UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Facoltà di Ingegneria–Sede di Modena

Tesi di Laurea in Ingegneria Informatica

Intelligent Information Integration systems: extending lexicon ontology

---

Sistemi di Integrazione dell'Informazione: estensione dell'ontologia lessicale

Relatore:
Chiar.mo Prof. Sonia Bergamaschi

Tesi di:
Veronica Guidetti

Correlatore:
Dott. Ing. Alberto Corni

Anno Accademico 2001 – 2002

Key words:
Intelligent integration
MOMIS
Ontology
Alignment
Similarity
Search
WordNet

Acknowledgements

I would like to thank Prof. Sonia Bergamaschi for her constant support during this research, but most of all for having trusted me. I had also the pleasure of meeting Dott.Ing. Alberto Corni and I would like to thank him for many stimulating discussions and, most of all, for being an excellent supervisor. I wish to thank Dott.ssa Federica Mandreoli for introducing me to the huge and fascinating area of information retrieval and for all the initial suggestions.

I am very grateful to my parents for their patience and love*, to Agata for her full support, to Pir for the needed liveliness, and to Ludwig Van Beethoven.*

Ciò che nel passato è probabile,
nel futuro è possibile

A. Einstein

Contents

List of Figures

List of Tables

Abstract

Taking into consideration how different cultures are far characterized but also the great number of shared key elements, it is clear how everyone could support and consequently extend a concept in a proper and particular way. This work is in the information integration research area; its basic assumption is that there is a pre–existing ontology and that the designer has to introduce new concepts in it. Furthermore we must consider a more complex scenario where more designers need to integrate new concepts into the same ontology. Thus, two different extensions of the same ontology, about the same domain, have to be aligned to obtain a consistent integrated view of all the data previously mapped on them. Concerning to this scenario, the need to study and to exploit techniques to align two or more ontologies takes shape. The aim of this work is to design and develop a tool that allows the designer of an integration system to browse efficiently the reference ontology and to extend and update it. The developed tool uses the lexical database WordNet available online as the reference lexical ontology and the **MOMIS** system, developed at the Department of Science Engineering at the University of Modena and Reggio Emilia, in Italy, as the information integration system.

Introduction

The basic assumption of this work is the following:
given a concept, every person is able to enrich and to interpret it in an individual way.

This work has to be placed in the intersection of two wide research areas like information integration and ontologies management, wandering from a local perspective of the integration systems till a distributed cooperation among them.

Two main limitations of the available Information Integration Systems are the following:

- the sources integration process can never be performed in a completely automated way, needing therefore a human supervisor

- the data sources annotation phase can often lead to loss of knowledge, due to the absence of concepts related to specific domains in the system's reference ontology

Considering the second limitation, we focus now on the data sources annotation phase.
We perform this operation annotating a source's structure, not in its description language, but in terms of attribute and class names of a common language, called ODL$_{I^3}$. In the following we will refer to these elements names as the *lexicon associated to the particular data source*. A software component, named *wrapper* extracts the structure of a source, translating the resulting description in the ODL$_{I^3}$ language.

Then, it can be said that, the designer of an integration process performs the annotation of all the data sources to be integrated in order to map their lexicon in a common language.

This annotation phase involves the lexicon reference ontology exploited by the particular integration system.

The annotation phase is a rather critical one, since, it compromises the next data integration step. Infact, given a particular concept of a data source, each designer may represent it with different terms or also interpret it in different ways, even if in the same context; thus it is necessary to reconciliate the different lexicons on the basis of a common meaning extracted from the lexicon reference ontology.
Moreover, during the annotation phase, it is possible that no meaning, which fits a particular element of a data source, be found in the lexicon reference ontology.
The above situation is not at all rare, since we have often to perform data integration relating to a *specific domain of interest*, i.e. we have often to manage specific lexicon characterized data.

On the other hand, in order to be able to integrate data belonging to many different domains, the most appropriate solution is the adoption of an *upper level ontology*, which represents a set of rather general and well–known concepts, shared by the most part of human knowledge.
If this should not be the case, we will be able to integrate only data characterized by a particular domain of interest. For example, if we exploit a medical reference ontology, it can be said we are able to successfully integrate only data related to the medical domain.

The adoption of an upper level ontology aids in annotating data belonging to many domains of interest, but, on the other side, it hinders the semantic richness related to them, lacking very often of specific and particular concepts.
Concerning to the last situation, a designer that doesn't find a satisfactory meaning in the reference ontology for a given concept, may annotate it with a similar one, generating in this way a *partial* loss of knowledge. On the other hand, the designer may also state not to annotate such a concept, thus generating a *total* loss of knowledge.

The problem explained above is a very serious one since it strongly limits the exploitation of the semantic associated to data sources in an information integration process.

This work focuses on the solution of the "semantic annotation of sources", offering to the designer the possibility to extend with his own meanings the lexicon reference ontology. Notice that this means we will be theoretically able to anno-

tate *every* concept.
Moreover, physically extending the reference ontology implies also the reusability, in the future, of the already annotated concepts.
The proposed solution is a centralized one since the supervisor designer is able to modify any extension of the lexicon reference ontology. However, the developed technical solution will allow in the future concurrent activities of more designers, which may also create inconsistent extensions of the reference ontology.

The developed software extended the SI–Designer tool of the **MOMIS** integration system which exploits WordNet as its reference lexicon ontology. The main aim of the developed tool, **WNEditor** , is to make the designer able to efficiently browse and to extend WordNet with his own new lexicons, meanings and relations among them.

In order to face the problem of an ontology extension and to be able to foresee all the consequences of such an action, we based our work on a wide interdisciplinary scientific literature.
In particular, during the work, two important aspects arose:

- the will to help the designer during the ontology extension process by exploiting information retrieval techniques
- the consideration of the consequences deriving from the use of more reference ontologies within a distributed cooperation among different instantiations of the same integration system

Both these aspects will be strongly deepen in the following of this thesis.

Now follows the main organization of the work:

Chapter 1
This chapter is an introduction to WordNet and an explanation of its main features, from the basic assumptions to the implementation.

Chapter 2
This chapter explains the interaction that exists between the **MOMIS** system and the **WordNet** ontology.
It also describes the information retrieval techniques exploited in the WordNet extension process.

Chapter 3
This chapter is an introduction to the practical use of the developed tool **WNEditor** showing all the GUI and the library functionalities and specifications.

Chapter 4
This chapter illustrates all the software architecture specifications and the development of **WNEditor** . The implemented similarity functions are also presented and an evaluation of a similarity search task is described.

Chapter 5
This chapter is an overview on the actual state of the art regarding ontologies definitions and management, from their alignment to their fully integration.

Chapter 6
Starting from a more complex scenario, concerning a distributed cooperation among more integration systems, this chapter is an overview on the new and different problems to be faced in data integration and ontology extensions.

Chapter 7
This chapter reports the main conclusions about this thesis considering also future work in the information integration research area.

Related Work

This is a brief introduction to the main related work exploited in this thesis. More considerations and details can be found in chapter 5.

The main part of the related work exploited in this thesis is concerning the so called "ontologies". Very useful is [Gua97] that argues about the general concept of ontology, giving different definitions of it and explaining the ontology–based modelling problem. In [KFvHH] the authors deepen the relation between ontology representation languages and web document structure techniques (the schemas), giving a detailed comparison of OIL, proposal language for representing ontologies on the Web, with XML Schema, the standard for describing the structure and semantics of XML based documents.
[HH00] can be seen as the next step since the authors discuss the problem of managing ontologies in distribuited environment such as the Web, considering a Web–based knowledge representation language. In order to have a clear distinction between the two main categories that every ontology may belong, [Pea01,

PN01, NP01] illustrate the need of an effort to create a *large general purpose and formal ontology*. In particular the IEEE Standard Upper Ontology project has the purpose of create an open standard ontology reusable for both academic and commercial purposes and able to support additional domain specific ontologies.
Analysis and solutions of problems like ontology reuse, maintenance and versioning can be found in [HH00, JW, DF].
An analysis of problems and solutions in combining and relating ontologies is discussed in [Kle].
The author illustrates not only technical obstacles but also practical problems in aligning and merging ontologies, focusing in particular on these aspects:

- it is difficult to find the terms that need to be aligned
- the consequences of a specific mapping (unforeseen implications) are difficult to see
- repeatability of merges, since the sources that must be merged evolve continuously

A more specific and formal framework on ontologies' integration problem can be found in [CDL02], while a different and new algebraic–based approach in composing diverse ontologies is shown in [WJ98] and [Wie94].
All these cited works were very useful to approach to the nowadays ontology treatment problem, however no specific work addressing the problem of extending the reference ontology of an integration system was found.
Our idea of an ontology's extension based on semantics is in the line of machine processable semantics introduced in WWW by Tim Berners Lee [LHL01] and that opens the research area called "Semantic Web".

