesFrom>
    </owl:Restriction>
    </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
    </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#UnclosedPizza" />
    </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Siciliana" />
  <rdfs:label xml:lang="pt">Veneziana</rdfs:label>
  <owl:disjointWith rdf:resource="#American" />
- <owl:disjointWith>
  <owl:Class rdf:about="#SloppyGiuseppe" />
    </owl:disjointWith>
    </owl:Class>
- <owl:Class rdf:ID="CheeseyVegetableTopping">
  <rdfs:subClassOf rdf:resource="#VegetableTopping" />
  <rdfs:comment xml:lang="en">This class will be inconsistent. This is because we have given it 2 disjoint parents, which
      means it could never have any members (as nothing can simultaneously be a CheeseTopping and a VegetableTopping).
      NB Called ProbeInconsistentTopping in the ProtegeOWL Tutorial.</rdfs:comment>
  <rdfs:label xml:lang="pt">CoberturaDeQueijoComVegetais</rdfs:label>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#CheeseTopping" />
    </rdfs:subClassOf>
    </owl:Class>
- <owl:Class rdf:about="#PetitPoisTopping">
  <owl:disjointWith rdf:resource="#AsparagusTopping" />
```

```
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Mild" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#PepperTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#CaperTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#RocketTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#TomatoTopping" />
  <owl:disjointWith rdf:resource="#ArtichokeTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#OnionTopping" />
      </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#VegetableTopping" />
  <owl:disjointWith rdf:resource="#GarlicTopping" />
  <owl:disjointWith rdf:resource="#SpinachTopping" />
  <owl:disjointWith rdf:resource="#MushroomTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#OliveTopping" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">CoberturaPetitPois</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#LeekTopping" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#SweetPepperTopping">
- <rdfs:subClassOf>
  <owl:Class rdf:about="#PepperTopping" />
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Mild" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#GreenPepperTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#PeperonataTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#JalapenoPepperTopping" />
  <rdfs:label xml:lang="pt">CoberturaDePimentaoDoce</rdfs:label>
      </owl:Class>
- <owl:Class rdf:ID="SpicyPizza">
  <rdfs:label xml:lang="pt">PizzaTemperada</rdfs:label>
  <rdfs:comment xml:lang="en">Any pizza that has a spicy topping is a SpicyPizza</rdfs:comment>
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Pizza" />
```

```
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#SpicyTopping" />
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
      </owl:Class>
- <owl:Class rdf:about="#Medium">
  <rdfs:subClassOf rdf:resource="#Spiciness" />
  <rdfs:label xml:lang="pt">Media</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="#FourCheesesTopping">
  <owl:disjointWith rdf:resource="#GoatsCheeseTopping" />
- <rdfs:subClassOf>
  <owl:Class rdf:about="#CheeseTopping" />
      </rdfs:subClassOf>
  <rdfs:label xml:lang="pt">CoberturaQuatroQueijos</rdfs:label>
  <owl:disjointWith rdf:resource="#GorgonzolaTopping" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Mild" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#ParmesanTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#Giardiniera">
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#PeperonataTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#PrinceCarlo" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#MushroomTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
```

```
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#FourSeasons" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
  <owl:disjointWith rdf:resource="#Siciliana" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#LeekTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#American" />
- <owl:disjointWith>
  <owl:Class rdf:about="#PolloAdAstra" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
  <rdfs:label xml:lang="pt">Giardiniera</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Capricciosa" />
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#SlicedTomatoTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Cajun" />
- <owl:disjointWith>
  <owl:Class rdf:about="#UnclosedPizza" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Rosa" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Veneziana" />
- <rdfs:subClassOf>
- <owl:Restriction>
```

```
  <owl:someValuesFrom rdf:resource="#PetitPoisTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Mushroom" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#OliveTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Napoletana" />
- <owl:disjointWith>
  <owl:Class rdf:about="#SloppyGiuseppe" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Margherita" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Parmense" />
- <owl:disjointWith>
  <owl:Class rdf:about="#LaReine" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#LeekTopping" />
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#MushroomTopping" />
  <owl:Class rdf:about="#OliveTopping" />
  <owl:Class rdf:about="#PeperonataTopping" />
  <owl:Class rdf:about="#PetitPoisTopping" />
  <owl:Class rdf:about="#SlicedTomatoTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#QuattroFormaggi" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#IceCream">
- <rdfs:subClassOf>
  <owl:Class rdf:about="#DomainConcept" />
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Pizza" />
  <owl:disjointWith rdf:resource="#PizzaBase" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
```

```
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#FruitTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#PizzaTopping" />
  <rdfs:comment xml:lang="en">A class to demonstrate mistakes made with setting a property domain. The property
        hasTopping has a domain of Pizza. This means that the reasoner can infer that all individuals using the hasTopping
        property must be of type Pizza. Because of the restriction on this class, all members of IceCream must use the
        hasTopping property, and therefore must also be members of Pizza. However, Pizza and IceCream are disjoint, so this
        causes an inconsistency. If they were not disjoint, IceCream would be inferred to be a subclass of
        Pizza.</rdfs:comment>
  <rdfs:label xml:lang="pt">Sorvete</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="#DomainConcept">
  <owl:disjointWith rdf:resource="#ValuePartition" />
      </owl:Class>
- <owl:Class rdf:about="#QuattroFormaggi">
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PolloAdAstra" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Veneziana" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Mushroom" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Napoletana" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#American" />
- <owl:disjointWith>
  <owl:Class rdf:about="#PrinceCarlo" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Cajun" />
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
- <owl:disjointWith>
  <owl:Class rdf:about="#UnclosedPizza" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Parmense" />
  <owl:disjointWith rdf:resource="#Siciliana" />
  <owl:disjointWith rdf:resource="#Capricciosa" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#FourCheesesTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Rosa" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Margherita" />
      </owl:disjointWith>
```

```
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#FourCheesesTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <rdfs:label xml:lang="pt">QuatroQueijos</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Giardiniera" />
- <owl:disjointWith>
  <owl:Class rdf:about="#SloppyGiuseppe" />
      </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
- <owl:disjointWith>
  <owl:Class rdf:about="#LaReine" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#FourSeasons" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#UnclosedPizza">
  <owl:disjointWith rdf:resource="#Parmense" />
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
- <owl:disjointWith>
  <owl:Class rdf:about="#SloppyGiuseppe" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#American" />
- <owl:disjointWith>
  <owl:Class rdf:about="#LaReine" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#FourSeasons" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PolloAdAstra" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Napoletana" />
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PrinceCarlo" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
```

```
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">An unclosed Pizza cannot be inferred to be
      either a VegetarianPizza or a NonVegetarianPizza, because it might have other toppings.</rdfs:comment>
<owl:disjointWith rdf:resource="#Rosa" />
- <owl:disjointWith>
<owl:Class rdf:about="#AmericanHot" />
    </owl:disjointWith>
<owl:disjointWith rdf:resource="#Giardiniera" />
<owl:disjointWith rdf:resource="#QuattroFormaggi" />
<owl:disjointWith rdf:resource="#Cajun" />
<owl:disjointWith rdf:resource="#Veneziana" />
- <owl:disjointWith>
<owl:Class rdf:about="#Margherita" />
    </owl:disjointWith>
- <owl:disjointWith>
<owl:Class rdf:about="#Mushroom" />
    </owl:disjointWith>
<owl:disjointWith rdf:resource="#Capricciosa" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
<owl:ObjectProperty rdf:about="#hasTopping" />
    </owl:onProperty>
- <owl:someValuesFrom>
<owl:Class rdf:about="#MozzarellaTopping" />
    </owl:someValuesFrom>
    </owl:Restriction>
    </rdfs:subClassOf>
<owl:disjointWith rdf:resource="#Siciliana" />
<rdfs:label xml:lang="pt">PizzaAberta</rdfs:label>
    </owl:Class>
- <owl:Class rdf:about="#ParmesanTopping">
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
<owl:FunctionalProperty rdf:about="#hasSpiciness" />
    </owl:onProperty>
- <owl:someValuesFrom>
<owl:Class rdf:about="#Mild" />
    </owl:someValuesFrom>
    </owl:Restriction>
    </rdfs:subClassOf>
<rdfs:label xml:lang="pt">CoberturaDeParmesao</rdfs:label>
<owl:disjointWith rdf:resource="#FourCheesesTopping" />
- <owl:disjointWith>
<owl:Class rdf:about="#MozzarellaTopping" />
    </owl:disjointWith>
<owl:disjointWith rdf:resource="#GorgonzolaTopping" />
- <rdfs:subClassOf>
<owl:Class rdf:about="#CheeseTopping" />
    </rdfs:subClassOf>
<owl:disjointWith rdf:resource="#GoatsCheeseTopping" />
    </owl:Class>
- <owl:Class rdf:about="#PolloAdAstra">
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="#CajunSpiceTopping" />
<owl:Class rdf:about="#ChickenTopping" />
<owl:Class rdf:about="#GarlicTopping" />
<owl:Class rdf:about="#MozzarellaTopping" />
<owl:Class rdf:about="#RedOnionTopping" />
<owl:Class rdf:about="#SweetPepperTopping" />
```

```
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Capricciosa" />
  <owl:disjointWith rdf:resource="#Rosa" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Veneziana" />
  <owl:disjointWith rdf:resource="#Napoletana" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#ChickenTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Mushroom" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">PolloAdAstra</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#LaReine" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#American" />
  <owl:disjointWith rdf:resource="#QuattroFormaggi" />
- <owl:disjointWith>
  <owl:Class rdf:about="#FourSeasons" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#RedOnionTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Siciliana" />
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Margherita" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Cajun" />
  <owl:disjointWith rdf:resource="#UnclosedPizza" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
```

```
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#SweetPepperTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Parmense" />
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#GarlicTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PrinceCarlo" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Giardiniera" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#CajunSpiceTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#SloppyGiuseppe" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#Margherita">
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PrinceCarlo" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Parmense" />
  <owl:disjointWith rdf:resource="#Rosa" />
  <owl:disjointWith rdf:resource="#American" />
  <owl:disjointWith rdf:resource="#Napoletana" />
  <rdfs:label xml:lang="pt">Margherita</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#FourSeasons" />
      </owl:disjointWith>
- <rdfs:subClassOf>
```

```
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#UnclosedPizza" />
- <owl:disjointWith>
  <owl:Class rdf:about="#SloppyGiuseppe" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Capricciosa" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#LaReine" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Veneziana" />
  <owl:disjointWith rdf:resource="#QuattroFormaggi" />
  <owl:disjointWith rdf:resource="#Cajun" />
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Mushroom" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#PolloAdAstra" />
  <owl:disjointWith rdf:resource="#Giardiniera" />
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
  <owl:disjointWith rdf:resource="#Siciliana" />
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#SloppyGiuseppe">
  <owl:disjointWith rdf:resource="#Margherita" />
- <owl:disjointWith>
```

```
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Giardiniera" />
  <owl:disjointWith rdf:resource="#Cajun" />
  <rdfs:label xml:lang="pt">SloppyGiuseppe</rdfs:label>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
- <owl:disjointWith>
  <owl:Class rdf:about="#PrinceCarlo" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#HotSpicedBeefTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Mushroom" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#PolloAdAstra" />
  <owl:disjointWith rdf:resource="#QuattroFormaggi" />
  <owl:disjointWith rdf:resource="#American" />
  <owl:disjointWith rdf:resource="#Veneziana" />
  <owl:disjointWith rdf:resource="#Parmense" />
  <owl:disjointWith rdf:resource="#UnclosedPizza" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#GreenPepperTopping" />
  <owl:Class rdf:about="#HotSpicedBeefTopping" />
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#OnionTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Napoletana" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#LaReine" />
```

```
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Siciliana" />
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#OnionTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Capricciosa" />
- <owl:disjointWith>
  <owl:Class rdf:about="#FourSeasons" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#GreenPepperTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Rosa" />
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#Mushroom">
  <owl:disjointWith rdf:resource="#Capricciosa" />
  <owl:disjointWith rdf:resource="#SloppyGiuseppe" />
- <owl:disjointWith>
  <owl:Class rdf:about="#LaReine" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#PolloAdAstra" />
  <owl:disjointWith rdf:resource="#American" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Siciliana" />
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#MushroomTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Margherita" />
  <rdfs:label xml:lang="pt">Cogumelo</rdfs:label>
  <owl:disjointWith rdf:resource="#Cajun" />
- <owl:disjointWith>
```

```
  <owl:Class rdf:about="#FourSeasons" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Parmense" />
- <owl:disjointWith>
  <owl:Class rdf:about="#PrinceCarlo" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#QuattroFormaggi" />
  <owl:disjointWith rdf:resource="#Rosa" />
  <owl:disjointWith rdf:resource="#UnclosedPizza" />
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Veneziana" />
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#MushroomTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Giardiniera" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Napoletana" />
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
      </owl:Class>
- <owl:Class rdf:about="#LaReine">
  <owl:disjointWith rdf:resource="#PolloAdAstra" />
  <owl:disjointWith rdf:resource="#Napoletana" />
  <owl:disjointWith rdf:resource="#Giardiniera" />
  <owl:disjointWith rdf:resource="#Veneziana" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
```

```
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#OliveTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#HamTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Siciliana" />
  <rdfs:label xml:lang="pt">LaReine</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#HamTopping" />
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#MushroomTopping" />
  <owl:Class rdf:about="#OliveTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#MushroomTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#QuattroFormaggi" />
  <owl:disjointWith rdf:resource="#UnclosedPizza" />
  <owl:disjointWith rdf:resource="#Margherita" />
  <owl:disjointWith rdf:resource="#American" />
  <owl:disjointWith rdf:resource="#Parmense" />
  <owl:disjointWith rdf:resource="#SloppyGiuseppe" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Mushroom" />
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
  <owl:disjointWith rdf:resource="#Rosa" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
```

```
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#FourSeasons" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Capricciosa" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Cajun" />
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
- <owl:disjointWith>
  <owl:Class rdf:about="#PrinceCarlo" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:ID="MeatyPizza">
  <rdfs:label xml:lang="pt">PizzaDeCarne</rdfs:label>
  <rdfs:comment xml:lang="en">Any pizza that has at least one meat topping</rdfs:comment>
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Pizza" />
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MeatTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
      </owl:Class>
- <owl:Class rdf:about="#VegetarianPizza">
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
- <owl:Class>
- <owl:complementOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MeatTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </owl:complementOf>
      </owl:Class>
```

```
- <owl:Class>
- <owl:complementOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#FishTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </owl:complementOf>
      </owl:Class>
  <owl:Class rdf:about="#Pizza" />
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:label xml:lang="pt">PizzaVegetariana</rdfs:label>
  <rdfs:comment xml:lang="en">Any pizza that does not have fish topping and does not have meat topping is a
        VegetarianPizza. Members of this class do not need to have any toppings at all.</rdfs:comment>
  <owl:disjointWith rdf:resource="#NonVegetarianPizza" />
      </owl:Class>
- <owl:Class rdf:about="#PeperonataTopping">
  <owl:disjointWith rdf:resource="#SweetPepperTopping" />
- <rdfs:subClassOf>
  <owl:Class rdf:about="#PepperTopping" />
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#JalapenoPepperTopping" />
  <owl:disjointWith rdf:resource="#GreenPepperTopping" />
  <rdfs:label xml:lang="pt">CoberturaPeperonata</rdfs:label>
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#Medium" />
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:about="#PrinceCarlo">
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
  <owl:disjointWith rdf:resource="#Mushroom" />
  <owl:disjointWith rdf:resource="#Capricciosa" />
- <owl:disjointWith>
  <owl:Class rdf:about="#FourSeasons" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#American" />
  <owl:disjointWith rdf:resource="#SloppyGiuseppe" />
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#RosemaryTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Margherita" />
  <rdfs:label xml:lang="pt">CoberturaPrinceCarlo</rdfs:label>
  <owl:disjointWith rdf:resource="#LaReine" />
```

```
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Parmense" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#LeekTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Veneziana" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#ParmesanTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Cajun" />
  <owl:disjointWith rdf:resource="#QuattroFormaggi" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#LeekTopping" />
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#ParmesanTopping" />
  <owl:Class rdf:about="#RosemaryTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#UnclosedPizza" />
  <owl:disjointWith rdf:resource="#Rosa" />
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
  <owl:disjointWith rdf:resource="#Napoletana" />
  <owl:disjointWith rdf:resource="#Siciliana" />
```

```
  <owl:disjointWith rdf:resource="#Giardiniera" />
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#PolloAdAstra" />
      </owl:Class>
- <owl:Class rdf:about="#FourSeasons">
  <owl:disjointWith rdf:resource="#Parmense" />
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#MushroomTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#OliveTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#UnclosedPizza" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#PeperoniSausageTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#SloppyGiuseppe" />
  <owl:disjointWith rdf:resource="#Napoletana" />
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#LaReine" />
  <owl:disjointWith rdf:resource="#Cajun" />
- <owl:disjointWith>
  <owl:Class rdf:about="#AmericanHot" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Siciliana" />
  <owl:disjointWith rdf:resource="#Veneziana" />
  <owl:disjointWith rdf:resource="#Giardiniera" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:allValuesFrom>
- <owl:Class>
```

```
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#AnchoviesTopping" />
  <owl:Class rdf:about="#CaperTopping" />
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#MushroomTopping" />
  <owl:Class rdf:about="#OliveTopping" />
  <owl:Class rdf:about="#PeperoniSausageTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Capricciosa" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#AnchoviesTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <rdfs:label xml:lang="pt">QuatroQueijos</rdfs:label>
  <owl:disjointWith rdf:resource="#Mushroom" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#PolloAdAstra" />
  <owl:disjointWith rdf:resource="#Rosa" />
  <owl:disjointWith rdf:resource="#American" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#CaperTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#PrinceCarlo" />
  <owl:disjointWith rdf:resource="#Margherita" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#QuattroFormaggi" />
      </owl:Class>
- <owl:Class rdf:about="#SauceTopping">
  <owl:disjointWith rdf:resource="#FishTopping" />
  <rdfs:subClassOf rdf:resource="#PizzaTopping" />
  <owl:disjointWith rdf:resource="#FruitTopping" />
```

```
- <owl:disjointWith>
  <owl:Class rdf:about="#MeatTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#CheeseTopping" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">CoberturaEmMolho</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#HerbSpiceTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#VegetableTopping" />
  <owl:disjointWith rdf:resource="#NutTopping" />
      </owl:Class>
- <owl:Class rdf:about="#OnionTopping">
  <rdfs:label xml:lang="pt">CoberturaDeCebola</rdfs:label>
  <owl:disjointWith rdf:resource="#AsparagusTopping" />
  <owl:disjointWith rdf:resource="#MushroomTopping" />
  <owl:disjointWith rdf:resource="#ArtichokeTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#RocketTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#PetitPoisTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#LeekTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#GarlicTopping" />
  <owl:disjointWith rdf:resource="#SpinachTopping" />
  <rdfs:subClassOf rdf:resource="#VegetableTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#OliveTopping" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#Medium" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#TomatoTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#CaperTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#PepperTopping" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:about="#HerbSpiceTopping">
  <owl:disjointWith rdf:resource="#FishTopping" />
  <owl:disjointWith rdf:resource="#VegetableTopping" />
  <owl:disjointWith rdf:resource="#NutTopping" />
  <owl:disjointWith rdf:resource="#SauceTopping" />
  <rdfs:label xml:lang="pt">CoberturaDeErvas</rdfs:label>
- <owl:disjointWith>
  <owl:Class rdf:about="#MeatTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#CheeseTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#FruitTopping" />
  <rdfs:subClassOf rdf:resource="#PizzaTopping" />
      </owl:Class>
- <owl:Class rdf:ID="RealItalianPizza">
- <owl:equivalentClass>
```

```
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:ID="hasCountryOfOrigin" />
      </owl:onProperty>
- <owl:hasValue>
  <Country rdf:ID="Italy" />
      </owl:hasValue>
      </owl:Restriction>
  <owl:Class rdf:about="#Pizza" />
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:comment xml:lang="en">This defined class has conditions that are part of the definition: ie any Pizza that has the
        country of origin, Italy is a RealItalianPizza. It also has conditions that merely describe the members - that all
        RealItalianPizzas must only have ThinAndCrispy bases.</rdfs:comment>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:InverseFunctionalProperty rdf:about="#hasBase" />
      </owl:onProperty>
  <owl:allValuesFrom rdf:resource="#ThinAndCrispyBase" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <rdfs:label xml:lang="pt">PizzaItalianaReal</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="#TobascoPepperSauce">
  <rdfs:subClassOf rdf:resource="#SauceTopping" />
  <rdfs:label xml:lang="pt">MolhoTobascoPepper</rdfs:label>
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#Hot" />
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:about="#PeperoniSausageTopping">
  <owl:disjointWith rdf:resource="#HotSpicedBeefTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#HamTopping" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#Medium" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#ChickenTopping" />
- <rdfs:subClassOf>
  <owl:Class rdf:about="#MeatTopping" />
      </rdfs:subClassOf>
  <rdfs:label xml:lang="pt">CoberturaDeCalabreza</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="#AmericanHot">
  <owl:disjointWith rdf:resource="#Veneziana" />
  <owl:disjointWith rdf:resource="#American" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:allValuesFrom>
```

```
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#HotGreenPepperTopping" />
  <owl:Class rdf:about="#JalapenoPepperTopping" />
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#PeperoniSausageTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#HotGreenPepperTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#LaReine" />
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
  <owl:disjointWith rdf:resource="#Margherita" />
  <owl:disjointWith rdf:resource="#QuattroFormaggi" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Giardiniera" />
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
  <owl:disjointWith rdf:resource="#UnclosedPizza" />
  <owl:disjointWith rdf:resource="#Capricciosa" />
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#Caprina" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#PeperoniSausageTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Mushroom" />
```

```
  <owl:disjointWith rdf:resource="#PrinceCarlo" />
  <owl:disjointWith rdf:resource="#Napoletana" />
  <owl:disjointWith rdf:resource="#PolloAdAstra" />
  <owl:disjointWith rdf:resource="#Siciliana" />
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#JalapenoPepperTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#FourSeasons" />
  <rdfs:label xml:lang="pt">AmericanaPicante</rdfs:label>
  <owl:disjointWith rdf:resource="#SloppyGiuseppe" />
  <owl:disjointWith rdf:resource="#Rosa" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Cajun" />
  <owl:disjointWith rdf:resource="#Parmense" />
      </owl:Class>
- <owl:Class rdf:about="#Mild">
  <rdfs:subClassOf rdf:resource="#Spiciness" />
  <owl:disjointWith rdf:resource="#Hot" />
  <rdfs:label xml:lang="pt">NaoPicante</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="#Caprina">
  <owl:disjointWith rdf:resource="#American" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Veneziana" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#GoatsCheeseTopping" />
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#SundriedTomatoTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#AmericanHot" />
  <owl:disjointWith rdf:resource="#Siciliana" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#GoatsCheeseTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#FourSeasons" />
  <owl:disjointWith rdf:resource="#PolloAdAstra" />
  <rdfs:label xml:lang="pt">Caprina</rdfs:label>
- <rdfs:subClassOf>
```

```
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#LaReine" />
  <owl:disjointWith rdf:resource="#Cajun" />
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#SundriedTomatoTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Parmense" />
  <owl:disjointWith rdf:resource="#Mushroom" />
  <owl:disjointWith rdf:resource="#QuattroFormaggi" />
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
  <owl:disjointWith rdf:resource="#Margherita" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Soho" />
      </owl:disjointWith>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Napoletana" />
  <owl:disjointWith rdf:resource="#SloppyGiuseppe" />
  <owl:disjointWith rdf:resource="#Capricciosa" />
  <owl:disjointWith rdf:resource="#UnclosedPizza" />
  <owl:disjointWith rdf:resource="#Giardiniera" />
  <owl:disjointWith rdf:resource="#PrinceCarlo" />
  <owl:disjointWith rdf:resource="#Rosa" />
      </owl:Class>
- <owl:Class rdf:about="#RocketTopping">
  <rdfs:label xml:lang="pt">CoberturaRocket</rdfs:label>
  <rdfs:subClassOf rdf:resource="#VegetableTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#OliveTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#TomatoTopping" />
  <owl:disjointWith rdf:resource="#SpinachTopping" />
  <owl:disjointWith rdf:resource="#AsparagusTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#CaperTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#ArtichokeTopping" />
  <owl:disjointWith rdf:resource="#MushroomTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#PepperTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#OnionTopping" />
```

```
  <owl:disjointWith rdf:resource="#GarlicTopping" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#Medium" />
      </owl:Restriction>
      </rdfs:subClassOf>
- <owl:disjointWith>
  <owl:Class rdf:about="#LeekTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#PetitPoisTopping" />
      </owl:Class>
- <owl:Class rdf:ID="Country">
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
- <owl:Class>
- <owl:oneOf rdf:parseType="Collection">
  <Country rdf:ID="America" />
  <Country rdf:ID="England" />
  <Country rdf:ID="France" />
  <Country rdf:ID="Germany" />
  <Country rdf:about="#Italy" />
      </owl:oneOf>
      </owl:Class>
  <owl:Class rdf:about="#DomainConcept" />
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:label xml:lang="pt">Pais</rdfs:label>
  <rdfs:comment xml:lang="en">A class that is equivalent to the set of individuals that are described in the enumeration - ie
      Countries can only be either America, England, France, Germany or Italy and nothing else. Note that these individuals
      have been asserted to be allDifferent from each other.</rdfs:comment>
      </owl:Class>
- <owl:Class rdf:about="#PepperTopping">
  <rdfs:subClassOf rdf:resource="#VegetableTopping" />
  <owl:disjointWith rdf:resource="#TomatoTopping" />
  <owl:disjointWith rdf:resource="#SpinachTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#OliveTopping" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#LeekTopping" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">CoberturaDePimentao</rdfs:label>
  <owl:disjointWith rdf:resource="#MushroomTopping" />
  <owl:disjointWith rdf:resource="#GarlicTopping" />
  <owl:disjointWith rdf:resource="#AsparagusTopping" />
  <owl:disjointWith rdf:resource="#RocketTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#CaperTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#PetitPoisTopping" />
  <owl:disjointWith rdf:resource="#ArtichokeTopping" />
  <owl:disjointWith rdf:resource="#OnionTopping" />
      </owl:Class>
- <owl:Class rdf:ID="SpicyPizzaEquivalent">
  <rdfs:label xml:lang="pt">PizzaTemperadaEquivalente</rdfs:label>
  <rdfs:comment xml:lang="en">An alternative definition for the SpicyPizza which does away with needing a definition of
      SpicyTopping and uses a slightly more complicated restriction: Pizzas that have at least one topping that is both a
      PizzaTopping and has spiciness hot are members of this class.</rdfs:comment>
- <owl:equivalentClass>
```

```
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Pizza" />
- <owl:Restriction>
- <owl:someValuesFrom>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#PizzaTopping" />
- <owl:Restriction>
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#Hot" />
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
      </owl:Class>
- <owl:Class rdf:about="#PineKernels">
  <rdfs:label xml:lang="pt">CoberturaPineKernels</rdfs:label>
  <rdfs:subClassOf rdf:resource="#NutTopping" />
      </owl:Class>
- <owl:Class rdf:about="#VegetarianTopping">
  <rdfs:label xml:lang="pt">CoberturaVegetariana</rdfs:label>
  <rdfs:comment xml:lang="en">An example of a covering axiom. VegetarianTopping is equivalent to the union of all
      toppings in the given axiom. VegetarianToppings can only be Cheese or Vegetable or....etc.</rdfs:comment>
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#PizzaTopping" />
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#VegetableTopping" />
  <owl:Class rdf:about="#CheeseTopping" />
  <owl:Class rdf:about="#HerbSpiceTopping" />
  <owl:Class rdf:about="#FruitTopping" />
  <owl:Class rdf:about="#NutTopping" />
  <owl:Class rdf:about="#SauceTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
      </owl:Class>
- <owl:Class rdf:about="#LeekTopping">
  <owl:disjointWith rdf:resource="#SpinachTopping" />
  <owl:disjointWith rdf:resource="#ArtichokeTopping" />
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#Mild" />
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#AsparagusTopping" />
  <owl:disjointWith rdf:resource="#PetitPoisTopping" />
```

```
  <owl:disjointWith rdf:resource="#GarlicTopping" />
  <owl:disjointWith rdf:resource="#OnionTopping" />
  <owl:disjointWith rdf:resource="#PepperTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#OliveTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#RocketTopping" />
- <owl:disjointWith>
  <owl:Class rdf:about="#CaperTopping" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">CoberturaDeLeek</rdfs:label>
  <owl:disjointWith rdf:resource="#MushroomTopping" />
  <rdfs:subClassOf rdf:resource="#VegetableTopping" />
  <owl:disjointWith rdf:resource="#TomatoTopping" />
      </owl:Class>
- <owl:Class rdf:about="#OliveTopping">
- <owl:disjointWith>
  <owl:Class rdf:about="#CaperTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#PetitPoisTopping" />
  <owl:disjointWith rdf:resource="#PepperTopping" />
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#Mild" />
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <rdfs:label xml:lang="pt">CoberturaDeAzeitona</rdfs:label>
  <owl:disjointWith rdf:resource="#OnionTopping" />
  <owl:disjointWith rdf:resource="#LeekTopping" />
  <owl:disjointWith rdf:resource="#GarlicTopping" />
  <owl:disjointWith rdf:resource="#ArtichokeTopping" />
  <owl:disjointWith rdf:resource="#MushroomTopping" />
  <owl:disjointWith rdf:resource="#AsparagusTopping" />
  <owl:disjointWith rdf:resource="#RocketTopping" />
  <owl:disjointWith rdf:resource="#SpinachTopping" />
  <owl:disjointWith rdf:resource="#TomatoTopping" />
  <rdfs:subClassOf rdf:resource="#VegetableTopping" />
      </owl:Class>
- <owl:Class rdf:ID="InterestingPizza">
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
- <owl:Restriction>
  <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">3</owl:minCardinality>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
  <owl:Class rdf:about="#Pizza" />
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:comment xml:lang="en">Any pizza that has at least 3 toppings. Note that this is a cardinality constraint on the
      hasTopping property and NOT a qualified cardinality constraint (QCR). A QCR would specify from which class the
      members in this relationship must be. eg has at least 3 toppings from PizzaTopping. This is currently not supported in
      OWL.</rdfs:comment>
  <rdfs:label xml:lang="pt">PizzaInteressante</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="#MixedSeafoodTopping">
  <rdfs:subClassOf rdf:resource="#FishTopping" />
  <owl:disjointWith rdf:resource="#AnchoviesTopping" />
```

```
  <rdfs:label xml:lang="pt">CoberturaDeFrutosDoMarMistos</rdfs:label>
  <owl:disjointWith rdf:resource="#PrawnsTopping" />
      </owl:Class>
- <owl:Class rdf:about="#Soho">
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#OliveTopping" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#GarlicTopping" />
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#OliveTopping" />
  <owl:Class rdf:about="#ParmesanTopping" />
  <owl:Class rdf:about="#RocketTopping" />
  <owl:Class rdf:about="#TomatoTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Parmense" />
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#GarlicTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#FourSeasons" />
  <owl:disjointWith rdf:resource="#Caprina" />
  <owl:disjointWith rdf:resource="#Rosa" />
  <owl:disjointWith rdf:resource="#UnclosedPizza" />
  <owl:disjointWith rdf:resource="#QuattroFormaggi" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#ParmesanTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Veneziana" />
  <owl:disjointWith rdf:resource="#Napoletana" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
```