In order to have an overview of the ongoing research in the area of the semantic web especially focusing on ontology technology, [DFKO02] and [Fou] were very interesting and useful.
Moreover, when we spread our view according with the ontology extensions export problem, we can refer to any possible distributed environment such as the Web, but we can also exploit the specific architecture illustrated in the european project SEWASIE, coordinated by Prof. Sonia Bergamaschi, which extends the **MOMIS** integration system (**http://www.sewasie.org**).
The SEWASIE project aims to provide an open and distributed architecture offering the ability to fit in changing and growing environments and to interoperate with other systems. Its network information nodes are independent components that semantically enrich existing data sources by linking the data to ontologies and

other metadata. Starting from this point we could see a couple of these nodes as two cooperating **MOMIS** integration systems and then refer to our case.

Base terminology

In this document we argue about concepts like sources, classes, interfaces, attributes, lemmas which are briefly introduced, here.

A *Local Source* is a source of data, such as a *database*, a text file or an html document. Each local source is interfaced to **MOMIS** through a *wrapper*. A *Local Class* is an *interface* (or *class*) present in a *local source*. A *Local Attribute* is an attribute of a *local class*. The *Integration Designer* is a person who performs the integration of sources by exploiting an integration system like **MOMIS**. *Extender* is the particular name with which we refer to an integrations designer that is extending the integration system reference ontology, that is who performs any ontology extension.

A *Lemma* is a word composed of more terms. A *Term* is the atomic unit in which every lemma can be decomposed. A *Gloss* is here intended as a particular meaning's definition.

Stemming is a technique for reducing words to their grammatical roots.

Stopwords are words which occurr frequently within a document text. Examples of stopwords are prepositions, articles and conjunctions.

When refer to a *demorphing phase* we intend we are removing all the stopwords from a definition and we are applying a stemming algorithm to it.

Chapter 1

The WordNet ontology

This chapter is an introduction to WordNet and an explanation of all its main features.
WordNet is not merely an online thesaurus and it couldn't neither be considered an ontology as well.
On the other hand, the number of applications where WordNet is used more as an ontology than just as a lexical resource is growing. As is reported in [Gua97], to treat WordNet like an ontology, some of its lexical links (for example hyponym/hypernym relation) need to be interpreted according to some formal semantics which could add information about *the world* beyond the ones about the language.

But, even this is not the place to deeply discuss about ontologies, what is exactly their definition?
Not only a unique explanation but many interpretations for this concept exist, among which:

> A (AI-) ontology is a theory of what entities can exist in the mind of a knowledgeable agent.
>
> [WS93]

> An ontology for a body of knowledge concerning a particular task or domain describes a taxonomy of concepts for that task or domain that define the semantic interpretation of the knowledge
>
> [Alb93]

> An ontology is an explicit, partial specification of a conceptualization that is expressible as a meta level viewpoint on a set of possible domain theories for the purpose of modular design, redesign and reuse of knowledge-intensive system components.
>
> [SWJ]

Concerning to this new and actual role of WordNet we will refer to it in this work also as a *lexicon ontology*.

1.1 What is WordNet exactly ?

> *WordNet is an online English lexical reference system whose design is inspired by current psycholinguistic theories of human lexical memory. English nouns, verbs, and adjectives are organized into synonym sets, each representing one underlying lexical concept. Different relations link the synonym sets.*

WordNet was developed at Princeton University by Miller *et al.* [Mil95]; it is available on line at **http://www.cogsci.princeton.edu/∼wn/**. The WordNet version we exploit in this work is the 1.6 one.

It sets its basis on the exploitation of an organization of terms in concepts rather than by alphabetical order. Since it instantiates hypotheses based on results of psycholinguistic research, WordNet can be said to be a dictionary based on psycholinguistic principles.

The most obvious difference between WordNet and a standard dictionary is that the former divides the lexicon into five syntactic categories: nouns, verb, adjectives, adverbs and function words. However actually, WordNet contains only nouns, verbs, adjectives and adverbs. The price of imposing this syntactic categorization is a certain amount of redundancy that conventional dictionaries avoid; word *home*, for example, turns up in more than one category.
On the other hand, the advantage is that fundamental differences in the semantic organization of these syntactic categories can be clearly seen and exploited.
In particular, nouns are organized in lexical memory as topical hierarchies, verbs are organized by a variety of entailment relations, adjectives and adverbs are organized as n-dimensional hyperspaces.

These different ways to implement the lexical structures are needed since attempts to impose a single organizing principle on all syntactic categories would badly misrepresent the psychological complexity of lexical knowledge.

1.1.1 The Lexical Matrix

Lexical semantics begins with the recognition that a word is a conventional association between a lexicalized concept and an utterance that plays a syntactic role.

Since the word *word* is commonly used to refer both to the utterance and to its associated concept, discussions of this lexical association are vulnerable to terminological confusion.

Therefore, in order to reduce ambiguity, *word form* is used here to refer to the physical utterance or inscription and *word meaning* to refer to the lexicalized concept that a form can be used to express.

The starting point for lexical semantics can be said to be the mapping between forms and meanings. To perform this operation it has to take into consideration that different syntactic categories of words may have different kinds of mappings. Referring to table 1.1, the notion of lexical matrix will be more clear. In particular consider F_1 and F_2 synonyms where F_2 is a polysemous word:

	F_1	F_2	F_3	$\dots$	F_n
M_1	$E_{1,1}$	$E_{1,2}$			
M_2		$E_{2,2}$			
M_3			$E_{3,3}$		
.					
M_m					$E_{m,n}$

Table 1.1: The WordNet's Lexical Matrix

Word forms are imagined to be listed as headings for the columns; word meanings as headings for the rows.

Each element in the matrix implies that the form in that particular column can be used in an appropriate context to express the meaning in that particular row. Thus, entry $E_{1,1}$ implies that word form F_1 can be used to express word meaning M_1. If there are at least two entries in the same column then the corresponding word form is polysemous (i.e. it can be used to represent more than one meaning, exactly two in this case); if there are at least two entries in the same row then two word forms are synonyms relative to a context.

But how are words' meanings represented in WordNet? In order to simulate a lexical matrix in a computer it must be possible to represent both forms and meanings.

Following the adopted differential theory the only requirement is that if the person who reads a definition has already acquired the underlying concept and needs merely to identify it, then a synonym (or near synonym) is often sufficient.
So the word meaning M_1 is represented by simply listing the word forms that can be used to express it: $\{F_1, F_2, \ldots\}$.
For example the two synonym sets {*board*, *plank*} and {*board*, *committee*} are the identifiers of the two meanings of *board a piece of lumber* and *a group of people assembled for some purpose* respectively.

Concerning to this over mentioned example and *assuming people who know English to be able to recognize the concepts from the words listed in the synonym set*, it became possible to represent the lexical matrix by a mapping between written words and *synsets*.

1.2 Relations

WordNet distinguishes between semantic and lexical relations, where the first ones are relation between meanings while the second ones are relations between words.

> *The relations represent associations that form a complex network; knowing where a word is situated in that network is an important part of knowing the word's meaning.*

Each relation is internally represented as a pointer. The general rule all these pointers are submitted is that *between different syntactic categories objects cannot exist a relation*.
On the other hand, following pointers' types cross from one syntactic category to another so they are an exception to the over mentioned rule.
Lexical relations exist between relational adjectives and the nouns that they relate to, and between adverbs and the adjectives from which they are derived. Antonyms are also lexically related.
Semantic relations exist between adjectives and the nouns for which they express values are encoded as attributes. The semantic relation between noun attributes

and the adjectives expressing their values are also encoded.
Synonymy of word forms is implicit by including them in the same synset.
Please refer to the next sections to have a clearer view of this distinction.