```
  <owl:disjointWith rdf:resource="#AmericanHot" />
  <owl:disjointWith rdf:resource="#LaReine" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#MozzarellaTopping" />
      </owl:someValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#RocketTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Mushroom" />
  <owl:disjointWith rdf:resource="#Capricciosa" />
  <owl:disjointWith rdf:resource="#Giardiniera" />
  <owl:disjointWith rdf:resource="#PolloAdAstra" />
  <owl:disjointWith rdf:resource="#Margherita" />
  <owl:disjointWith rdf:resource="#American" />
  <owl:disjointWith rdf:resource="#Siciliana" />
  <owl:disjointWith rdf:resource="#PrinceCarlo" />
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
  <owl:disjointWith rdf:resource="#Cajun" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Fiorentina" />
      </owl:disjointWith>
  <rdfs:label xml:lang="pt">Soho</rdfs:label>
  <owl:disjointWith rdf:resource="#SloppyGiuseppe" />
      </owl:Class>
- <owl:Class rdf:about="#HamTopping">
  <owl:disjointWith rdf:resource="#PeperoniSausageTopping" />
- <rdfs:subClassOf>
  <owl:Class rdf:about="#MeatTopping" />
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#HotSpicedBeefTopping" />
  <rdfs:label xml:lang="pt">CoberturaDePresunto</rdfs:label>
  <owl:disjointWith rdf:resource="#ChickenTopping" />
      </owl:Class>
- <owl:Class rdf:about="#MeatTopping">
- <owl:disjointWith>
  <owl:Class rdf:about="#CheeseTopping" />
      </owl:disjointWith>
  <owl:disjointWith rdf:resource="#HerbSpiceTopping" />
  <owl:disjointWith rdf:resource="#VegetableTopping" />
  <owl:disjointWith rdf:resource="#FruitTopping" />
  <rdfs:label xml:lang="pt">CoberturaDeCarne</rdfs:label>
  <owl:disjointWith rdf:resource="#SauceTopping" />
  <owl:disjointWith rdf:resource="#NutTopping" />
  <rdfs:subClassOf rdf:resource="#PizzaTopping" />
  <owl:disjointWith rdf:resource="#FishTopping" />
      </owl:Class>
- <owl:Class rdf:about="#MozzarellaTopping">
  <owl:disjointWith rdf:resource="#GoatsCheeseTopping" />
  <owl:disjointWith rdf:resource="#FourCheesesTopping" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
```

```
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#Mild" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <rdfs:label xml:lang="pt">CoberturaDeMozzarella</rdfs:label>
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:onProperty rdf:resource="#hasCountryOfOrigin" />
  <owl:hasValue rdf:resource="#Italy" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#GorgonzolaTopping" />
  <owl:disjointWith rdf:resource="#ParmesanTopping" />
- <rdfs:subClassOf>
  <owl:Class rdf:about="#CheeseTopping" />
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:about="#SundriedTomatoTopping">
  <owl:disjointWith rdf:resource="#SlicedTomatoTopping" />
  <rdfs:subClassOf rdf:resource="#TomatoTopping" />
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#Mild" />
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <rdfs:label xml:lang="pt">CoberturaDeTomateRessecadoAoSol</rdfs:label>
      </owl:Class>
- <owl:Class rdf:about="#CaperTopping">
  <owl:disjointWith rdf:resource="#PetitPoisTopping" />
  <owl:disjointWith rdf:resource="#SpinachTopping" />
  <owl:disjointWith rdf:resource="#TomatoTopping" />
  <owl:disjointWith rdf:resource="#PepperTopping" />
  <rdfs:label xml:lang="pt">CoberturaDeCaper</rdfs:label>
  <owl:disjointWith rdf:resource="#RocketTopping" />
  <owl:disjointWith rdf:resource="#AsparagusTopping" />
  <owl:disjointWith rdf:resource="#OnionTopping" />
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#Mild" />
- <owl:onProperty>
  <owl:FunctionalProperty rdf:about="#hasSpiciness" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#OliveTopping" />
  <owl:disjointWith rdf:resource="#GarlicTopping" />
  <owl:disjointWith rdf:resource="#MushroomTopping" />
  <rdfs:subClassOf rdf:resource="#VegetableTopping" />
  <owl:disjointWith rdf:resource="#LeekTopping" />
  <owl:disjointWith rdf:resource="#ArtichokeTopping" />
      </owl:Class>
- <owl:Class rdf:about="#CheeseTopping">
  <owl:disjointWith rdf:resource="#HerbSpiceTopping" />
  <owl:disjointWith rdf:resource="#VegetableTopping" />
  <rdfs:subClassOf rdf:resource="#PizzaTopping" />
  <owl:disjointWith rdf:resource="#NutTopping" />
  <owl:disjointWith rdf:resource="#MeatTopping" />
  <owl:disjointWith rdf:resource="#FishTopping" />
  <owl:disjointWith rdf:resource="#SauceTopping" />
  <rdfs:label xml:lang="pt">CoberturaDeQueijo</rdfs:label>
```

```
  <owl:disjointWith rdf:resource="#FruitTopping" />
      </owl:Class>
- <owl:Class rdf:about="#Fiorentina">
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#GarlicTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Soho" />
  <owl:disjointWith rdf:resource="#LaReine" />
  <owl:disjointWith rdf:resource="#PrinceCarlo" />
  <owl:disjointWith rdf:resource="#Giardiniera" />
  <owl:disjointWith rdf:resource="#Rosa" />
  <owl:disjointWith rdf:resource="#FruttiDiMare" />
  <owl:disjointWith rdf:resource="#Cajun" />
  <owl:disjointWith rdf:resource="#American" />
  <owl:disjointWith rdf:resource="#PolloAdAstra" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#OliveTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <rdfs:label xml:lang="pt">Fiorentina</rdfs:label>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#SpinachTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Mushroom" />
  <owl:disjointWith rdf:resource="#Margherita" />
  <owl:disjointWith rdf:resource="#QuattroFormaggi" />
  <owl:disjointWith rdf:resource="#Caprina" />
  <rdfs:subClassOf rdf:resource="#NamedPizza" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#MozzarellaTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#FourSeasons" />
  <owl:disjointWith rdf:resource="#AmericanHot" />
  <owl:disjointWith rdf:resource="#UnclosedPizza" />
  <owl:disjointWith rdf:resource="#Siciliana" />
  <owl:disjointWith rdf:resource="#Veneziana" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
- <owl:allValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#ParmesanTopping" />
```

```
  <owl:Class rdf:about="#TomatoTopping" />
  <owl:Class rdf:about="#MozzarellaTopping" />
  <owl:Class rdf:about="#SpinachTopping" />
  <owl:Class rdf:about="#GarlicTopping" />
  <owl:Class rdf:about="#OliveTopping" />
      </owl:unionOf>
      </owl:Class>
      </owl:allValuesFrom>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Capricciosa" />
  <owl:disjointWith rdf:resource="#Napoletana" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#TomatoTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Parmense" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasTopping" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#ParmesanTopping" />
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#SloppyGiuseppe" />
      </owl:Class>
- <owl:Class rdf:about="#DeepPanBase">
  <rdfs:subClassOf rdf:resource="#PizzaBase" />
  <owl:disjointWith rdf:resource="#ThinAndCrispyBase" />
  <rdfs:label xml:lang="pt">BaseEspessa</rdfs:label>
      </owl:Class>
- <owl:ObjectProperty rdf:ID="isIngredientOf">
  <rdfs:comment xml:lang="en">The inverse property tree to hasIngredient - all subproperties and attributes of the properties
      should reflect those under hasIngredient.</rdfs:comment>
- <owl:inverseOf>
  <owl:ObjectProperty rdf:ID="hasIngredient" />
      </owl:inverseOf>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty" />
      </owl:ObjectProperty>
- <owl:ObjectProperty rdf:about="#hasTopping">
  <rdfs:domain rdf:resource="#Pizza" />
- <owl:inverseOf>
  <owl:ObjectProperty rdf:ID="isToppingOf" />
      </owl:inverseOf>
- <rdfs:subPropertyOf>
  <owl:ObjectProperty rdf:about="#hasIngredient" />
      </rdfs:subPropertyOf>
  <rdfs:range rdf:resource="#PizzaTopping" />
      </owl:ObjectProperty>
- <owl:ObjectProperty rdf:about="#isToppingOf">
  <rdfs:subPropertyOf rdf:resource="#isIngredientOf" />
  <rdfs:domain rdf:resource="#PizzaTopping" />
  <owl:inverseOf rdf:resource="#hasTopping" />
  <rdfs:range rdf:resource="#Pizza" />
      </owl:ObjectProperty>
- <owl:ObjectProperty rdf:about="#hasIngredient">
  <rdfs:comment xml:lang="en">NB Transitive - the ingredients of ingredients are ingredients of the whole</rdfs:comment>
  <owl:inverseOf rdf:resource="#isIngredientOf" />
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty" />
```

```
    </owl:ObjectProperty>
- <owl:FunctionalProperty rdf:about="#hasSpiciness">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty" />
  <rdfs:range rdf:resource="#Spiciness" />
  <rdfs:comment xml:lang="en">A property created to be used with the ValuePartition - Spiciness.</rdfs:comment>
    </owl:FunctionalProperty>
- <owl:InverseFunctionalProperty rdf:ID="isBaseOf">
- <owl:inverseOf>
  <owl:InverseFunctionalProperty rdf:about="#hasBase" />
    </owl:inverseOf>
  <rdfs:subPropertyOf rdf:resource="#isIngredientOf" />
  <rdfs:domain rdf:resource="#PizzaBase" />
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty" />
  <rdfs:range rdf:resource="#Pizza" />
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty" />
    </owl:InverseFunctionalProperty>
- <owl:InverseFunctionalProperty rdf:about="#hasBase">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty" />
  <owl:inverseOf rdf:resource="#isBaseOf" />
  <rdfs:range rdf:resource="#PizzaBase" />
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty" />
  <rdfs:subPropertyOf rdf:resource="#hasIngredient" />
  <rdfs:domain rdf:resource="#Pizza" />
    </owl:InverseFunctionalProperty>
- <owl:AllDifferent>
- <owl:distinctMembers rdf:parseType="Collection">
  <Country rdf:about="#America" />
  <Country rdf:about="#England" />
  <Country rdf:about="#France" />
  <Country rdf:about="#Germany" />
  <Country rdf:about="#Italy" />
    </owl:distinctMembers>
    </owl:AllDifferent>
    </rdf:RDF>
```

Wine Ontology

Namespace: http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#
Description: Sample ontology used in the OWL specification documents.
Location: http://www.w3.org/TR/owl-guide/wine.rdf

```
><?xml version="1.0"?>
><!DOCTYPE rdf:RDF [
>    <!ENTITY vin  "http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#" >
>    <!ENTITY food "http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#" >
>    <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
>    <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
>  ]>
>
><rdf:RDF
>  xmlns     = "http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#"
>  xmlns:vin = "http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#"
>  xml:base  = "http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#"
>  xmlns:food= "http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#"
>  xmlns:owl = "http://www.w3.org/2002/07/owl#"
>  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>  xmlns:rdfs= "http://www.w3.org/2000/01/rdf-schema#"
>  xmlns:xsd = "http://www.w3.org/2001/XMLSchema#">
>
>  <owl:Ontology rdf:about="">
>    <rdfs:comment>An example OWL ontology</rdfs:comment>
>    <owl:priorVersion>
>      <owl:Ontology rdf:about="http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine"/>
>    </owl:priorVersion>
>    <owl:imports rdf:resource="http://www.w3.org/TR/2003/PR-owl-guide-20031209/food"/>
>    <rdfs:comment>Derived from the DAML Wine ontology at
>      http://ontolingua.stanford.edu/doc/chimaera/ontologies/wines.daml
>      Substantially changed, in particular the Region based relations.
>    </rdfs:comment>
>    <rdfs:label>Wine Ontology</rdfs:label>
>  </owl:Ontology>
>
>  <owl:Class rdf:ID="Wine">
>    <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
>    <rdfs:subClassOf>
>      <owl:Restriction>
>        <owl:onProperty rdf:resource="#hasMaker" />
>            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
>      </owl:Restriction>
>    </rdfs:subClassOf>
>    <rdfs:subClassOf>
>      <owl:Restriction>
>        <owl:onProperty rdf:resource="#hasMaker" />
```

```
>            <owl:allValuesFrom rdf:resource="#Winery" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>            <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn"/>
>       <owl:someValuesFrom rdf:resource="&vin;Region"/>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:label xml:lang="en">wine</rdfs:label>
>   <rdfs:label xml:lang="fr">vin</rdfs:label>
> </owl:Class>
>
> <owl:Class rdf:ID="Vintage">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasVintageYear"/>
>       <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
>     </owl:Restriction>
>   </rdfs:subClassOf>
```

```
> </owl:Class>
>
> <owl:Class rdf:ID="WineGrape">
>   <rdfs:subClassOf rdf:resource="&food;Grape" />
> </owl:Class>
>
> <owl:Class rdf:ID="WhiteWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#White" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="WhiteTableWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#TableWine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#White" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="WhiteNonSweetWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#WhiteWine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Dry" />
>             <owl:Thing rdf:about="#OffDry" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="WhiteLoire">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Loire" />
>     <owl:Class rdf:about="#WhiteWine" />
>   </owl:intersectionOf>
> </owl:Class>
>
```

```
> <owl:Class rdf:about="#WhiteLoire">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#CheninBlancGrape" />
>             <owl:Thing rdf:about="#PinotBlancGrape" />
>             <owl:Thing rdf:about="#SauvignonBlancGrape" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
> </owl:Class>
>
> <owl:Class rdf:ID="WhiteBurgundy">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Burgundy" />
>     <owl:Class rdf:about="#WhiteWine" />
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:about="#WhiteBurgundy">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#ChardonnayGrape" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </rdfs:subClassOf>
> </owl:Class>
>
> <owl:Class rdf:ID="WhiteBordeaux">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Bordeaux" />
>     <owl:Class rdf:about="#WhiteWine" />
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:about="#WhiteBordeaux">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
```

```
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#SemillonGrape" />
>             <owl:Thing rdf:about="#SauvignonBlancGrape" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Region" />
>
> <owl:ObjectProperty rdf:ID="locatedIn">
>   <rdf:type rdf:resource="&owl;TransitiveProperty" />
>   <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
>   <rdfs:range rdf:resource="#Region" />
> </owl:ObjectProperty>
>
> <owl:ObjectProperty rdf:ID="adjacentRegion">
>   <rdf:type rdf:resource="&owl;SymmetricProperty" />
>   <rdfs:domain rdf:resource="#Region" />
>   <rdfs:range rdf:resource="#Region" />
> </owl:ObjectProperty>
>
> <owl:Class rdf:ID="VintageYear" />
>
> <owl:DatatypeProperty rdf:ID="yearValue">
>   <rdfs:domain rdf:resource="#VintageYear" />
>   <rdfs:range  rdf:resource="&xsd;positiveInteger" />
> </owl:DatatypeProperty>
>
> <VintageYear rdf:ID="Year1998">
>   <yearValue rdf:datatype="&xsd;positiveInteger">1998</yearValue>
> </VintageYear>
>
> <owl:ObjectProperty rdf:ID="hasVintageYear">
>   <rdf:type rdf:resource="&owl;FunctionalProperty" />
>   <rdfs:domain rdf:resource="#Vintage" />
>   <rdfs:range  rdf:resource="#VintageYear" />
> </owl:ObjectProperty>
>
> <owl:ObjectProperty rdf:ID="madeFromGrape">
>   <rdfs:subPropertyOf rdf:resource="&food;madeFromFruit" />
>   <rdfs:domain rdf:resource="#Wine" />
>   <rdfs:range rdf:resource="#WineGrape" />
> </owl:ObjectProperty>
>
> <owl:ObjectProperty rdf:ID="madeIntoWine">
```

```
>   <owl:inverseOf rdf:resource="#madeFromGrape" />
> </owl:ObjectProperty>
>
> <owl:ObjectProperty rdf:ID="hasWineDescriptor">
>   <rdfs:domain rdf:resource="#Wine" />
>   <rdfs:range  rdf:resource="#WineDescriptor" />
> </owl:ObjectProperty>
>
> <owl:ObjectProperty rdf:ID="hasSugar">
>   <rdf:type rdf:resource="&owl;FunctionalProperty" />
>   <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
>   <rdfs:range rdf:resource="#WineSugar" />
> </owl:ObjectProperty>
>
> <owl:ObjectProperty rdf:ID="hasBody">
>   <rdf:type rdf:resource="&owl;FunctionalProperty" />
>   <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
>   <rdfs:range rdf:resource="#WineBody" />
> </owl:ObjectProperty>
>
> <owl:ObjectProperty rdf:ID="hasFlavor">
>   <rdf:type rdf:resource="&owl;FunctionalProperty" />
>   <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
>   <rdfs:range rdf:resource="#WineFlavor" />
> </owl:ObjectProperty>
>
> <owl:ObjectProperty rdf:ID="hasColor">
>   <rdf:type rdf:resource="&owl;FunctionalProperty" />
>   <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
>   <rdfs:domain rdf:resource="#Wine" />
>   <rdfs:range rdf:resource="#WineColor" />
> </owl:ObjectProperty>
>
> <owl:ObjectProperty rdf:ID="hasMaker">
>   <rdf:type rdf:resource="&owl;FunctionalProperty" />
> </owl:ObjectProperty>
>
> <owl:ObjectProperty rdf:ID="producesWine">
>   <owl:inverseOf rdf:resource="#hasMaker" />
> </owl:ObjectProperty>
>
> <owl:Class rdf:ID="Zinfandel">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#ZinfandelGrape" />
>     </owl:Restriction>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
```

```
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:about="#Zinfandel">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#Red" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Dry" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Full" />
>             <owl:Thing rdf:about="#Medium" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Moderate" />
>             <owl:Thing rdf:about="#Strong" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Winery" />
>
> <owl:Class rdf:ID="WineDescriptor">
>   <rdfs:comment>Made WineDescriptor unionType of tastes and color</rdfs:comment>
```

```
>   <owl:unionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#WineTaste" />
>     <owl:Class rdf:about="#WineColor" />
>   </owl:unionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="WineTaste">
>   <rdfs:subClassOf rdf:resource="#WineDescriptor" />
> </owl:Class>
>
> <owl:Class rdf:ID="WineColor">
>   <rdfs:subClassOf rdf:resource="#WineDescriptor" />
>   <owl:oneOf rdf:parseType="Collection">
>     <owl:Thing rdf:about="#White" />
>     <owl:Thing rdf:about="#Rose" />
>     <owl:Thing rdf:about="#Red" />
>   </owl:oneOf>
> </owl:Class>
>
> <owl:Class rdf:ID="WineSugar">
>   <rdfs:subClassOf rdf:resource="#WineTaste" />
>   <owl:oneOf rdf:parseType="Collection">
>     <owl:Thing rdf:about="#Sweet" />
>     <owl:Thing rdf:about="#OffDry" />
>     <owl:Thing rdf:about="#Dry" />
>   </owl:oneOf>
> </owl:Class>
>
> <owl:Class rdf:ID="WineFlavor">
>   <rdfs:subClassOf rdf:resource="#WineTaste" />
>   <owl:oneOf rdf:parseType="Collection">
>     <owl:Thing rdf:about="#Delicate" />
>     <owl:Thing rdf:about="#Moderate" />
>     <owl:Thing rdf:about="#Strong" />
>   </owl:oneOf>
> </owl:Class>
>
> <owl:Class rdf:ID="WineBody">
>   <rdfs:subClassOf rdf:resource="#WineTaste" />
>   <owl:oneOf rdf:parseType="Collection">
>     <owl:Thing rdf:about="#Light" />
>     <owl:Thing rdf:about="#Medium" />
>     <owl:Thing rdf:about="#Full" />
>   </owl:oneOf>
> </owl:Class>
>
> <Region rdf:ID="USRegion" />
>
> <owl:Class rdf:ID="Tours">
>   <owl:intersectionOf rdf:parseType="Collection">
```

```
>     <owl:Class rdf:about="#Loire" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#ToursRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:about="#Tours">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#CheninBlancGrape" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </rdfs:subClassOf>
> </owl:Class>
>
> <owl:Class rdf:ID="TableWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Dry" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="SweetWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Sweet" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="SweetRiesling">
>   <rdfs:subClassOf rdf:resource="#DessertWine" />
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:hasValue rdf:resource="#Full" />
>     </owl:Restriction>
```

```
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Moderate" />
>             <owl:Thing rdf:about="#Strong" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Riesling" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Sweet" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="StEmilion">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#Red" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:hasValue rdf:resource="#Strong" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#CabernetSauvignonGrape" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Bordeaux" />
```

```
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#StEmilionRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="SemillonOrSauvignonBlanc">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#White" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Medium" />
>             <owl:Thing rdf:about="#Full" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#SemillonGrape" />
>             <owl:Thing rdf:about="#SauvignonBlancGrape" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Semillon">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#SemillonOrSauvignonBlanc" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#SemillonGrape" />
>     </owl:Restriction>
```

```
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="SauvignonBlanc">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#SemillonOrSauvignonBlanc" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#SauvignonBlancGrape" />
>     </owl:Restriction>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Sauternes">
>   <rdfs:subClassOf rdf:resource="#LateHarvest" />
>   <rdfs:subClassOf rdf:resource="#Bordeaux" />
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#SauterneRegion" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:hasValue rdf:resource="#Medium" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#White" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Sancerre">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:hasValue rdf:resource="#Medium" />
>     </owl:Restriction>
```

```
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#OffDry" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:hasValue rdf:resource="#Delicate" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#SauvignonBlancGrape" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Loire" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#SancerreRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="RoseWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#Rose" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Riesling">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#White" />
>     </owl:Restriction>
```

```
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#RieslingGrape" />
>     </owl:Restriction>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="RedWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#Red" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="RedTableWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#TableWine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#Red" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="RedBurgundy">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#PinotNoirGrape" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Burgundy" />
>     <owl:Class rdf:about="#RedWine" />
```

```
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="RedBordeaux">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#CabernetSauvignonGrape" />
>             <owl:Thing rdf:about="#MerlotGrape" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Bordeaux" />
>     <owl:Class rdf:about="#RedWine" />
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Port">
>   <rdfs:subClassOf rdf:resource="#RedWine" />
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#PortugalRegion" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:hasValue rdf:resource="#Full" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:hasValue rdf:resource="#Strong" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Sweet" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
> </owl:Class>
```

```
>
> <owl:Class rdf:ID="PinotNoir">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#Red" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#PinotNoirGrape" />
>     </owl:Restriction>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="PinotBlanc">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#White" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#PinotBlancGrape" />
>     </owl:Restriction>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="PetiteSyrah">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#Red" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
```

```
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Dry" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Moderate" />
>             <owl:Thing rdf:about="#Strong" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Medium" />
>             <owl:Thing rdf:about="#Full" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#PetiteSyrahGrape" />
>     </owl:Restriction>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Pauillac">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:hasValue rdf:resource="#Full" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
```

```
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:hasValue rdf:resource="#Strong" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#CabernetSauvignonGrape" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Medoc" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#PauillacRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Muscadet">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:hasValue rdf:resource="#Light" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:hasValue rdf:resource="#Delicate" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Dry" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#PinotBlancGrape" />
```

```
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Loire" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#MuscadetRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Meursault">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:hasValue rdf:resource="#Full" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#WhiteBurgundy" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#MeursaultRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Merlot">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#Red" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Dry" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:allValuesFrom>
```

```
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Moderate" />
>             <owl:Thing rdf:about="#Delicate" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Light" />
>             <owl:Thing rdf:about="#Medium" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#MerlotGrape" />
>     </owl:Restriction>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Meritage">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#Red" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#CabernetSauvignonGrape" />
```

```
>            <owl:Thing rdf:about="#CabernetFrancGrape" />
>            <owl:Thing rdf:about="#MalbecGrape" />
>            <owl:Thing rdf:about="#PetiteVerdotGrape" />
>            <owl:Thing rdf:about="#MerlotGrape" />
>          </owl:oneOf>
>        </owl:Class>
>      </owl:allValuesFrom>
>    </owl:Restriction>
>    <owl:Restriction>
>      <owl:onProperty rdf:resource="#madeFromGrape" />
>      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">2</owl:minCardinality>
>    </owl:Restriction>
>  </owl:intersectionOf>
> </owl:Class>
>
> <Region rdf:ID="MedocRegion">
>   <locatedIn rdf:resource="#BordeauxRegion" />
> </Region>
>
> <owl:Class rdf:ID="Medoc">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#Red" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Dry" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Bordeaux" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#MedocRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Margaux">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:hasValue rdf:resource="#Delicate" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
```

```
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#MerlotGrape" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Medoc" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#MargauxRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <Region rdf:ID="LoireRegion">
>   <locatedIn rdf:resource="#FrenchRegion" />
> </Region>
>
> <owl:Class rdf:ID="Loire">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#LoireRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="LateHarvest">
>   <rdfs:subClassOf rdf:resource="#Wine" />
>   <owl:disjointWith rdf:resource="#EarlyHarvest" />
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Sweet" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Moderate" />
>             <owl:Thing rdf:about="#Strong" />
```

```
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
> </owl:Class>
>
> <owl:Class rdf:ID="ItalianWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#ItalianRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <Region rdf:ID="ItalianRegion" />
>
> <owl:Class rdf:ID="IceWine">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Medium" />
>             <owl:Thing rdf:about="#Full" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Moderate" />
>             <owl:Thing rdf:about="#Strong" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#LateHarvest" />
>     <owl:Class rdf:about="#DessertWine" />
>     <owl:Restriction>
```

```
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#White" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="GermanWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#GermanyRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Gamay">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#GamayGrape" />
>     </owl:Restriction>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="FullBodiedWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:hasValue rdf:resource="#Full" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <Region rdf:ID="FrenchRegion" />
>
> <owl:Class rdf:ID="FrenchWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#FrenchRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
```

```
> </owl:Class>
>
> <owl:Class rdf:ID="EarlyHarvest">
>   <rdfs:subClassOf rdf:resource="#Wine" />
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Dry" />
>             <owl:Thing rdf:about="#OffDry" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
> </owl:Class>
>
> <owl:Class rdf:ID="DryWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Dry" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="DryWhiteWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#DryWine" />
>     <owl:Class rdf:about="#WhiteWine" />
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="DryRiesling">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#White" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:hasValue rdf:resource="#Delicate" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
```

```
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Light" />
>             <owl:Thing rdf:about="#Medium" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Riesling" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Dry" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="DryRedWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#DryWine" />
>     <owl:Class rdf:about="#RedWine" />
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="DessertWine">
>   <rdfs:subClassOf rdf:resource="#Wine" />
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#OffDry" />
>             <owl:Thing rdf:about="#Sweet" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
> </owl:Class>
>
> <owl:Class rdf:ID="CotesDOr">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:hasValue rdf:resource="#Moderate" />
```

```
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#RedBurgundy" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#CotesDOrRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Chianti">
>   <rdfs:subClassOf rdf:resource="#ItalianWine" />
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#ChiantiRegion" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#Red" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#SangioveseGrape" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:hasValue rdf:resource="#Moderate" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Dry" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Light" />
```

```
>            <owl:Thing rdf:about="#Medium" />
>          </owl:oneOf>
>        </owl:Class>
>      </owl:allValuesFrom>
>    </owl:Restriction>
>  </rdfs:subClassOf>
> </owl:Class>
>
> <owl:Class rdf:ID="CheninBlanc">
>  <rdfs:subClassOf>
>    <owl:Restriction>
>      <owl:onProperty rdf:resource="#hasColor" />
>      <owl:hasValue rdf:resource="#White" />
>    </owl:Restriction>
>  </rdfs:subClassOf>
>  <rdfs:subClassOf>
>    <owl:Restriction>
>      <owl:onProperty rdf:resource="#hasFlavor" />
>      <owl:hasValue rdf:resource="#Moderate" />
>    </owl:Restriction>
>  </rdfs:subClassOf>
>  <rdfs:subClassOf>
>    <owl:Restriction>
>      <owl:onProperty rdf:resource="#hasBody" />
>      <owl:allValuesFrom>
>        <owl:Class>
>          <owl:oneOf rdf:parseType="Collection">
>            <owl:Thing rdf:about="#Full" />
>            <owl:Thing rdf:about="#Medium" />
>          </owl:oneOf>
>        </owl:Class>
>      </owl:allValuesFrom>
>    </owl:Restriction>
>  </rdfs:subClassOf>
>  <rdfs:subClassOf>
>    <owl:Restriction>
>      <owl:onProperty rdf:resource="#hasSugar" />
>      <owl:allValuesFrom>
>        <owl:Class>
>          <owl:oneOf rdf:parseType="Collection">
>            <owl:Thing rdf:about="#Dry" />
>            <owl:Thing rdf:about="#OffDry" />
>          </owl:oneOf>
>        </owl:Class>
>      </owl:allValuesFrom>
>    </owl:Restriction>
>  </rdfs:subClassOf>
>  <owl:intersectionOf rdf:parseType="Collection">
>    <owl:Class rdf:about="#Wine" />
>    <owl:Restriction>
```

```
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#CheninBlancGrape" />
>     </owl:Restriction>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Chardonnay">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#White" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Full" />
>             <owl:Thing rdf:about="#Medium" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Strong" />
>             <owl:Thing rdf:about="#Moderate" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#ChardonnayGrape" />
>     </owl:Restriction>
>     <owl:Restriction>
```

```
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <Region rdf:ID="CaliforniaRegion">
>   <locatedIn rdf:resource="#USRegion" />
> </Region>
>
> <Region rdf:ID="TexasRegion">
>   <locatedIn rdf:resource="#USRegion" />
> </Region>
>
> <owl:Class rdf:ID="CaliforniaWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#CaliforniaRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="TexasWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#TexasRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="CabernetSauvignon">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#Red" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Dry" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
```

```
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Moderate" />
>             <owl:Thing rdf:about="#Strong" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:allValuesFrom>
>         <owl:Class>
>           <owl:oneOf rdf:parseType="Collection">
>             <owl:Thing rdf:about="#Medium" />
>             <owl:Thing rdf:about="#Full" />
>           </owl:oneOf>
>         </owl:Class>
>       </owl:allValuesFrom>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#CabernetSauvignonGrape" />
>     </owl:Restriction>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="CabernetFranc">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#Red" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:hasValue rdf:resource="#Moderate" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
```

```
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:hasValue rdf:resource="#Medium" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Dry" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#CabernetFrancGrape" />
>     </owl:Restriction>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Burgundy">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#BourgogneRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Dry" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
> </owl:Class>
>
> <Region rdf:ID="BourgogneRegion">
>   <locatedIn rdf:resource="#FrenchRegion" />
> </Region>
>
> <Region rdf:ID="BordeauxRegion">
>   <locatedIn rdf:resource="#FrenchRegion" />
> </Region>
>
> <owl:Class rdf:ID="Bordeaux">
>   <owl:intersectionOf rdf:parseType="Collection">
```

```
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#BordeauxRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="Beaujolais">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#Red" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:hasValue rdf:resource="#Light" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#Dry" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:hasValue rdf:resource="#Delicate" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:hasValue rdf:resource="#GamayGrape" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#madeFromGrape" />
>       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#BeaujolaisRegion" />
```

```
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <Region rdf:ID="AustralianRegion" />
>
> <owl:Class rdf:ID="Anjou">
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasColor" />
>       <owl:hasValue rdf:resource="#Rose" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasBody" />
>       <owl:hasValue rdf:resource="#Light" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasFlavor" />
>       <owl:hasValue rdf:resource="#Delicate" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <rdfs:subClassOf>
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#hasSugar" />
>       <owl:hasValue rdf:resource="#OffDry" />
>     </owl:Restriction>
>   </rdfs:subClassOf>
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Loire" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#AnjouRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
> <owl:Class rdf:ID="AmericanWine">
>   <owl:intersectionOf rdf:parseType="Collection">
>     <owl:Class rdf:about="#Wine" />
>     <owl:Restriction>
>       <owl:onProperty rdf:resource="#locatedIn" />
>       <owl:hasValue rdf:resource="#USRegion" />
>     </owl:Restriction>
>   </owl:intersectionOf>
> </owl:Class>
>
```