1.2.1 Semantic Relations

WordNet is founded on the semantic relations.
Since a semantic relation is a relation between meanings, and since meanings can be represented by synsets, it is natural to think of semantic relations as pointers between synsets.
This relation category is characterized by reciprocity: if there is a semantic relation R between meaning $\{x, x^{'},\ldots\}$ and meaning $\{y, y^{'},\ldots\}$, then there is also a relation $R^{'}$ between $\{y, y^{'},\ldots\}$ and $\{x, x^{'},\ldots\}$.
Moreover, if the relation between the meanings $\{x, x^{'},\ldots\}$ and $\{y, y^{'},\ldots\}$ is called R, then R will also be used to designate the relation between all individual word forms belonging to those synsets.

Hyponymy

Hyponymy/hypernymy (also indicated with subordination/superordination, subset/superset, or the ISA relation) is a semantic relation between word meanings: e.g., {maple} is a hyponym of {tree}, and {tree} is a hyponym of {plant}.
A concept represented by the synset $\{x, x^{'},\ldots\}$ is said to be a hyponym of the concept represented by the synset $\{y, y^{'},\ldots\}$ if native speakers of English accept sentences constructed from such frames as *An x is a (kind of) y*.
The relation can be implemented by including in $\{x, x^{'},\ldots\}$ a pointer to its superordinate, and including in $\{y, y^{'},\ldots\}$ pointers to its hyponyms.

Hyponymy is transitive and asymmetrical, and, since there is normally a single superordinate, it generates a hierarchical semantic structure, in which an hyponym is said to be below its superordinate.
An hyponym inherits all the features of the more generic concept and adds at least one feature that distinguishes it from its superordinate and from any other hyponyms of that superordinate. This convention provides the central organizing principle for the nouns in WordNet.

Meronymy

Meronym is the partwhole or HASA relation, known as meronymy/holonymy.

A concept represented by the synset $\{x, x^{'},\ldots\}$ is a meronym of a concept represented by the synset $\{y, y^{'},\ldots\}$ if native speakers of English accept sentences constructed from such frames as *A y has an x* (as a part) or *An x is a part of y*. The meronymic relation is transitive (with qualifications) and asymmetrical, and can be used to construct a part hierarchy (with some reservations, since a meronym can have many holonyms). It is assumed that the concept of a part of a whole can be a part of a concept of the whole.

Entailment

The semantic relations among verbs in WordNet all interact with entailment.

In logic, entailment, or strict implication, is properly defined for propositions; a proposition P entails a proposition Q if and only if there is no conceivable state of affairs that could make P true and Q false.

Entailment is a semantic relation because it involves reference to the states of affairs that P and Q represent. The term will be generalized here to refer to the relation between two verbs V_1 and V_2 that holds when the sentence Someone V_1 logically entails the sentence Someone V_2; this use of entailment can be called lexical entailment.

Thus, for example, snore lexically entails sleep because the sentence He is snoring entails He is sleeping; the second sentence necessarily holds if the first one does.

Lexical entailment is a unilateral relation: if a verb V_1 entails another verb V_2 , then it cannot be that case that V_2 entails V_1. In particular, where two verbs can be said to be mutually entailing, they must also be synonyms, that is, they must have the same sense.

Negation reverses the direction of entailment: not sleeping entails not snoring, but not snoring does not entail not sleeping. The converse of entailment is contradiction: If the sentence He is snoring entails He is sleeping, then He is snoring also contradicts the sentence He is not sleeping.

The entailment relation between verbs resembles meronymy between nouns, but meronymy is better suited to nouns than to verbs.

Cause To

Although many languages have a means to express causation, not all languages lexicalize the causative member independently; causation is often marked by a morpheme reserved for this function. However, English does not have many lexicalized causative-resultative pairs, such as show-see.

Verb group

The semantic relations used to build networks of nouns and adjectives cannot be applied to verbs without modification, but have to be adapted to fit the semantics of them, which differ substantially from those of the other lexical categories.

Summary different semantic groups of verbs have distinct structures. Some parts can be cast into a taxonomic framework by means of the troponymy relation; this is generally true for verbs of creation, communication, competition, contact, motion, and consumption.

Similar to

Adjective synsets are regarded as clusters of adjectives associated by semantic similarity to a focal adjective that relates the cluster to a contrasting cluster at the opposite pole of the attribute. Thus, ponderous is similar to heavy and heavy is the antonym of light, so a conceptual opposition of ponderous/light is mediated by heavy. Direct antonyms, like heavy/light, are conceptual opposites that are also lexical pairs, while indirect antonyms, like heavy/weightless, are conceptual opposites that are not lexically paired. Under this formulation, all descriptive adjectives have antonyms; those lacking direct antonyms have indirect antonyms, i.e., are synonyms of adjectives that have direct antonyms.

In WordNet, direct antonyms are represented by an antonymy pointer; indirect antonyms are inherited through similarity, which is indicated by the similarity pointer.

Attribute

Attributes are given by adjectives. As the coverage of WordNet increased, it became increasingly obvious that alternative senses of a word could not always be identified by the use of synonyms.

Values of attributes are expressed by adjectives. For example, size and color are attributes of canaries: the size of canaries can be expressed by the adjective small, and the usual color of canaries can be expressed by the adjective yellow. There is no semantic relation comparable to synonymy or hyponymy that can serve this function, however. Instead, adjectives are said to modify nouns, or nouns are said to serve as arguments for attributes: Size(canary) = small.

Here it is sufficient to point out that the attributes associated with a noun are reflected in the adjectives that can normally modify it.

For example, a canary can be hungry or satiated because hunger is a feature of animals and canaries are animals, but a stingy canary or a generous canary could only be interpreted metaphorically, since generosity is not a feature of animals in general, or of canaries in particular.

Descriptive adjectives are what one usually thinks of when adjectives are mentioned. A descriptive adjective is one that ascribes a value of an attribute to a noun. That is to say, x is adjective Adj presupposes that there is an attribute A such that $A(x) = Adj$.

To say The package is heavy presupposes that there is an attribute WEIGHT such that WEIGHT(package) = heavy. Similarly, low and high are values for the attribute HEIGHT.

WordNet contains pointers between descriptive adjectives and the noun synsets that refer to the appropriate attributes.

1.2.2 Lexical Relations

Synonymy

Synonymy is, of course, a lexical relation between word forms, on the other hand it should be considered the most important relation among existing ones. Actually since the ability to judge that relation between word forms is a prerequisite for the representation of meanings in a lexical matrix, it is clear that the most important relation for WordNet is similarity of meaning.

One synonymy's definition usually attributed to Leibniz is:

> two expressions are synonymous if the substitution of one for the other never changes the truth value of a sentence in which the substitution is made

According to this definition, true synonyms are rare, if they exist at all.

Another version of synonymy would make this relation relative to a context:

> two expressions are synonymous in a linguistic context C if the substitution of one for the other in C does not alter the truth value

It's a certainty that the definition of synonymy in terms of substitutability makes it necessary to partition WordNet into nouns, verbs, adjectives, and adverbs.

That is to say, if concepts are represented by synsets, and if synonyms must be interchangeable, then words in different syntactic categories cannot be synonyms, i.e. cannot form synsets, because they are not interchangeable.

In the end, synonymy is best thought of as one end of a continuum along which similarity of meaning can be graded. It is probably the case that semantically similar words can be interchanged in more contexts than can semantically dissimilar words. This relation is assumed to be a symmetric one, i.e. if x is similar to y then y is equally similar to x.

Antonymy

The antonym of a word x is sometimes $notx$, but not always.

For example, rich and poor are antonyms, but to say that someone is not rich does not imply that they must be poor.

Antonymy is a lexical relation between word forms, not a semantic relation between word meanings. For example, the meanings $\{$rise, ascend$\}$ and $\{$fall, descend$\}$ may be conceptual opposites, but they are not antonyms: [rise/fall] are antonyms and so are [ascend/descend]. In the end, antonymy provides a central organizing principle for the adjectives and adverbs in WordNet.