```
>  <owl:Class rdf:ID="AlsatianWine">
>    <owl:intersectionOf rdf:parseType="Collection">
>      <owl:Class rdf:about="#Wine" />
>      <owl:Restriction>
>        <owl:onProperty rdf:resource="#locatedIn" />
>        <owl:hasValue rdf:resource="#AlsaceRegion" />
>      </owl:Restriction>
>    </owl:intersectionOf>
>  </owl:Class>
>
>  <WineBody rdf:ID="Full" />
>
>  <WineBody rdf:ID="Medium" />
>
>  <WineBody rdf:ID="Light" />
>
>  <WineColor rdf:ID="Red" />
>
>  <WineColor rdf:ID="Rose" />
>
>  <WineColor rdf:ID="White" />
>
>  <WineFlavor rdf:ID="Strong" />
>
>  <WineFlavor rdf:ID="Moderate" />
>
>  <WineFlavor rdf:ID="Delicate" />
>
>  <WineSugar rdf:ID="Dry" />
>
>  <WineSugar rdf:ID="OffDry">
>    <owl:differentFrom rdf:resource="#Dry"/>
>    <owl:differentFrom rdf:resource="#Sweet"/>
>  </WineSugar>
>
>  <WineSugar rdf:ID="Sweet">
>    <owl:differentFrom rdf:resource="#Dry"/>
>  </WineSugar>
>  <owl:AllDifferent>
>    <owl:distinctMembers rdf:parseType="Collection">
>      <vin:WineColor rdf:about="#Red" />
>      <vin:WineColor rdf:about="#White" />
>      <vin:WineColor rdf:about="#Rose" />
>    </owl:distinctMembers>
>  </owl:AllDifferent>
>  <owl:AllDifferent>
>    <owl:distinctMembers rdf:parseType="Collection">
>      <vin:WineBody rdf:about="#Light" />
>      <vin:WineBody rdf:about="#Medium" />
>      <vin:WineBody rdf:about="#Full" />
```

```
>   </owl:distinctMembers>
> </owl:AllDifferent>
> <owl:AllDifferent>
>   <owl:distinctMembers rdf:parseType="Collection">
>     <vin:WineFlavor rdf:about="#Delicate" />
>     <vin:WineFlavor rdf:about="#Moderate" />
>     <vin:WineFlavor rdf:about="#Strong" />
>   </owl:distinctMembers>
> </owl:AllDifferent>
> <owl:AllDifferent>
>   <owl:distinctMembers rdf:parseType="Collection">
>     <vin:WineSugar rdf:about="#Sweet" />
>     <vin:WineSugar rdf:about="#OffDry" />
>     <vin:WineSugar rdf:about="#Dry" />
>   </owl:distinctMembers>
> </owl:AllDifferent>
> <Region rdf:ID="AlsaceRegion">
>   <locatedIn rdf:resource="#FrenchRegion" />
> </Region>
> <Region rdf:ID="AnjouRegion">
>   <locatedIn rdf:resource="#LoireRegion" />
> </Region>
> <Region rdf:ID="ArroyoGrandeRegion">
>   <locatedIn rdf:resource="#CaliforniaRegion" />
> </Region>
> <Winery rdf:ID="Beringer" />
> <Winery rdf:ID="Bancroft" />
> <Chardonnay rdf:ID="BancroftChardonnay">
>   <locatedIn rdf:resource="#NapaRegion" />
>   <hasMaker  rdf:resource="#Bancroft" />
>   <hasSugar  rdf:resource="#Dry" />
>   <hasFlavor rdf:resource="#Moderate" />
>   <hasBody   rdf:resource="#Medium" />
> </Chardonnay>
> <Region rdf:ID="BeaujolaisRegion">
>   <locatedIn rdf:resource="#FrenchRegion" />
> </Region>
> <WineGrape rdf:ID="CabernetFrancGrape" />
> <WineGrape rdf:ID="CabernetSauvignonGrape" />
> <Region rdf:ID="CentralCoastRegion">
>   <locatedIn rdf:resource="#CaliforniaRegion" />
> </Region>
> <WineGrape rdf:ID="ChardonnayGrape" />
> <Winery rdf:ID="ChateauChevalBlanc" />
> <StEmilion rdf:ID="ChateauChevalBlancStEmilion">
>   <hasMaker  rdf:resource="#ChateauChevalBlanc" />
> </StEmilion>
> <Winery rdf:ID="ChateauDYchem" />
> <Sauternes rdf:ID="ChateauDYchemSauterne">
>   <madeFromGrape rdf:resource="#SauvignonBlancGrape" />
```

```
>    <madeFromGrape rdf:resource="#SemillonGrape" />
>    <hasFlavor rdf:resource="#Strong" />
>    <hasMaker  rdf:resource="#ChateauDYchem" />
> </Sauternes>
> <Winery rdf:ID="ChateauDeMeursault" />
> <Meursault rdf:ID="ChateauDeMeursaultMeursault">
>    <hasFlavor rdf:resource="#Moderate" />
>    <hasMaker  rdf:resource="#ChateauDeMeursault" />
> </Meursault>
> <Winery rdf:ID="ChateauLafiteRothschild" />
> <Pauillac rdf:ID="ChateauLafiteRothschildPauillac">
>    <hasMaker  rdf:resource="#ChateauLafiteRothschild" />
> </Pauillac>
> <Margaux rdf:ID="ChateauMargaux">
>    <hasMaker  rdf:resource="#ChateauMargauxWinery" />
> </Margaux>
> <Winery rdf:ID="ChateauMargauxWinery" />
> <Winery rdf:ID="ChateauMorgon" />
> <Beaujolais rdf:ID="ChateauMorgonBeaujolais">
>    <hasMaker  rdf:resource="#ChateauMorgon" />
> </Beaujolais>
> <WineGrape rdf:ID="CheninBlancGrape" />
> <WineGrape rdf:ID="ZinfandelGrape" />
> <Chianti rdf:ID="ChiantiClassico">
>    <hasBody   rdf:resource="#Medium" />
>    <hasMaker  rdf:resource="#McGuinnesso" />
> </Chianti>
> <Region rdf:ID="ChiantiRegion">
>    <locatedIn rdf:resource="#ItalianRegion" />
> </Region>
> <Winery rdf:ID="ClosDeLaPoussie" />
> <Sancerre rdf:ID="ClosDeLaPoussieSancerre">
>    <hasMaker  rdf:resource="#ClosDeLaPoussie" />
> </Sancerre>
> <Winery rdf:ID="ClosDeVougeot" />
> <CotesDOr rdf:ID="ClosDeVougeotCotesDOr">
>    <hasMaker  rdf:resource="#ClosDeVougeot" />
> </CotesDOr>
> <Winery rdf:ID="CongressSprings" />
>
> <Semillon rdf:ID="CongressSpringsSemillon">
>    <hasMaker  rdf:resource="#CongressSprings" />
>    <hasSugar  rdf:resource="#Dry" />
>    <hasFlavor rdf:resource="#Moderate" />
>    <hasBody   rdf:resource="#Medium" />
> </Semillon>
> <Winery rdf:ID="Corbans" />
> <Riesling rdf:ID="CorbansDryWhiteRiesling">
>    <locatedIn rdf:resource="#NewZealandRegion" />
>    <hasMaker  rdf:resource="#Corbans" />
```

> <hasSugar rdf:resource="#OffDry" />
> <hasFlavor rdf:resource="#Moderate" />
> <hasBody rdf:resource="#Medium" />
> </Riesling>
> <SauvignonBlanc rdf:ID="CorbansPrivateBinSauvignonBlanc">
> <locatedIn rdf:resource="#NewZealandRegion" />
> <hasMaker rdf:resource="#Corbans" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Strong" />
> <hasBody rdf:resource="#Full" />
> </SauvignonBlanc>
> <SauvignonBlanc rdf:ID="CorbansSauvignonBlanc">
> <locatedIn rdf:resource="#NewZealandRegion" />
> <hasMaker rdf:resource="#Corbans" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Strong" />
> <hasBody rdf:resource="#Medium" />
> </SauvignonBlanc>
> <Winery rdf:ID="CortonMontrachet" />
> <WhiteBurgundy rdf:ID="CortonMontrachetWhiteBurgundy">
> <hasMaker rdf:resource="#CortonMontrachet" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Strong" />
> <hasBody rdf:resource="#Full" />
> </WhiteBurgundy>
> <Region rdf:ID="CotesDOrRegion">
> <locatedIn rdf:resource="#BourgogneRegion" />
> </Region>
> <Winery rdf:ID="Cotturi" />
> <Zinfandel rdf:ID="CotturiZinfandel">
> <locatedIn rdf:resource="#SonomaRegion" />
> <hasMaker rdf:resource="#Cotturi" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Strong" />
> <hasBody rdf:resource="#Full" />
> </Zinfandel>
> <Winery rdf:ID="DAnjou" />
> <Region rdf:ID="EdnaValleyRegion">
> <locatedIn rdf:resource="#CaliforniaRegion" />
> </Region>
> <Winery rdf:ID="Elyse" />
> <Zinfandel rdf:ID="ElyseZinfandel">
> <locatedIn rdf:resource="#NapaRegion" />
> <hasMaker rdf:resource="#Elyse" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Moderate" />
> <hasBody rdf:resource="#Full" />
> </Zinfandel>
> <Winery rdf:ID="Forman" />
> <CabernetSauvignon rdf:ID="FormanCabernetSauvignon">

> <locatedIn rdf:resource="#NapaRegion" />
> <hasMaker rdf:resource="#Forman" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Strong" />
> <hasBody   rdf:resource="#Medium" />
> </CabernetSauvignon>
> <Chardonnay rdf:ID="FormanChardonnay">
> <locatedIn rdf:resource="#NapaRegion" />
> <hasMaker rdf:resource="#Forman" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Moderate" />
> <hasBody   rdf:resource="#Full" />
> </Chardonnay>
> <Winery rdf:ID="Foxen" />
> <CheninBlanc rdf:ID="FoxenCheninBlanc">
> <locatedIn rdf:resource="#SantaBarbaraRegion" />
> <hasMaker rdf:resource="#Foxen" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Moderate" />
> <hasBody   rdf:resource="#Full" />
> </CheninBlanc>
> <WineGrape rdf:ID="GamayGrape" />
> <Winery rdf:ID="GaryFarrell" />
> <Merlot rdf:ID="GaryFarrellMerlot">
> <locatedIn rdf:resource="#SonomaRegion" />
> <hasMaker rdf:resource="#GaryFarrell" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Moderate" />
> <hasBody   rdf:resource="#Medium" />
> </Merlot>
> <Region rdf:ID="GermanyRegion" />
> <Winery rdf:ID="Handley" />
> <Winery rdf:ID="KalinCellars" />
> <Semillon rdf:ID="KalinCellarsSemillon">
> <hasMaker rdf:resource="#KalinCellars" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Strong" />
> <hasBody   rdf:resource="#Full" />
> </Semillon>
> <Winery rdf:ID="KathrynKennedy" />
> <Meritage rdf:ID="KathrynKennedyLateral">
> <hasMaker rdf:resource="#KathrynKennedy" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Delicate" />
> <hasBody   rdf:resource="#Medium" />
> </Meritage>
> <Winery rdf:ID="LaneTanner" />
> <PinotNoir rdf:ID="LaneTannerPinotNoir">
> <locatedIn rdf:resource="#SantaBarbaraRegion" />
> <hasMaker rdf:resource="#LaneTanner" />

```
>    <hasSugar  rdf:resource="#Dry" />
>    <hasFlavor rdf:resource="#Delicate" />
>    <hasBody   rdf:resource="#Light" />
> </PinotNoir>
> <Winery rdf:ID="Longridge" />
> <Merlot rdf:ID="LongridgeMerlot">
>    <locatedIn rdf:resource="#NewZealandRegion" />
>    <hasMaker  rdf:resource="#Longridge" />
>    <hasSugar  rdf:resource="#Dry" />
>    <hasFlavor rdf:resource="#Moderate" />
>    <hasBody   rdf:resource="#Light" />
> </Merlot>
> <WineGrape rdf:ID="MalbecGrape" />
> <Region rdf:ID="MargauxRegion">
>    <locatedIn rdf:resource="#MedocRegion" />
> </Region>
> <Winery rdf:ID="Marietta" />
> <CabernetSauvignon rdf:ID="MariettaCabernetSauvignon">
>    <locatedIn rdf:resource="#SonomaRegion" />
>    <hasMaker  rdf:resource="#Marietta" />
>    <hasSugar  rdf:resource="#Dry" />
>    <hasFlavor rdf:resource="#Moderate" />
>    <hasBody   rdf:resource="#Medium" />
> </CabernetSauvignon>
> <RedTableWine rdf:ID="MariettaOldVinesRed">
>    <locatedIn rdf:resource="#SonomaRegion" />
>    <hasMaker  rdf:resource="#Marietta" />
>    <hasSugar  rdf:resource="#Dry" />
>    <hasFlavor rdf:resource="#Moderate" />
>    <hasBody   rdf:resource="#Medium" />
> </RedTableWine>
> <PetiteSyrah rdf:ID="MariettaPetiteSyrah">
>    <locatedIn rdf:resource="#SonomaRegion" />
>    <hasMaker  rdf:resource="#Marietta" />
>    <hasSugar  rdf:resource="#Dry" />
>    <hasFlavor rdf:resource="#Moderate" />
>    <hasBody   rdf:resource="#Medium" />
> </PetiteSyrah>
> <Zinfandel rdf:ID="MariettaZinfandel">
>    <locatedIn rdf:resource="#SonomaRegion" />
>    <hasMaker  rdf:resource="#Marietta" />
>    <hasSugar  rdf:resource="#Dry" />
>    <hasFlavor rdf:resource="#Moderate" />
>    <hasBody   rdf:resource="#Medium" />
> </Zinfandel>
> <Winery rdf:ID="McGuinnesso" />
> <Region rdf:ID="MendocinoRegion">
>    <locatedIn rdf:resource="#CaliforniaRegion" />
>    <adjacentRegion rdf:resource="#SonomaRegion" />
> </Region>
```

```
> <WineGrape rdf:ID="MerlotGrape" />
> <Region rdf:ID="MeursaultRegion">
>   <locatedIn rdf:resource="#BourgogneRegion" />
> </Region>
> <Winery rdf:ID="MountEdenVineyard" />
> <Chardonnay rdf:ID="MountEdenVineyardEdnaValleyChardonnay">
>   <locatedIn rdf:resource="#EdnaValleyRegion" />
>   <hasMaker  rdf:resource="#MountEdenVineyard" />
>   <hasSugar  rdf:resource="#Dry" />
>   <hasFlavor rdf:resource="#Moderate" />
>   <hasBody   rdf:resource="#Medium" />
> </Chardonnay>
> <PinotNoir rdf:ID="MountEdenVineyardEstatePinotNoir">
>   <locatedIn rdf:resource="#EdnaValleyRegion" />
>   <hasMaker  rdf:resource="#MountEdenVineyard" />
>   <hasSugar  rdf:resource="#Dry" />
>   <hasFlavor rdf:resource="#Strong" />
>   <hasBody   rdf:resource="#Full" />
> </PinotNoir>
> <Winery rdf:ID="Mountadam" />
> <Chardonnay rdf:ID="MountadamChardonnay">
>   <locatedIn rdf:resource="#SouthAustraliaRegion" />
>   <hasMaker  rdf:resource="#Mountadam" />
>   <hasSugar  rdf:resource="#Dry" />
>   <hasFlavor rdf:resource="#Strong" />
>   <hasBody   rdf:resource="#Full" />
> </Chardonnay>
> <PinotNoir rdf:ID="MountadamPinotNoir">
>   <locatedIn rdf:resource="#SouthAustraliaRegion" />
>   <hasMaker  rdf:resource="#Mountadam" />
>   <hasSugar  rdf:resource="#Dry" />
>   <hasFlavor rdf:resource="#Moderate" />
>   <hasBody   rdf:resource="#Medium" />
> </PinotNoir>
> <DryRiesling rdf:ID="MountadamRiesling">
>   <locatedIn rdf:resource="#SouthAustraliaRegion" />
>   <hasMaker  rdf:resource="#Mountadam" />
>   <hasSugar  rdf:resource="#Dry" />
>   <hasFlavor rdf:resource="#Delicate" />
>   <hasBody   rdf:resource="#Medium" />
> </DryRiesling>
> <Region rdf:ID="MuscadetRegion">
>   <locatedIn rdf:resource="#LoireRegion" />
> </Region>
> <Region rdf:ID="NapaRegion">
>   <locatedIn rdf:resource="#CaliforniaRegion" />
> </Region>
> <Region rdf:ID="NewZealandRegion" />
> <Winery rdf:ID="PageMillWinery" />
> <CabernetSauvignon rdf:ID="PageMillWineryCabernetSauvignon">
```

> <locatedIn rdf:resource="#NapaRegion" />
> <hasMaker rdf:resource="#PageMillWinery" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Moderate" />
> <hasBody   rdf:resource="#Medium" />
> </CabernetSauvignon>
> <Region rdf:ID="PauillacRegion">
> <locatedIn rdf:resource="#MedocRegion" />
> </Region>
> <Winery rdf:ID="PeterMccoy" />
> <Chardonnay rdf:ID="PeterMccoyChardonnay">
> <locatedIn rdf:resource="#SonomaRegion" />
> <hasMaker rdf:resource="#PeterMccoy" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Moderate" />
> <hasBody   rdf:resource="#Medium" />
> </Chardonnay>
> <WineGrape rdf:ID="PetiteSyrahGrape" />
> <WineGrape rdf:ID="PetiteVerdotGrape" />
> <WineGrape rdf:ID="PinotBlancGrape" />
> <WineGrape rdf:ID="PinotNoirGrape" />
> <Region rdf:ID="PortugalRegion" />
> <Winery rdf:ID="PulignyMontrachet" />
> <WhiteBurgundy rdf:ID="PulignyMontrachetWhiteBurgundy">
> <hasMaker rdf:resource="#PulignyMontrachet" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Moderate" />
> <hasBody   rdf:resource="#Medium" />
> </WhiteBurgundy>
> <WineGrape rdf:ID="RieslingGrape" />
> <Anjou rdf:ID="RoseDAnjou">
> <hasMaker rdf:resource="#DAnjou" />
> </Anjou>
> <Region rdf:ID="SancerreRegion">
> <locatedIn rdf:resource="#LoireRegion" />
> </Region>
> <WineGrape rdf:ID="SangioveseGrape" />
> <Region rdf:ID="SantaBarbaraRegion">
> <locatedIn rdf:resource="#CaliforniaRegion" />
> </Region>
> <Winery rdf:ID="SantaCruzMountainVineyard" />
> <CabernetSauvignon rdf:ID="SantaCruzMountainVineyardCabernetSauvignon">
> <locatedIn rdf:resource="#SantaCruzMountainsRegion" />
> <hasMaker rdf:resource="#SantaCruzMountainVineyard" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Strong" />
> <hasBody   rdf:resource="#Full" />
> </CabernetSauvignon>
> <Region rdf:ID="CentralTexasRegion">
> <locatedIn rdf:resource="#TexasRegion" />

```
> </Region>
> <Winery rdf:ID="StGenevieve" />
> <WhiteWine rdf:ID="StGenevieveTexasWhite">
>   <locatedIn rdf:resource="#CentralTexasRegion" />
>   <hasMaker  rdf:resource="#StGenevieve" />
>   <hasSugar  rdf:resource="#Dry" />
>   <hasFlavor rdf:resource="#Moderate" />
> </WhiteWine>
> <Region rdf:ID="SantaCruzMountainsRegion">
>   <locatedIn rdf:resource="#CaliforniaRegion" />
> </Region>
> <Winery rdf:ID="SaucelitoCanyon" />
> <Zinfandel rdf:ID="SaucelitoCanyonZinfandel">
>   <locatedIn rdf:resource="#ArroyoGrandeRegion" />
>   <hasMaker  rdf:resource="#SaucelitoCanyon" />
>   <hasSugar  rdf:resource="#Dry" />
>   <hasFlavor rdf:resource="#Moderate" />
>   <hasBody   rdf:resource="#Medium" />
> </Zinfandel>
> <Zinfandel rdf:ID="SaucelitoCanyonZinfandel1998">
>   <locatedIn rdf:resource="#ArroyoGrandeRegion" />
>   <hasVintageYear rdf:resource="#Year1998" />
>   <hasMaker  rdf:resource="#SaucelitoCanyon" />
>   <hasSugar  rdf:resource="#Dry" />
>   <hasFlavor rdf:resource="#Moderate" />
>   <hasBody   rdf:resource="#Medium" />
> </Zinfandel>
> <Region rdf:ID="SauterneRegion">
>   <locatedIn rdf:resource="#BordeauxRegion" />
> </Region>
> <WineGrape rdf:ID="SauvignonBlancGrape" />
> <Winery rdf:ID="SchlossRothermel" />
> <SweetRiesling rdf:ID="SchlossRothermelTrochenbierenausleseRiesling">
>   <locatedIn rdf:resource="#GermanyRegion" />
>   <hasMaker  rdf:resource="#SchlossRothermel" />
>   <hasSugar  rdf:resource="#Sweet" />
>   <hasFlavor rdf:resource="#Strong" />
>   <hasBody   rdf:resource="#Full" />
> </SweetRiesling>
> <Winery rdf:ID="SchlossVolrad" />
> <SweetRiesling rdf:ID="SchlossVolradTrochenbierenausleseRiesling">
>   <locatedIn rdf:resource ="#GermanyRegion" />
>   <hasMaker  rdf:resource="#SchlossVolrad" />
>   <hasSugar  rdf:resource="#Sweet" />
>   <hasFlavor rdf:resource="#Moderate" />
>   <hasBody   rdf:resource="#Full" />
> </SweetRiesling>
> <Winery rdf:ID="SeanThackrey" />
> <PetiteSyrah rdf:ID="SeanThackreySiriusPetiteSyrah">
>   <locatedIn rdf:resource="#NapaRegion" />
```

> <hasMaker rdf:resource="#SeanThackrey" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Strong" />
> <hasBody   rdf:resource="#Full" />
> </PetiteSyrah>
> <Winery rdf:ID="Selaks" />
> <IceWine rdf:ID="SelaksIceWine">
> <locatedIn rdf:resource="#NewZealandRegion" />
> <hasMaker rdf:resource="#Selaks" />
> <hasFlavor rdf:resource="#Moderate" />
> <hasBody   rdf:resource="#Medium" />
> <hasColor rdf:resource="#White" />
> </IceWine>
> <SauvignonBlanc rdf:ID="SelaksSauvignonBlanc">
> <locatedIn rdf:resource="#NewZealandRegion" />
> <hasMaker rdf:resource="#Selaks" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Moderate" />
> <hasBody   rdf:resource="#Medium" />
> </SauvignonBlanc>
> <WineGrape rdf:ID="SemillonGrape" />
> <Winery rdf:ID="SevreEtMaine" />
> <Muscadet rdf:ID="SevreEtMaineMuscadet">
> <hasMaker rdf:resource="#SevreEtMaine" />
> </Muscadet>
> <Region rdf:ID="SonomaRegion">
> <locatedIn rdf:resource="#CaliforniaRegion" />
> </Region>
> <Region rdf:ID="SouthAustraliaRegion">
> <locatedIn rdf:resource="#AustralianRegion" />
> </Region>
> <Region rdf:ID="StEmilionRegion">
> <locatedIn rdf:resource="#BordeauxRegion" />
> </Region>
> <Winery rdf:ID="Stonleigh" />
> <SauvignonBlanc rdf:ID="StonleighSauvignonBlanc">
> <locatedIn rdf:resource="#NewZealandRegion" />
> <hasMaker rdf:resource="#Stonleigh" />
> <hasSugar rdf:resource="#Dry" />
> <hasFlavor rdf:resource="#Delicate" />
> <hasBody   rdf:resource="#Medium" />
> </SauvignonBlanc>
> <Winery rdf:ID="Taylor" />
> <Port rdf:ID="TaylorPort">
> <hasMaker rdf:resource="#Taylor" />
> </Port>
> <Region rdf:ID="ToursRegion">
> <locatedIn rdf:resource="#LoireRegion" />
> </Region>
> <Winery rdf:ID="Ventana" />

```
> <CheninBlanc rdf:ID="VentanaCheninBlanc">
>   <locatedIn rdf:resource="#CentralCoastRegion" />
>   <hasMaker  rdf:resource="#Ventana" />
>   <hasSugar  rdf:resource="#OffDry" />
>   <hasFlavor rdf:resource="#Moderate" />
>   <hasBody   rdf:resource="#Medium" />
> </CheninBlanc>
> <Winery rdf:ID="WhitehallLane" />
> <CabernetFranc rdf:ID="WhitehallLaneCabernetFranc">
>   <locatedIn rdf:resource="#NapaRegion" />
>   <hasMaker  rdf:resource="#WhitehallLane" />
>   <hasSugar  rdf:resource="#Dry" />
>   <hasFlavor rdf:resource="#Moderate" />
>   <hasBody   rdf:resource="#Medium" />
> </CabernetFranc>
> <DessertWine rdf:ID="WhitehallLanePrimavera">
>   <locatedIn rdf:resource="#NapaRegion" />
>   <hasSugar  rdf:resource="#Sweet" />
>   <hasFlavor rdf:resource="#Delicate" />
>   <hasBody   rdf:resource="#Light" />
> </DessertWine>
> <owl:AllDifferent>
>   <owl:distinctMembers rdf:parseType="Collection">
>     <vin:Winery rdf:about="#Bancroft" />
>     <vin:Winery rdf:about="#ChateauChevalBlanc" />
>     <vin:Winery rdf:about="#ChateauDYchem" />
>     <vin:Winery rdf:about="#ChateauDeMeursault" />
>     <vin:Winery rdf:about="#ChateauLafiteRothschild" />
>     <vin:Winery rdf:about="#ChateauMargauxWinery" />
>     <vin:Winery rdf:about="#ChateauMorgon" />
>     <vin:Winery rdf:about="#ClosDeLaPoussie" />
>     <vin:Winery rdf:about="#ClosDeVougeot" />
>     <vin:Winery rdf:about="#CongressSprings" />
>     <vin:Winery rdf:about="#Corbans" />
>     <vin:Winery rdf:about="#CortonMontrachet" />
>     <vin:Winery rdf:about="#Cotturi" />
>     <vin:Winery rdf:about="#DAnjou" />
>     <vin:Winery rdf:about="#Elyse" />
>     <vin:Winery rdf:about="#Forman" />
>     <vin:Winery rdf:about="#Foxen" />
>     <vin:Winery rdf:about="#GaryFarrell" />
>     <vin:Winery rdf:about="#KalinCellars" />
>     <vin:Winery rdf:about="#KathrynKennedy" />
>     <vin:Winery rdf:about="#LaneTanner" />
>     <vin:Winery rdf:about="#Longridge" />
>     <vin:Winery rdf:about="#Marietta" />
>     <vin:Winery rdf:about="#McGuinnesso" />
>     <vin:Winery rdf:about="#MountEdenVineyard" />
>     <vin:Winery rdf:about="#Mountadam" />
>     <vin:Winery rdf:about="#PageMillWinery" />
```

```
>     <vin:Winery rdf:about="#PeterMccoy" />
>     <vin:Winery rdf:about="#PulignyMontrachet" />
>     <vin:Winery rdf:about="#SantaCruzMountainVineyard" />
>     <vin:Winery rdf:about="#SaucelitoCanyon" />
>     <vin:Winery rdf:about="#SchlossRothermel" />
>     <vin:Winery rdf:about="#SchlossVolrad" />
>     <vin:Winery rdf:about="#SeanThackrey" />
>     <vin:Winery rdf:about="#Selaks" />
>     <vin:Winery rdf:about="#SevreEtMaine" />
>     <vin:Winery rdf:about="#StGenevieve" />
>     <vin:Winery rdf:about="#Stonleigh" />
>     <vin:Winery rdf:about="#Taylor" />
>     <vin:Winery rdf:about="#Ventana" />
>     <vin:Winery rdf:about="#WhitehallLane" />
>   </owl:distinctMembers>
>  </owl:AllDifferent>
></rdf:RDF>
```

Travel Ontology

Namespace: http://learn.tsinghua.edu.cn/homepage/2003214945/travelontology.owl

Description: Ontology for the domain of traveling with emphasis in the hotels' description.

Location: http://learn.tsinghua.edu.cn/homepage/2003214945/travelontology.owl

```
<?xml version="1.0" ?>
- <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns="http://www.owl-ontologies.com/travel.owl#"
    xml:base="http://www.owl-ontologies.com/travel.owl">
- <owl:Ontology rdf:about="">
  <owl:versionInfo rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.0 by Holger Knublauch
    (holger@smi.stanford.edu)</owl:versionInfo>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">An example ontology for tutorial
    purposes.</rdfs:comment>
    </owl:Ontology>
- <owl:Class rdf:ID="Sunbathing">
- <rdfs:subClassOf>
  <owl:Class rdf:about="#Relaxation" />
    </rdfs:subClassOf>
    </owl:Class>
- <owl:Class rdf:ID="Accommodation">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A place to stay for tourists.</rdfs:comment>
    </owl:Class>
- <owl:Class rdf:ID="QuietDestination">
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:ID="Destination" />
- <owl:Class>
- <owl:complementOf>
  <owl:Class rdf:about="#FamilyDestination" />
    </owl:complementOf>
    </owl:Class>
    </owl:intersectionOf>
    </owl:Class>
    </owl:equivalentClass>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A destination that is not frequented by noisy
    families.</rdfs:comment>
    </owl:Class>
- <owl:Class rdf:ID="BackpackersDestination">
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Destination" />
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasAccommodation" />
    </owl:onProperty>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#BudgetAccommodation" />
    </owl:someValuesFrom>
    </owl:Restriction>
- <owl:Restriction>
- <owl:someValuesFrom>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Sports" />
  <owl:Class rdf:about="#Adventure" />
```

```
        </owl:unionOf>
        </owl:Class>
        </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasActivity" />
        </owl:onProperty>
        </owl:Restriction>
        </owl:intersectionOf>
        </owl:Class>
        </owl:equivalentClass>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A destination that provides budget
        accommodation and offers sport or adventure activities.</rdfs:comment>
        </owl:Class>
- <owl:Class rdf:ID="Sports">
- <owl:disjointWith>
  <owl:Class rdf:about="#Adventure" />
        </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Relaxation" />
        </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Sightseeing" />
        </owl:disjointWith>
- <rdfs:subClassOf>
  <owl:Class rdf:ID="Activity" />
        </rdfs:subClassOf>
        </owl:Class>
- <owl:Class rdf:ID="Yoga">
- <rdfs:subClassOf>
  <owl:Class rdf:about="#Relaxation" />
        </rdfs:subClassOf>
        </owl:Class>
- <owl:Class rdf:ID="BudgetAccommodation">
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Accommodation" />
- <owl:Restriction>
- <owl:someValuesFrom>
- <owl:Class>
- <owl:oneOf rdf:parseType="Collection">
- <AccommodationRating rdf:ID="OneStarRating">
- <owl:differentFrom>
- <AccommodationRating rdf:ID="ThreeStarRating">
- <owl:differentFrom>
- <AccommodationRating rdf:ID="TwoStarRating">
  <owl:differentFrom rdf:resource="#OneStarRating" />
  <owl:differentFrom rdf:resource="#ThreeStarRating" />
        </AccommodationRating>
        </owl:differentFrom>
  <owl:differentFrom rdf:resource="#OneStarRating" />
        </AccommodationRating>
        </owl:differentFrom>
  <owl:differentFrom rdf:resource="#TwoStarRating" />
        </AccommodationRating>
  <AccommodationRating rdf:about="#TwoStarRating" />
        </owl:oneOf>
        </owl:Class>
        </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasRating" />
        </owl:onProperty>
        </owl:Restriction>
        </owl:intersectionOf>
```