See also

Many head synsets contain pointers to other, related clusters. In this AWAKE–ASLEEP cluster, the capitalized pointer ALERT,& points to the head word of the ALERT–UNALERT cluster. These capitalized pointers are planned to serve as *see also* crossreferences to related clusters.

Pertaynym

Concerning to the adjectives organization and bipartition, the relational adjectives too, can occur only in attributive position, although for some adjectives, this constraint is somewhat relaxed. Relational adjectives mean something like *of, relating/pertaining to, or associated with* some noun, and they play a role similar to that of a modifying noun.

For example, fraternal, as in fraternal twins relates to brother, and dental, as in dental hygiene, is related to tooth. Some head nouns can be modified by both the relational adjective and the noun from which it is derived: both atomic bomb and atom bomb are admissible.

Derived from

Relational adjectives are most often derived from Greek or Latin nouns, and less often from the appropriate AngloSaxon noun. The English lexicon frequently has several (synonymous) adjectives derived from nouns in different languages that express the same concept: Greekbased rhinal and AngloSaxon nasal both relate to nose; the relational adjectives corresponding to word are verbal (from Latin) and lexical (from the Greek). Moreover, many verbs are also derived from adjectives via the en suffix; they tend to have an antonymic verb derived from the base adjective's antonym: weakenstrengthen, shortenlengthen . . . Also adverbs can be derived from adjectives.

Participle of verb

All languages provide some means of modifying or elaborating the meanings of nouns, although they differ in the syntactic form that such modification can assume. English syntax allows for a variety of ways to express the qualification of a noun.
For example, if chair alone is not adequate to select the particular chair a speaker has in mind, a more specific designation can be produced with adjectives like large and comfortable. Words belonging to other syntactic categories can function as adjectives, such as present and past participles of verbs (the creaking chair; the overstuffed chair) and nouns (armchair, barber chair).

1.3 WordNet's organization for each syntactic category

The figure 1.1 shows the percentage distributions of word forms and synsets respectively for each syntactic category.

1.3.1 Nouns

Since the definitions of common nouns typically give a superordinate term plus distinguishing features, WordNet organizes noun files by exploiting the superordinate relation (hyponymy) that resulting in a hierarchical semantic organization of nouns.
The hierarchy is limited in depth, seldom exceeding more than a dozen levels. Distinguishing features are entered in such a way as to create a lexical hierarchy in which each word inherits the distinguishing features of all its superordinates.

These above mentioned hierarchies can also be referred as *inheritance systems* since a specific item inherits information from its generic superordinate; more-

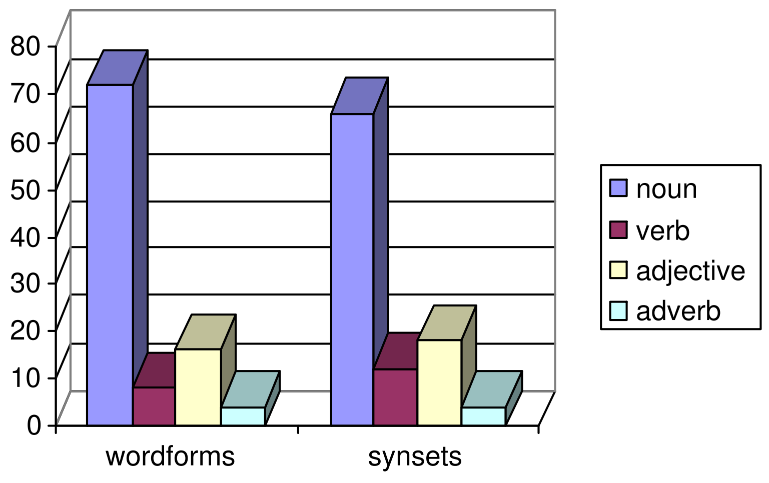


Figure 1.1: Distribution of word forms and synsets within WordNet for each sytactic category

over instead of listing these common properties redundantly with both items, they are listed only with the superordinate while a pointer from the subordinate to superordinate is maintained.

1.3.2 Verbs

Verbs are organized as in a hierarchical way like nouns but unlike their organization, this hierarchy is much more limited in depth. Hypernym relation is here named troponymy, however the most important relations for this syntactic category are the entailment and the cause.

1.3.3 Adjectives and Adverbs

The set of adjectives is divided into two classes: descriptive and relational.

Decriptive adjectives ascribe to their head nouns values of (typically) bipolar attributes and consequently are organized in terms of binary oppositions (antonymy) and similarity of meaning (synonymy).

The descriptive adjectives that do not have direct antonyms are said to have indirect antonyms by virtue of their semantic similarity to adjectives that do have direct antonyms.

WordNet contains pointers between descriptive adjectives expressing a value of an attribute and the noun by which that attribute is lexicalized.

Relational adjectives are assumed to be stylistic variants of modifying nouns and so are crossreferenced to the noun files.

Descriptive adjectives are organized into clusters that represent the values, from one extreme to the other, of some attribute. Thus each adjective cluster has two (occasionally three) parts, each part headed by an antonymous pair of word forms called a head synset.

Most head synsets are followed by one or more satellite synsets, each representing a concept that is similar in meaning to the concept represented by the head synset. One way to think of the cluster organization is to visualize a wheel, with each head synset as a hub and its satellite synsets as the spokes. Two or more wheels are logically connected via antonymy, which can be thought of as an axle between wheels.

Often adverbs are derived from adjectives having sometimes an antonym.

1.4 WordNet's implementation

WordNet's source files are written by lexicographers: a set of lexical and semantic relations are used to represent the organization of lexical knowledge.

As already mentioned, two kinds of building blocks coexist in the source files: word forms and word meanings.

Each word form is represented as the orthographic representation of an individual word or a string of individual words joined with underscore characters.

Due to the use of this particular character to represent a lemma (i.e. a set of terms forming a word. For example the lemma fountain_pen is a single word formed by two terms) also the user of the developed program **WNEditor** is requested to respect this rule when searching for a synonym.

A relational pointer represents the relation between the word forms belonging to a synset and others belonging to another one. These pointers can be lexical or semantic.

Table 1.2 summarizes the relational pointers by syntactic category.

Moreover, many pointers are reflexive, meaning that if a synset contains a pointer to another synset, the other synset should contain a corresponding reflexive pointer back to the original synset.

The developed tool **WNEditor** automatically generates the reflexive relation's

Noun		Verb		Adjective		Adverb	
Antonym	!	Antonym	!	Antonym	!	Antonym	!
Hyponym	~	Troponym	~	Similar to	&	Derived from	\
Hypernym	@	Hypernym	@	Relational adj.	\		
Meronym	#	Entailment	*	Also see	^		
Holonym	%	Cause	>	Attribute	=		
Attribute	=	Also see	^				

Table 1.2: Relational pointers for each syntactic category

pointer when the user creates a relation belonging to one of the following types listed in table 1.3.

Pointer	Reflexive pointer
Hyponym	Hypernym
Hypernym	Hyponym
Holonym	Meronym
Meronym	Holonym
Antonym	Antonym
Similar to	Similar to
Attribute	Attribute

Table 1.3: WordNet's Reflexive pointers

For each syntactic category, two files represent the WordNet database:

1. **index.pos** (pos.idx for all other platforms different from *Sun*)
2. **data.pos** (pos.dat for all other platforms different from *Sun*)

where pos is either NOUN, VERB, ADJ or ADV.
All these files are in an ASCII format, that is human and machine readable. All fields, unless otherwise noted, are one space character separated while all lines end with a *newline* character.

Index files

Each index file index.pos is an alphabetized list of all of the word forms found in WordNet in the corresponding syntactic category. On each line, following the word, there is a list of *byte_offset* in the corresponding data file data.pos, one for every synset containing the word. All words in this files are in lower case only, that results in allowing case insensitive database searches.

Data files

Each data file's line contains all the information about a specific synset ending with a newline character. Going into details, for each synset in the corresponding syntactic category its byte_offset, its synonym's number, its gloss and all existing pointers from this synset to another one are further reported. Table 1.4 shows the generic pointer's format.

pointer_symbol	byte_offset	pos	source/target

Table 1.4: Generic relational pointer's format

where pointer_symbol is one of the symbols listed in table 1.2, byte_offset is the synset_offset of the target synset in the data file corresponding to pos and source/target field distinguishes between lexical and semantic relation. It is a four byte hexadecimal field where the most significant two bytes indicate the word number in the source synset, i.e. the head synset for the current line, and the other two bytes indicate the word number in the target synset.
If its value is 0000 this pointer_symbol represents a semantic relation between the source and the target synset, on the contrary case there is a lexical relation between two words placed in source and target synset respectively. In particular, the first and last two bytes indicate the words' number within the synsets.
The index and data files are interrelated. Part of each entry in an index file is a list of one or more byte offsets, each indicating the starting address of a synset in a data file.
The first step to the retrieval of synsets or other information is typically a search for a word form in one or more index files to obtain all data file addresses of the synsets containing the word form.

Although all these over mentioned files are in ASCII and are therefore editable and in theory extensible, in practice this is almost impossible due to the wrong created byte offsets that would thus result in the incorrectness of searching strategies.
Concerning to this limitation, the main aim of this work is to make the user able to extend WordNet through a specific library and an easy to use GUI application.

Chapter 2

Extending the pre–existing ontology

In order to clarify how the integration system we have been working with exploits its reference lexical ontology follows a brief explanation of the interaction that exists between the **MOMIS** architecture and the **WordNet** ontology.
Moreover, an overview of the information retrieval techniques implemented in order to improve the WordNet extension process is illustrated.

2.1 The MOMIS-WordNet interaction

Developing intelligent tools for the integration of information extracted from multiple heterogeneous sources is a challenging issue to effectively exploit the numerous sources available on-line, for example, in global, Internet-based information systems.
Information sources to be integrated are usually pre-existing and have been developed independently.
Consequently, semantic heterogeneity can arise for the aspects related to terminology, structure, and context of the information, and has to be properly dealt with during integration in order to effectively and correctly exploit the information available in the sources.
Integration and reconciliation of data coming from heterogeneous sources is a research topic in databases [Hul97]. Several contributions have appeared in literature, including methods, techniques and tools for integrating and querying heterogeneous databases [CHS$^+$94, GKD97, LRO96, PGMW95].
This work exploits the *MOMIS* (Mediator envirOnment for Multiple Information

Sources) [BCV99, BCVB00, BBC$^+$00] project[1][2].

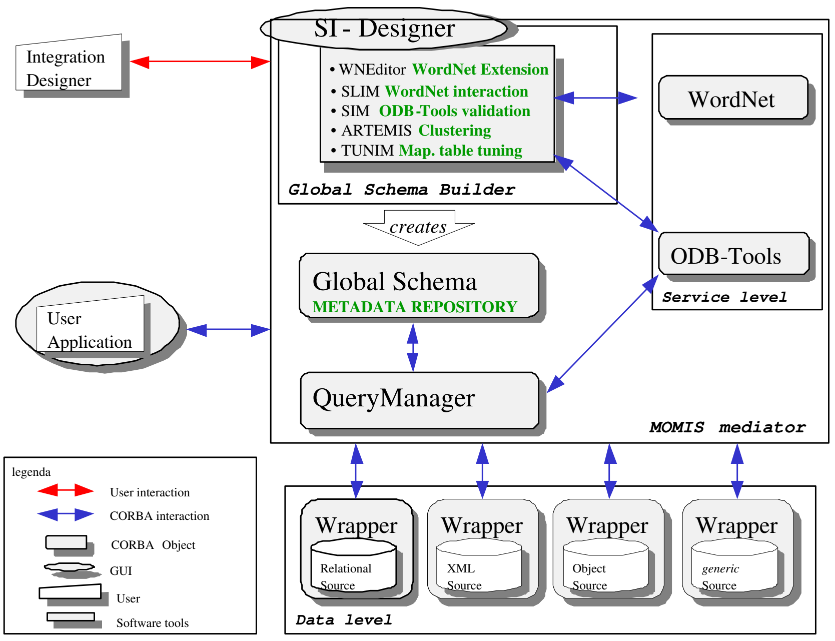


Figure 2.1: **MOMIS** architecture

The main goal of the information extraction and integration techniques developed in **MOMIS** is to construct synthesized, integrated descriptions (i.e., *a global virtual view*) of the information coming from multiple heterogeneous sources, to provide the user with a uniform query interface against the sources independent from their location and the level of heterogeneity of their data.

Moreover, to meet the requirements of global, Internet-based information systems with a possibly high number of sources to be integrated, it is important to develop tool-assisted techniques in order to automate as much as possible information extraction and integration activities. This goal has been achieved with the development of the SI-Designer tool [BBC$^+$00].

Most of the information for integration is provided by the *wrappers*, using ODL$_{I^3}$ from the source description and stored in ODL$_{I^3}$ data structure.

[1]**MOMIS** is a joint project among the Università di Modena e Reggio Emilia, the Università di Milano, and the Università di Brescia started within the Italian research project INTERDATA, theme n.3 “Integration of Information over the Web”, coordinated by V. De Antonellis, Università di Brescia.

[2]**MOMIS** was financed by the MURST in 2000/2001 within the *D2I: Integration, Warehousing, and Mining of Heterogeneous Data Sources* project

MOMIS uses WordNet lexical system [Mil95] as its *lexicon reference ontology* and the OLCD Description Logic inference capabilities to build the Common Thesaurus [CAV00, BCVB00, BN94]. The Common Thesaurus is composed of relationships between schema elements extracted from the schemata descriptions, derived from the semantic and lexical relationships between the concepts in WordNet, and explicitly stated by the integration designer.

The choice for **MOMIS** integration system concerning to its reference ontology is due most to these reasons: WordNet is a well-known and widespread lexical database, it seems to be a complete and professional instrument, is mentioned in many scientific papers and it is continuously reviewed. Last but not least WordNet is freeware.

The **MOMIS**-WordNet interface was developed by Giovanni Malvezzi [Mal00] and implements a technique to find intensional relations inter-schema. The goal is to discover affinities/similitudes between classes belonging to different data sources. The module extracts lexical relations between schema element names (classes and attributes are the elements) and works on the meaning of the names used to describe the classes and attributes content.
Having different sources it is however necessary to translate all their attribute and classes names in a common language, that resulting in a *mapping process of all those terms in a well defined set of concepts* (assumed as the WordNet ontology). This mapping process is often referred in literature as the *sources annotation phase*.
The above mentioned interface provides the interaction with WordNet by accessing the database files directly and it has been integrated in the SLIM module of the SI-Designer GUI. Actually, it is possible to extract from WordNet the following types of lexical and semantic relations:

Synonymy, **Hypernymy**[3], **Hyponymy**[4],

Holonymy[5], **Meronymy**[6], **Correlation**[7]

The relations coming from WordNet are proposed as semantic relations to be added to the *Common Thesaurus* according to the following mapping:

[3]Generalization relation
[4]Specialization relation
[5]Aggregation relation, *part*
[6]Aggregation relation, *all*
[7]The correlation is a relation between two terms in two synonym sets that shares the same father in hypernymy sense.

Synonymy:	corresponds to a	SYN	relation
Hypernymy:	corresponds to a	BT	relation
Hyponymy:	corresponds to a	NT	relation
Holonomy:	corresponds to a	RT	relation
Meronymy:	corresponds to a	RT	relation
Correlation:	corresponds to a	RT	relation

2.1.1 Sources annotation phase

What exactly the designer is requested for annotating the schema is to manually choose a WordNet's meaning for each schema element. Starting from the schema to be integrated, the designer must declare a relation between each schema element's name (class name or attribute name) and the WordNet meaning appropriated for the particular context.
Then the designer must choose one or more meanings for each name. This is a two steps process.

- **Word form choice**

  In this step, the WordNet morphologic processor should aid the designer. A *word form* is the word without any suffixes due to declination of conjugation.