```
        </owl:Class>
        </owl:equivalentClass>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Accommodation that has either one or two star
        rating.</rdfs:comment>
        </owl:Class>
- <owl:Class rdf:ID="LuxuryHotel">
- <rdfs:subClassOf>
    <owl:Class rdf:about="#Hotel" />
        </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
    <owl:hasValue rdf:resource="#ThreeStarRating" />
- <owl:onProperty>
    <owl:ObjectProperty rdf:about="#hasRating" />
        </owl:onProperty>
        </owl:Restriction>
        </rdfs:subClassOf>
        </owl:Class>
- <owl:Class rdf:ID="FamilyDestination">
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Destination" />
- <owl:Restriction>
- <owl:onProperty>
    <owl:ObjectProperty rdf:about="#hasAccommodation" />
        </owl:onProperty>
    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:minCardinality>
        </owl:Restriction>
- <owl:Restriction>
- <owl:onProperty>
    <owl:ObjectProperty rdf:about="#hasActivity" />
        </owl:onProperty>
    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">2</owl:minCardinality>
        </owl:Restriction>
        </owl:intersectionOf>
        </owl:Class>
        </owl:equivalentClass>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A destination with at least one accommodation
        and at least 2 activities.</rdfs:comment>
        </owl:Class>
- <owl:Class rdf:ID="Beach">
    <rdfs:subClassOf rdf:resource="#Destination" />
        </owl:Class>
- <owl:Class rdf:ID="Hotel">
- <owl:disjointWith>
    <owl:Class rdf:about="#BedAndBreakfast" />
        </owl:disjointWith>
- <owl:disjointWith>
    <owl:Class rdf:about="#Campground" />
        </owl:disjointWith>
    <rdfs:subClassOf rdf:resource="#Accommodation" />
        </owl:Class>
- <owl:Class rdf:ID="Museums">
- <rdfs:subClassOf>
    <owl:Class rdf:about="#Sightseeing" />
        </rdfs:subClassOf>
        </owl:Class>
- <owl:Class rdf:ID="BudgetHotelDestination">
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Destination" />
- <owl:Restriction>
```

```
- <owl:someValuesFrom>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#BudgetAccommodation" />
  <owl:Class rdf:about="#Hotel" />
      </owl:intersectionOf>
      </owl:Class>
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasAccommodation" />
      </owl:onProperty>
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A destination with a hotel that is also a budget
        accommodation.</rdfs:comment>
      </owl:Class>
- <owl:Class rdf:ID="City">
- <rdfs:subClassOf>
  <owl:Class rdf:about="#UrbanArea" />
      </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasAccommodation" />
      </owl:onProperty>
  <owl:someValuesFrom rdf:resource="#LuxuryHotel" />
      </owl:Restriction>
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:ID="AccommodationRating">
- <owl:equivalentClass>
- <owl:Class>
- <owl:oneOf rdf:parseType="Collection">
  <AccommodationRating rdf:about="#OneStarRating" />
  <AccommodationRating rdf:about="#TwoStarRating" />
  <AccommodationRating rdf:about="#ThreeStarRating" />
      </owl:oneOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Consists of exactly three
        individuals.</rdfs:comment>
      </owl:Class>
- <owl:Class rdf:ID="BedAndBreakfast">
  <owl:disjointWith rdf:resource="#Hotel" />
  <rdfs:subClassOf rdf:resource="#Accommodation" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Campground" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:ID="Campground">
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:hasValue rdf:resource="#OneStarRating" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasRating" />
      </owl:onProperty>
      </owl:Restriction>
      </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#BedAndBreakfast" />
  <rdfs:subClassOf rdf:resource="#Accommodation" />
  <owl:disjointWith rdf:resource="#Hotel" />
      </owl:Class>
```

```
- <owl:Class rdf:ID="Safari">
- <rdfs:subClassOf>
  <owl:Class rdf:about="#Adventure" />
      </rdfs:subClassOf>
- <rdfs:subClassOf>
  <owl:Class rdf:about="#Sightseeing" />
      </rdfs:subClassOf>
      </owl:Class>
- <owl:Class rdf:ID="RuralArea">
  <rdfs:subClassOf rdf:resource="#Destination" />
- <owl:disjointWith>
  <owl:Class rdf:about="#UrbanArea" />
      </owl:disjointWith>
      </owl:Class>
- <owl:Class rdf:ID="RetireeDestination">
- <owl:equivalentClass>
- <owl:Class>
- <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Destination" />
- <owl:Restriction>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasAccommodation" />
      </owl:onProperty>
- <owl:someValuesFrom>
- <owl:Restriction>
  <owl:hasValue rdf:resource="#ThreeStarRating" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasRating" />
      </owl:onProperty>
      </owl:Restriction>
      </owl:someValuesFrom>
      </owl:Restriction>
- <owl:Restriction>
- <owl:someValuesFrom>
  <owl:Class rdf:about="#Sightseeing" />
      </owl:someValuesFrom>
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasActivity" />
      </owl:onProperty>
      </owl:Restriction>
      </owl:intersectionOf>
      </owl:Class>
      </owl:equivalentClass>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A destination with at least one three star
      accommodation and sightseeing opportunities.</rdfs:comment>
      </owl:Class>
- <owl:Class rdf:ID="Relaxation">
  <owl:disjointWith rdf:resource="#Sports" />
- <owl:disjointWith>
  <owl:Class rdf:about="#Sightseeing" />
      </owl:disjointWith>
- <owl:disjointWith>
  <owl:Class rdf:about="#Adventure" />
      </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#Activity" />
      </owl:Class>
- <owl:Class rdf:ID="Capital">
- <rdfs:subClassOf>
- <owl:Restriction>
  <owl:someValuesFrom rdf:resource="#Museums" />
- <owl:onProperty>
  <owl:ObjectProperty rdf:about="#hasActivity" />
      </owl:onProperty>
      </owl:Restriction>
```