  If such *word form* is not found or there is ambiguity[8], or it is not satisfactory, the designer can set a custom *word form.* Notice that this operation, however, doesn't allow the designer to extend physically WordNet ontology with his particular annotation concept.

- **Meaning choice**

  The designer can choose to map an element on zero, one or more senses. For example WordNet has 15 meanings for the *word form* `address` from which the most appropriate ones to the particular context are chosen. Notice that the user can choose a sense only among the existing ones, that is he is not able to extend WordNet with his new meanings which could be more appropriate to the particular domain of interest then the pre–existing ones.

[8]E.g. `axes` has 3 word forms: `ax` (1 sense), `axis` (5 senses), `axe` (2 senses).

2.2 Extending WordNet

Referring to the annotation phase described in section 2.1.1 and to the chapter 1 where the WordNet structure is deeply illustrated, we are now able to treat in a theoretical way the ontology extension process.

But what is the meaning of the “extension process” ?

For extending WordNet we refer to the possibility and need at the same time to exploit **new concepts**.

Concerning to the need of an extension process this is exactly the basic assumption of this work: in order to fully exploit the semantics associated to all the sources that must be integrated we should be able to maintain those semantics as they are. Often this is not possible as well due to the interaction between **MOMIS** and WordNet exploited in the sources annotation phase. If a source’s description element (i.e. a class or an attribute name) doesn’t find a correspondence within the reference ontology, then the designer is requested to adapt that element to an already existing concept or to completely ignore it. However, both these choices cause loss of information.

The **WNEditor** tool developed in this work is a GUI (graphic user interface) that exploits an underlying developed Java library in order to make the reference ontology extension possible.

The designer is able, by exploiting **WNEditor** , to create and manage new concepts, appending them to the pre–existing WordNet’s network by new relationships.

Clearly, due to the criticalness of the extension process, the designer should have the possibility to perform step–by–step operations on the ontology and check every his action.

Moreover, due to the huge and complexity of any lexical ontology, in particular of WordNet, we took into consideration to help the designer in the extension process.

Going into details, the main difficulty must be found in relating new concepts with the pre–existing ones, i.e. in the *building relations phase*.

In order to make more flexible and easy this onerous extension’s step we decided to exploit *information retrieval techniques*.

Referring to the previous chapter 1, every WordNet relation holds between two member: a *source synset* that can be considered the first member of the relation and a *target synset* considered the second member.

The problem to face in is the following: *supposed fixed the source synset, in what* *way we are able to help the designer in searching for the most appropriate target synset of a relation ?*

The immediate answer to this question is to make the designer able to choose the target synset of a new relation among a list of synsets regarded **similar** in some way to that representing the first member.
To obtain this set of candidates meanings we have to exploit the semantic associated to the definition of the first member. To perform this operation we referred to an heuristic known in literature as **definition match**.
The basic assumption of this text match technique is that *similar–enough natural language definitions should also provide some evidence of concepts similarity*.

To know and to study the needed information retrieval techniques we mostly referred to [BYRN]. The first problem we had to face in was that the majority of the cited methods were related to out and out text documents, while we had to retrieve semantic information from senses' definitions (WordNet glosses), that are in general less rich and shorter than a typical text document.
However, we focused on the techniques related to the vector spaces where a single document/definition is represented as a vector of terms.
At this point we run up another obstacle, since all these mentioned vectors based retrieval techniques make use of term weighting algorithms, that resulting in the need to assign a *numeric weight* to each term within a definition.
This last requirement can be well exploited in automated alignment processes but this was not our case. The main reason to believe that is concerning the way in which the designer should have interacted with the developed GUI (graphic user interface) **WNEditor**.
Actually, for a correct program's working way, each time the designer would have inserted a new concept into the ontology or modified an existing one, then all the actual ontology's definitions' term's weights should have been computed again.
Taking into consideration that there are at least 99600 synsets in the underlying ontology and that the computation of weights is a rather time consuming operation, that would have resulted in a too much heavy process for a real time application like the developed one.
Another implementation choice could have be to compute again all the weights only on an explicit designer's demand. However this last way could have led to uncorrect results of the similarity search process since we have exploited an up–to–date ontology not consistent with the reality.

Concerning to the above reasons we preferred to focus on a much more simple

and efficient way to get the set of synsets regarded similar to the first member of an hypothetical relation.
Referring in particular to the definition match proposed in [Hov98], the adoption of that technique in the developed tool seemed to be a less heavy computation process even if probably less accurate than terms' weights based methods.

On the other hand, the classical information retrieval techniques we studied and all the data represented with a WordNet synset, like synonyms and relations, are very important features in order to study a solution for the merging problem between a reference ontology and any WordNet extension.
This more complex scenario where more designers have to integrate their WordNet extensions is treated in chapter 6.

2.2.1 Analysis on the exploited similarity functions

Given a source synset's definition, two methods to perform a similarity search on it were implemented (see chapter 3):

1. a computation of the similarity involving all current definition's terms, referred in the following as the *full* one
2. a keywords based computation of the similarity involving only a subset, selected by the designer, of the current definition's terms, referred in the following as the *partial* one

Both these methods are based on the same assumption:

similar enough natural language definitions should also provide some evidence of concept similarity

and both are able to generate automatically the candidates list from which the designer can pick up a synset to relate to the current one.

To find out this list of proposal meanings we first implemented the definition match technique suggested in [Hov98], adapting it to our specific case:
consider the English definitions D_1 and D_2 of two concepts. Both definitions are separated into individual words which are compared to an English stop words list and then demorphed by a stemming process. With the remaining words three values are computed:

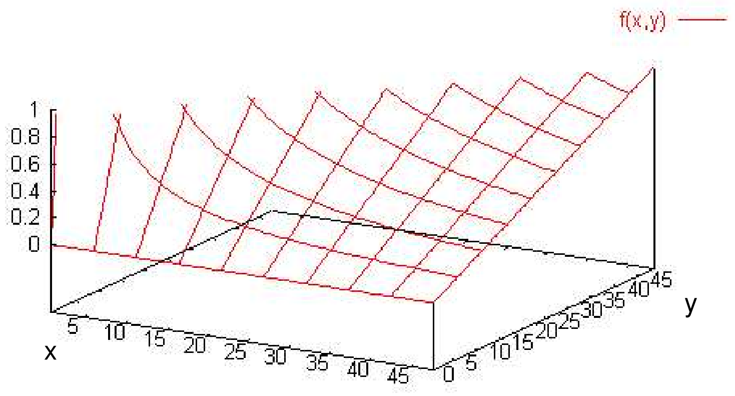


Figure 2.2: Function $f_0(x, y)$

- *reliability = number of shared words*
- *strength = ratio of reliability to number of words in the shorter definition*
- *defscore = (number of shared words/length of the shorter definition)*number of shared words*

We assume that D_1 is the current sense's gloss and D_2 is any another meaning's definition existing in the reference ontology.

The choice to consider the shorter definition to compute the strength value instead of the longer one is due most to the reason that in this way strength could max at 1.0. Otherwise, if we consider the longer definition, strength value can never max at 1.0. Moreover if the whole shorter gloss is inside the longer one we would still get some arbitrary score that depends on the length of the shorter definition.

Computing the score value in this way, we exploit a method that rewards for example three words matching more than just three–times one word matching, since once we have matched "easy" words, then matches should really count for something: so to "stretch" the match, strength value is multiplied by the number of shared words again.

However, since *defscore* doesn't in generally max at 1.0, we initially chose just strength value as a similarity index (actually it belongs to the interval $[0, 1]$).

Assuming y as the number of shared words between the two definitions and x as the number of words in the shorter one, the first implemented function, defined as $f_0(x, y) = y/x$, can be seen in figure 2.2.