```
        </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#City" />
        </owl:Class>
- <owl:Class rdf:ID="Hiking">
    <rdfs:subClassOf rdf:resource="#Sports" />
        </owl:Class>
- <owl:Class rdf:ID="UrbanArea">
    <owl:disjointWith rdf:resource="#RuralArea" />
    <rdfs:subClassOf rdf:resource="#Destination" />
        </owl:Class>
- <owl:Class rdf:ID="BunjeeJumping">
- <rdfs:subClassOf>
    <owl:Class rdf:about="#Adventure" />
        </rdfs:subClassOf>
        </owl:Class>
- <owl:Class rdf:ID="Adventure">
    <owl:disjointWith rdf:resource="#Sports" />
    <rdfs:subClassOf rdf:resource="#Activity" />
- <owl:disjointWith>
    <owl:Class rdf:about="#Sightseeing" />
        </owl:disjointWith>
    <owl:disjointWith rdf:resource="#Relaxation" />
        </owl:Class>
    <owl:Class rdf:ID="Contact" />
- <owl:Class rdf:ID="NationalPark">
    <rdfs:subClassOf rdf:resource="#RuralArea" />
- <rdfs:subClassOf>
- <owl:Restriction>
    <owl:someValuesFrom rdf:resource="#Campground" />
- <owl:onProperty>
    <owl:ObjectProperty rdf:about="#hasAccommodation" />
        </owl:onProperty>
        </owl:Restriction>
        </rdfs:subClassOf>
- <rdfs:subClassOf>
- <owl:Restriction>
    <owl:someValuesFrom rdf:resource="#Hiking" />
- <owl:onProperty>
    <owl:ObjectProperty rdf:about="#hasActivity" />
        </owl:onProperty>
        </owl:Restriction>
        </rdfs:subClassOf>
        </owl:Class>
- <owl:Class rdf:ID="Town">
    <rdfs:subClassOf rdf:resource="#UrbanArea" />
        </owl:Class>
- <owl:Class rdf:ID="Sightseeing">
    <owl:disjointWith rdf:resource="#Sports" />
    <rdfs:subClassOf rdf:resource="#Activity" />
    <owl:disjointWith rdf:resource="#Relaxation" />
    <owl:disjointWith rdf:resource="#Adventure" />
        </owl:Class>
- <owl:Class rdf:ID="Farmland">
    <rdfs:subClassOf rdf:resource="#RuralArea" />
        </owl:Class>
- <owl:Class rdf:ID="Surfing">
    <rdfs:subClassOf rdf:resource="#Sports" />
        </owl:Class>
- <owl:ObjectProperty rdf:ID="isOfferedAt">
    <rdfs:range rdf:resource="#Destination" />
    <rdfs:domain rdf:resource="#Activity" />
- <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#hasActivity" />
        </owl:inverseOf>
```

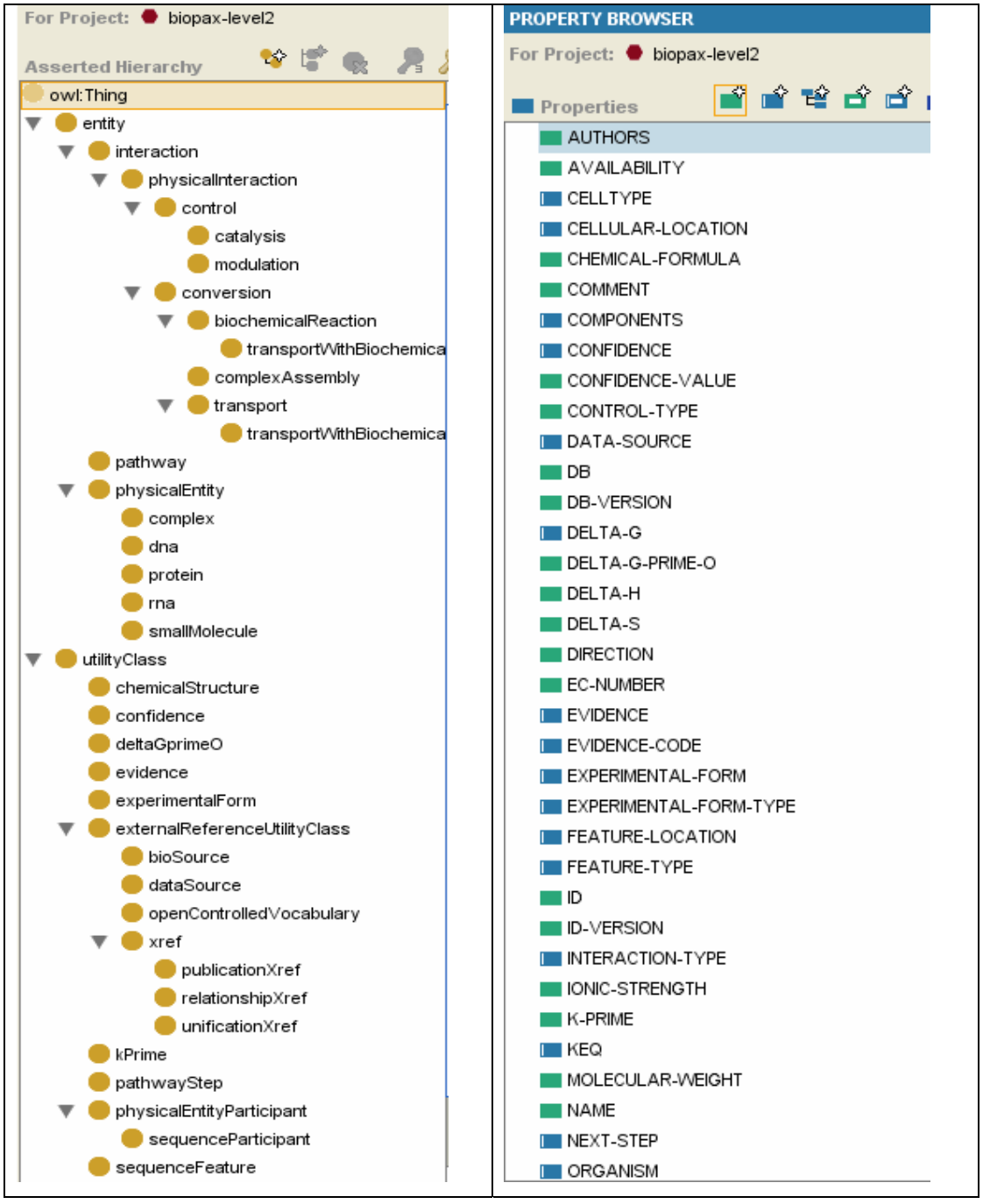
```
    </owl:ObjectProperty>
- <owl:ObjectProperty rdf:ID="hasRating">
  <rdfs:range rdf:resource="#AccommodationRating" />
  <rdfs:domain rdf:resource="#Accommodation" />
    </owl:ObjectProperty>
- <owl:ObjectProperty rdf:ID="hasActivity">
  <owl:inverseOf rdf:resource="#isOfferedAt" />
  <rdfs:range rdf:resource="#Activity" />
  <rdfs:domain rdf:resource="#Destination" />
    </owl:ObjectProperty>
- <owl:ObjectProperty rdf:ID="hasContact">
  <rdfs:range rdf:resource="#Contact" />
  <rdfs:domain rdf:resource="#Activity" />
    </owl:ObjectProperty>
- <owl:ObjectProperty rdf:ID="hasAccommodation">
  <rdfs:range rdf:resource="#Accommodation" />
  <rdfs:domain rdf:resource="#Destination" />
    </owl:ObjectProperty>
- <owl:ObjectProperty rdf:ID="hasPart">
  <rdfs:range rdf:resource="#Destination" />
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty" />
  <rdfs:domain rdf:resource="#Destination" />
    </owl:ObjectProperty>
- <owl:DatatypeProperty rdf:ID="hasCity">
  <rdfs:domain rdf:resource="#Contact" />
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty" />
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
    </owl:DatatypeProperty>
- <owl:DatatypeProperty rdf:ID="hasEMail">
  <rdfs:domain rdf:resource="#Contact" />
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty" />
    </owl:DatatypeProperty>
- <owl:FunctionalProperty rdf:ID="hasZipCode">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int" />
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty" />
  <rdfs:domain rdf:resource="#Contact" />
    </owl:FunctionalProperty>
- <owl:FunctionalProperty rdf:ID="hasStreet">
  <rdfs:domain rdf:resource="#Contact" />
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty" />
    </owl:FunctionalProperty>
  <RuralArea rdf:ID="Woomera" />
  <Town rdf:ID="Coonabarabran" />
  <LuxuryHotel rdf:ID="FourSeasons" />
  <NationalPark rdf:ID="BlueMountains" />
  <Capital rdf:ID="Canberra" />
  <Beach rdf:ID="BondiBeach" />
  <Beach rdf:ID="CurrawongBeach" />
  <NationalPark rdf:ID="Warrumbungles" />
  <RuralArea rdf:ID="CapeYork" />
- <Capital rdf:ID="Sydney">
  <hasAccommodation rdf:resource="#FourSeasons" />
  <hasPart rdf:resource="#BondiBeach" />
  <hasPart rdf:resource="#CurrawongBeach" />
    </Capital>
  <City rdf:ID="Cairns" />
    </rdf:RDF>
```

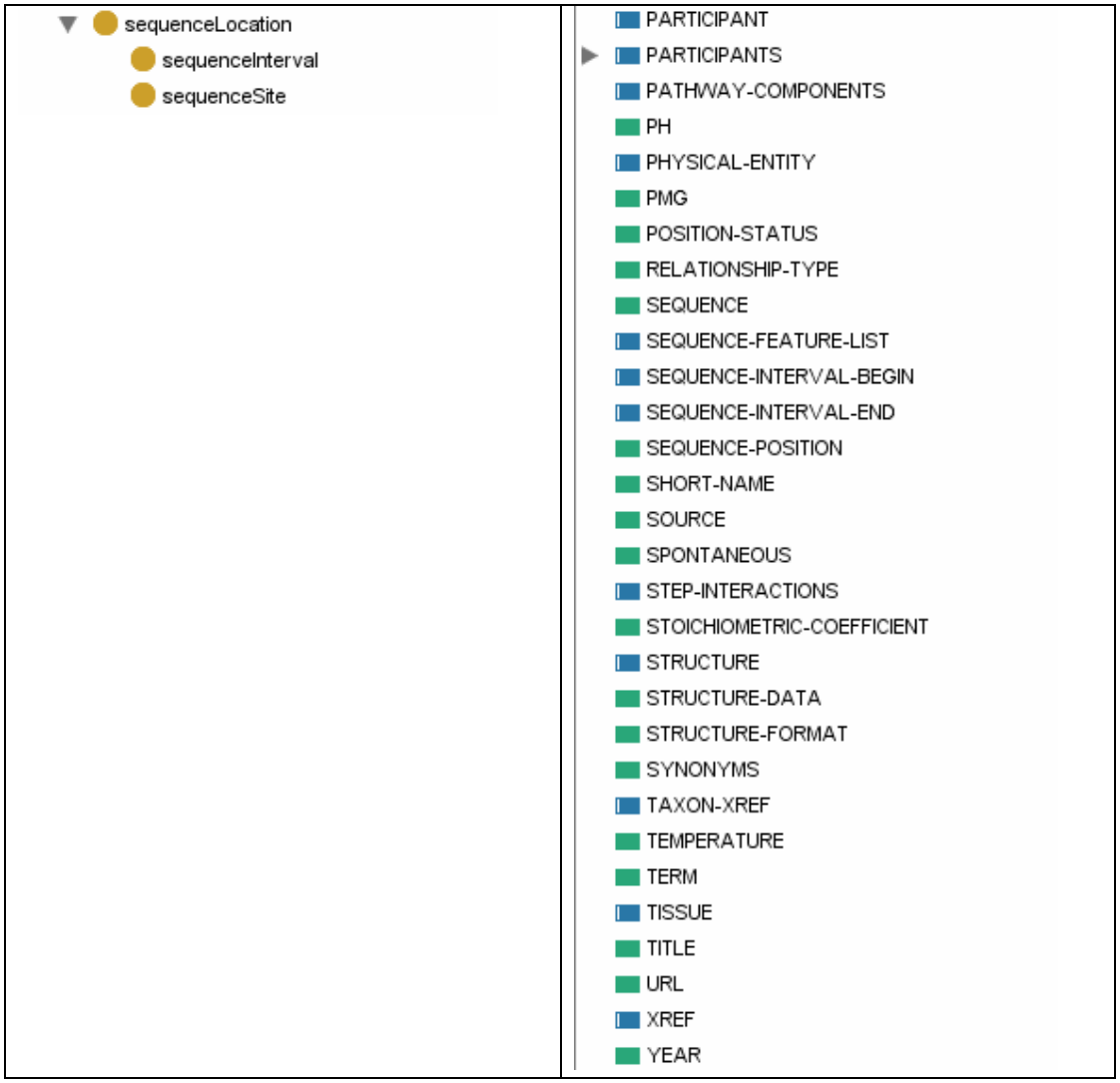
Biopax-Level 2 Ontology

Namespace: http://www.biopax.org/release/biopax-level2.owl

Description: BioPAX Level 2 covers metabolic pathways, molecular interactions and protein post-translational modifications and is backwards compatible with Level 1. Future levels will expand support for signaling pathways, gene regulatory networks and genetic interactions.

Location: http://www.biopax.org/release/biopax-level2.owl




sequenceLocation
sequenceInterval
sequenceSite
PARTICIPANT
PARTICIPANTS
PATHWAY-COMPONENTS
PH
PHYSICAL-ENTITY
PMG
POSITION-STATUS
RELATIONSHIP-TYPE
SEQUENCE
SEQUENCE-FEATURE-LIST
SEQUENCE-INTERVAL-BEGIN
SEQUENCE-INTERVAL-END
SEQUENCE-POSITION
SHORT-NAME
SOURCE
SPONTANEOUS
STEP-INTERACTIONS
STOICHIOMETRIC-COEFFICIENT
STRUCTURE
STRUCTURE-DATA
STRUCTURE-FORMAT
SYNONYMS
TAXON-XREF
TEMPERATURE
TERM
TISSUE
TITLE
URL
XREF
YEAR

Soccer Ontology

The Soccer Ontology is a domain ontology that is used for soccer in the context of OntoNL. The ontology extends the Upper Ontology capturing the MPEG-7 MDS. Thus, the soccer ontology classes are distinguished into agents (including persons, organizations and person groups), events, times, places, objects and states.

Namespace: http://lamia.ced.tuc.gr/ontologies/avmds03/soccer

Description: Ontology for the domain of traveling with emphasis in the hotels' description.

Location: http://lamia.ced.tuc.gr

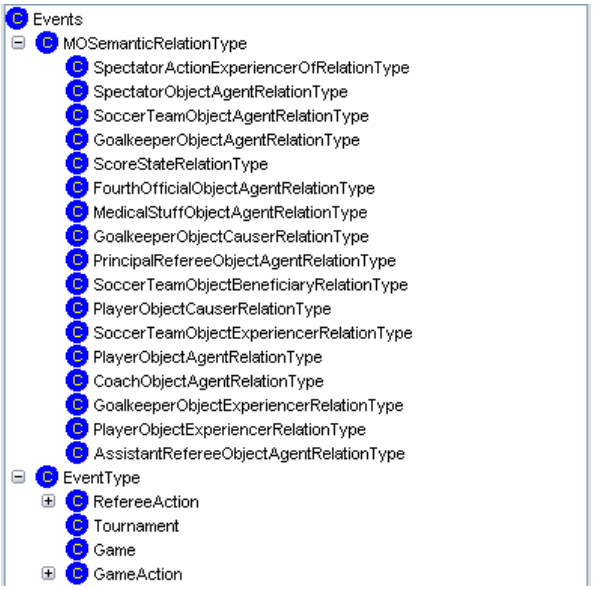


Figure 52: The Hierarchy of the Soccer Event Classes

Figure 53: The Hierarchy of the Referee Action Classes

Figure 54: The Hierarchy of the Game Action Classes, where Technical Stuff Actions and Spectator Actions are expanded

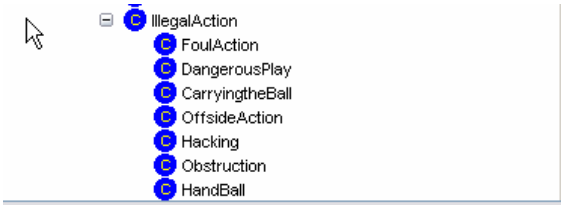


Figure 55: The Hierarchy of the Game Action Classes, where Illegal Actions are expanded

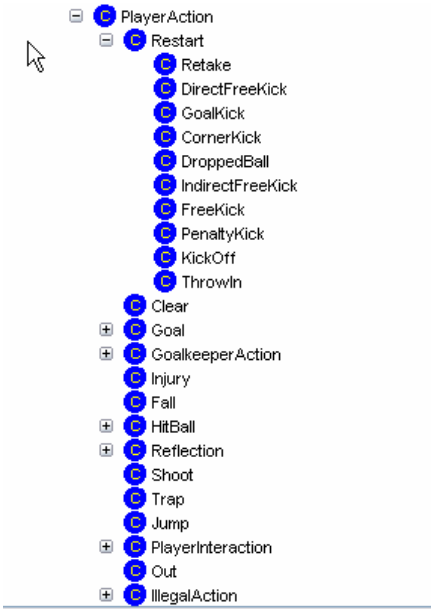


Figure 56: The Hierarchy of the Player Action Classes, where Restart Actions are expanded

Figure 57: The Hierarchy of the Player Action Classes, where Goal and Goalkeeper Actions are expanded

Figure 58: The Hierarchy of the Player Action Classes, where Hitball and Reflection Actions are expanded

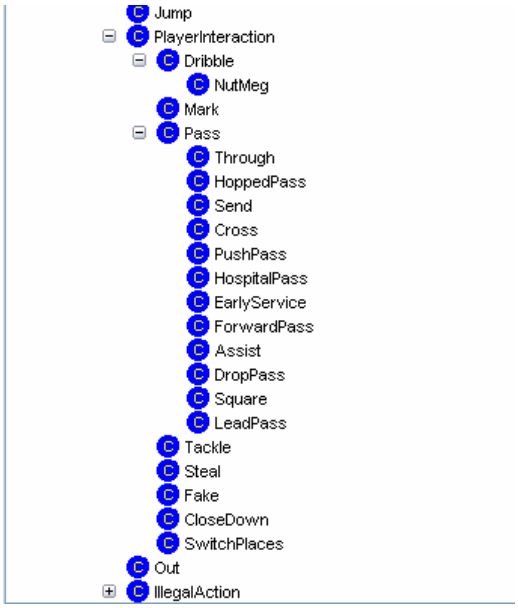


Figure 59: The Hierarchy of the Player Action Classes, where PlayerInteractions are expanded

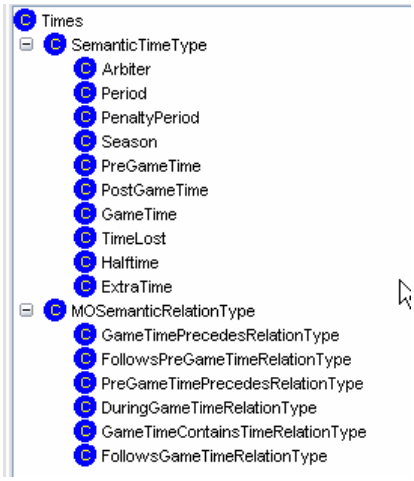


Figure 60: The Hierarchy of the Soccer Time Classes

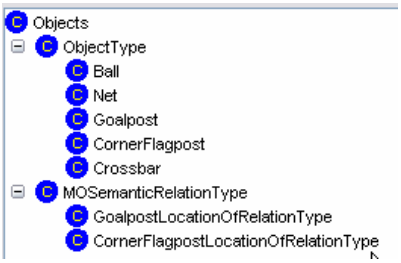


Figure 61: The Hierarchy of the Soccer Object Classes

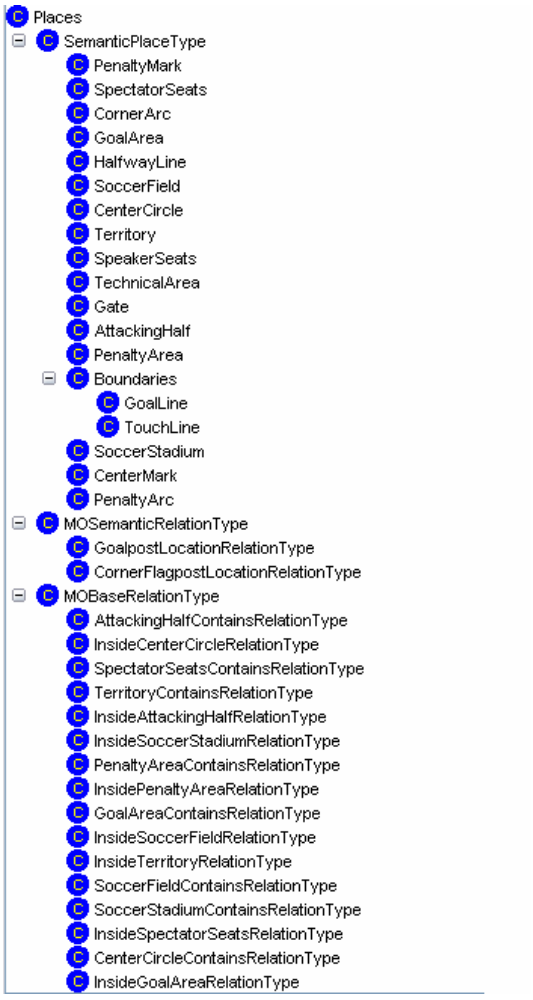


Figure 62: The Hierarchy of the Soccer Place Classes

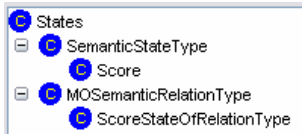


Figure 63: The Hierarchy of the Soccer State Classes

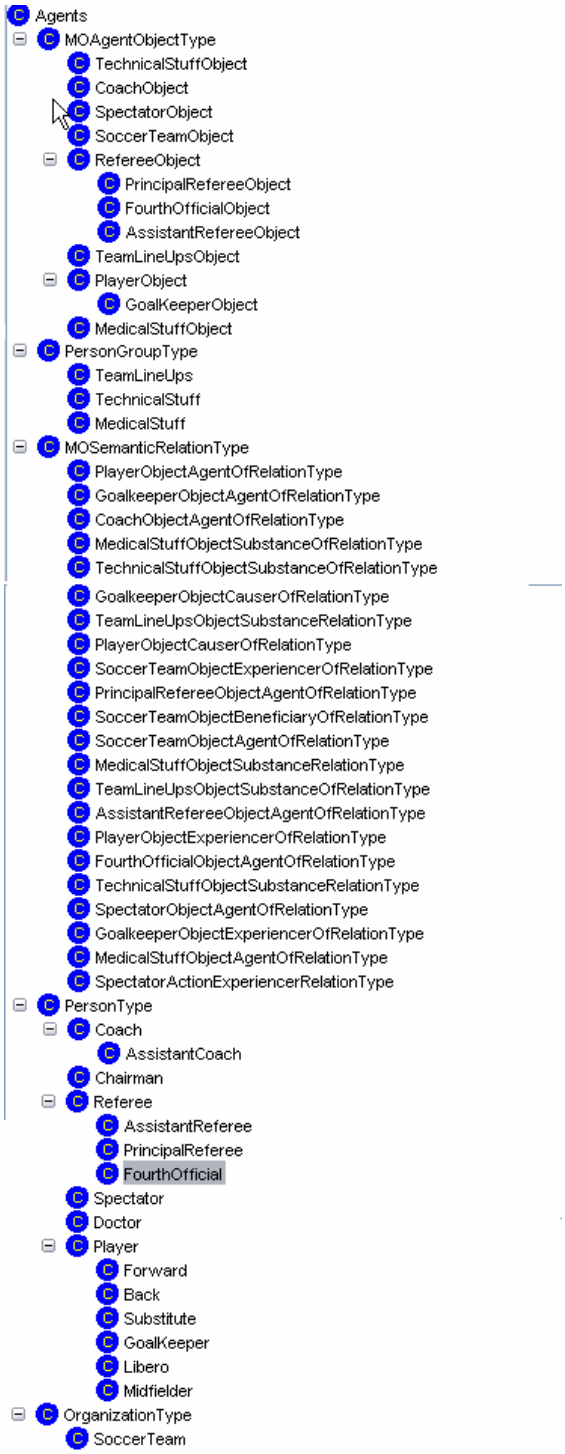


Figure 64: The Hierarchy of the Soccer Agent Classes

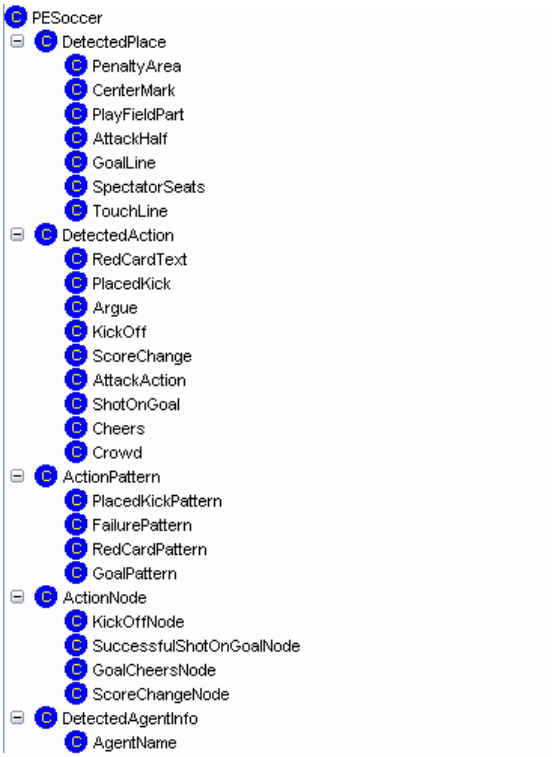


Figure 65: The Hierarchy of the Soccer Action Pattern Classes