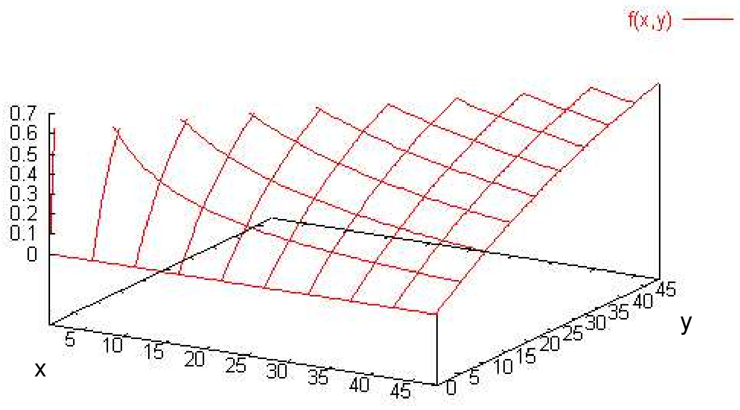


Figure 2.3: Function $f_1(x, y)$

However this initial choice didn't seem to be as sensible as we expected, so I studied and implemented two other functions that still get to max 1.0 but **grow more quickly with more matches**; in particular the most appropriate tendency in order to emphasize the matches grown seemed the exponential one.
First we compare $f_0(x, y)$ with its exponential version, i.e.

$$f_1(x, y) = 1 - \exp(-(y/x))$$

As we expected, $f_1(x, y)$ improves the similarity score assignment, giving a more "right" weight to some retrieved meanings than $f_0(x, y)$, but it performs the same score distribution of $f_0(x, y)$, like is reported in section 4.2.
Mainly for these reasons, we focus on a second function in order to get a more fine granularity in assigning score values and also to get a different results distribution. This function is defined as follows:

$$f_2(x, y) = 1 - \exp(-(y^2/x))$$

Function $f_1(x, y)$ is represented in figure 2.3, while figure 2.4 shows the 2D graphic representation of function $f_2(x, y)$.

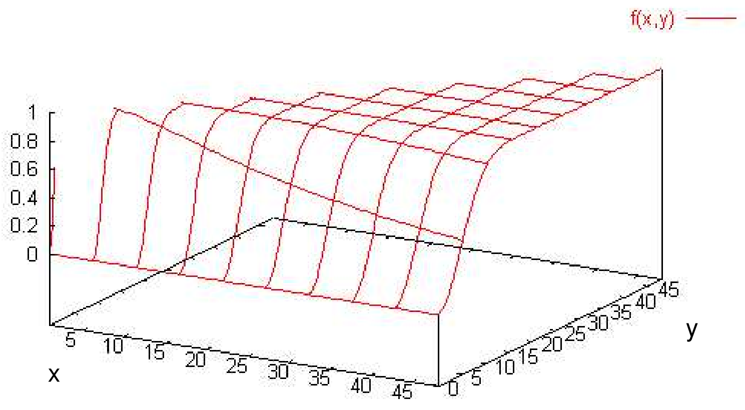


Figure 2.4: Function $f_2(x, y)$

We can now formally summarize the constraints on which $f_0(x, y)$, $f_1(x, y)$, $f_2(x, y)$ are based:

1. y can never be grater than x:

   $$y \leq x \qquad \forall x, y \in \mathbb{N}$$

   Actually the max match case happens when the whole shorter definition is included in the longer one.

2. x can never assume the value 0. Actually, the demorphing implemented algorithm replaces all the original terms of a gloss in the case that all of them are stop words. This choice is due to the impossibility, on the contrary case, to find out in a similarity search also meanings with an only *common words definition*.

   y can assume the value 0 when there are no matches between the two definitions.

3. both x and y assume only discrete values. In particular, not worrying about the max value (which is 48 for both x and y, see section 4.1.5):

   $$x \in \mathbb{N}$$

   $$y \in \mathbb{N}^0$$

Referring to figures 2.2, 2.3, 2.4, it should be noted how the three functions differently grow with incrementing values of variable y, i.e. with incrementing values of matches.

In particular, as we expected, function $f_2(x, y)$ grows more quickly with more matches than the others two. An example that compares similarity search results among $f_2(x, y)$, $f_1(x, y)$ and $f_0(x, y)$ is illustrated in section 4.2.

Chapter 3

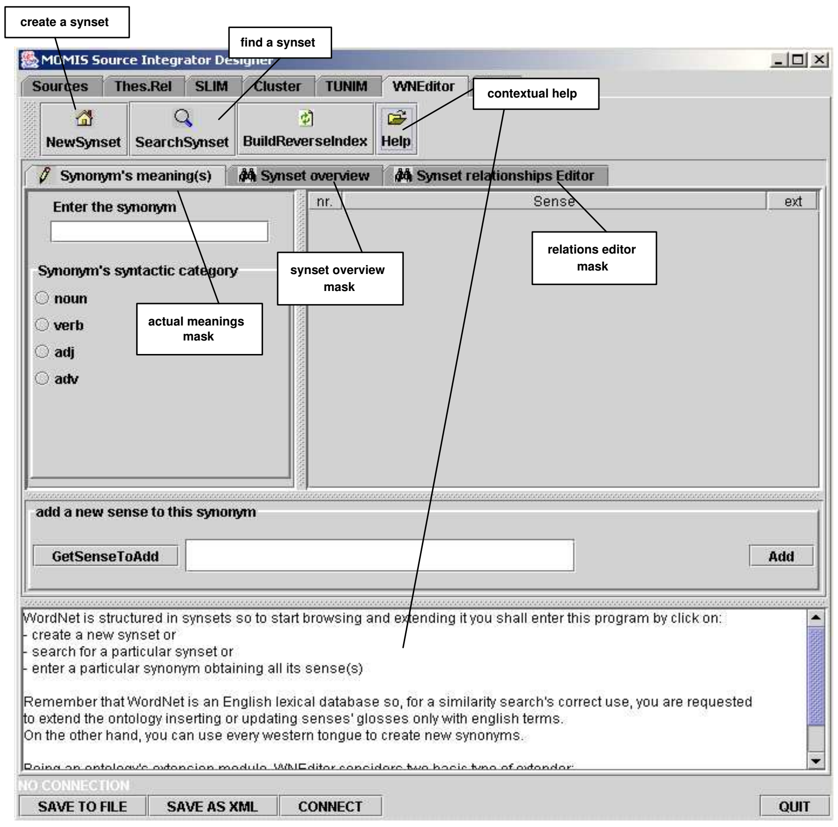
The WNEditor tool: user guide

This chapter is an introduction to the practical use of **WNEditor** showing all the GUI functionalities and specifications. To have a clear view of the working process of this tool the reader is requested to know deeply the WordNet ontology's structure (see chapter 1). We will both refer to the *user* and to the *designer* always intending the person who is extending WordNet.

3.1 How WNEditor works

The **WNEditor** 's philosophy is based on the awareness that the designer, who wants to browse and extend WordNet, knows the organization in synsets of this ontology.
WNEditor is a GUI application (Graphical User Interface), exploiting an underlying Java library to make the WordNet extension possible, that allows the user of **MOMIS** system to browse the lexicon reference ontology (WordNet) and, most of all, to extend it with his own new concepts.

The initial screen of **WNEditor** shown in figure 3.1 points out the whole work environment: on the toolbar located in the top of the screen the designer will find among the existing buttons the ones needed to *create a new synset*, to *find an existing synset* in the reference ontology and to *display the contextual help*.
Starting from this point, when the designer selects a synset by any of those ways described in the following sections, makes it automatically the **current synset** also referred as the **head** or **master synset** for that work session. Clearly, the designer is able to change the current synset, by selecting another one, at every moment.



WordNet is structured in synsets so to start browsing and extending it you shall enter this program by click on:
- create a new synset or
- search for a particular synset or
- enter a particular synonym obtaining all its sense(s)

Remember that WordNet is an English lexical database so, for a similarity search's correct use, you are requested to extend the ontology inserting or updating senses' glosses only with english terms.
On the other hand, you can use every western tongue to create new synonyms.

Figure 3.1: Tool **WNEditor** : initial screen

In the panel's central area there are three masks to let the user see respectively: all the actual meanings associated to a given synonym, an overview of the **current synset** including all its actual synonyms with all the relations involving it and all the actual relations between the **master synset** and another one selected by the designer.

WNEditor is characterized by three entry points as illustrated in figure 3.1 where the selected synset is automatically set as the current synset:

- By click on *NewSynset* button
- By click on *SearchSynset* button
- By write a synonym in the *Synonym's meaning(s)* panel

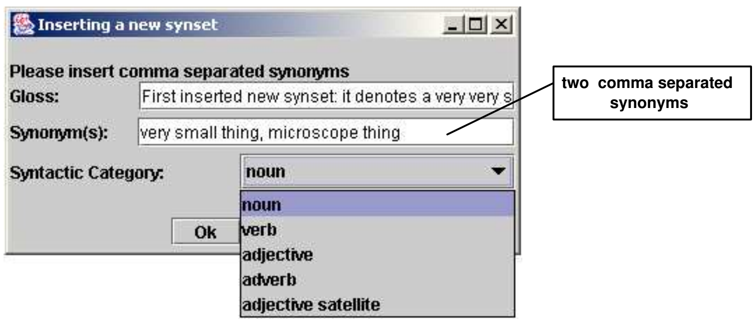
The next section goes into details on these three entry cases.

3.2 Use example of WNEditor

According to the initial selected entry point, chosen by the designer, the following sections show the program's behavior and its most important features.

3.2.1 Selecting the *NewSynset* entry point

The designer does this choice when he wants to create a **new** synset. Word **new** means *neither the meaning nor any of the entered synonyms already exist in the reference ontology*.


Inserting a new synset

Please insert comma separated synonyms

Gloss: First inserted new synset: it denotes a very very s

Synonym(s): very small thing, microscope thing

Syntactic Category: noun

Figure 3.2: Form to create a new synset in the reference ontology

The form shown in figure 3.2 requests to the designer to write

- the english meaning he wants to associate to this new synset
- zero or more *comma separated* synonyms. Notice that to extend WordNet the designer is requested to respect its conventions: in particular every lemma composed of more terms is managed as a string like *term1_term2_ _termn*. So, for example, the lemma "word for word" will be stored in the reference ontology as "word_for_word" (see section 1.4).

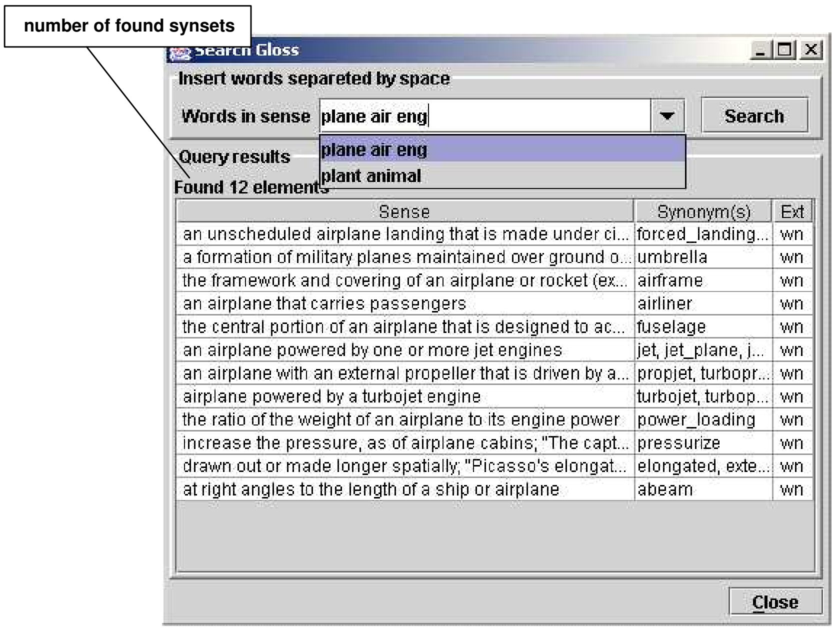
In the end the designer has to choose the syntactic category for the new synset. After having introduced the required data and confirmed them, the user can receive error messages: it could happen when the entered sense or at least one of the inserted synonyms already exist in the reference ontology.

The first case is explained with the possibility given by **WNEditor** to add synonyms, existing or not, to an already existing meaning in the ontology. The second

case is explained with the possibility given by **WNEditor** to add a sense, existing or not, to a synonym already existing in the ontology.
If the fitting is successful the new created synset is automatically set as the **master synset** of the current work session, setting up ***Synset overview*** and ***Synset relationships Editor*** panels.

3.2.2 Selecting the *SearchSynset* entry point

The designer does this choice when he has got clear in mind the ontology's concept that he is looking for and indeed he is able to insert keywords that presumably are in the sense's definition of that concept.



Found 12 elements

Sense	Synonym(s)	Ext
an unscheduled airplane landing that is made under ci...	forced_landing...	wn
a formation of military planes maintained over ground o...	umbrella	wn
the framework and covering of an airplane or rocket (ex...	airframe	wn
an airplane that carries passengers	airliner	wn
the central portion of an airplane that is designed to ac...	fuselage	wn
an airplane powered by one or more jet engines	jet, jet_plane, j...	wn
an airplane with an external propeller that is driven by a...	propjet, turbopr...	wn
airplane powered by a turbojet engine	turbojet, turbop...	wn
the ratio of the weight of an airplane to its engine power	power_loading	wn
increase the pressure, as of airplane cabins; "The capt...	pressurize	wn
drawn out or made longer spatially; "Picasso's elongat...	elongated, exte...	wn
at right angles to the length of a ship or airplane	abeam	wn

Figure 3.3: Form to search for a synset within the reference ontology

The displayed form shown in figure 3.3 is a search engine where the designer can write one or more space separated keywords. The search is done considering the whole ontology's network, that results in a set of synsets belonging, in general, to different syntactic categories.

Of course, according to the precision and selectivity of the inserted keywords, the result will be more or less satisfactory and of different size. In particular the implemented technique to perform this search is an approximate string match,

so the user is not requested to know exactly all the words in the meaning. For example we have inserted the keywords: *plane, air, eng*.
As reported in figure 3.3, every founded synset is characterized by its own sense, its all actual synonyms and its creator's code. It is worth noting that all these features allow the underlying ontology's browsing to be extremely flexible and easy.
Now the satisfied designer, double clicking on a particular synset, sets it as the current **master synset** making also active ***Synset overview*** and ***Synset relationships Editor*** panels.

WNEditor caches all the already inserted keywords. For example we insert the keywords: *home, house* and after performing the search we insert another set of keywords: *plant, animal*. Now the list displays both the query. It should be noted that even if the designer close this search form, his data are cached so he is able to do again a particular query by simply open this form.

3.2.3 Inserting a *synonym*

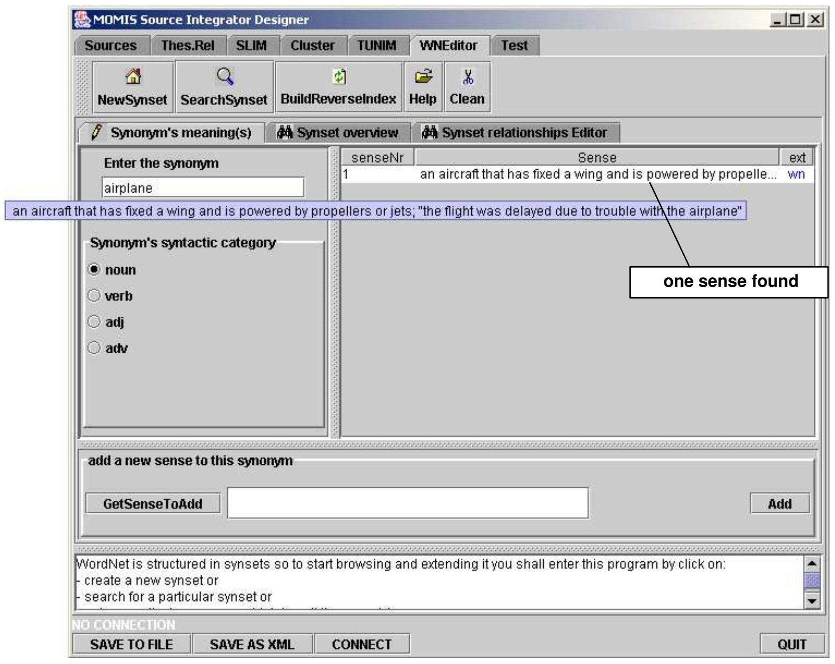
The designer who wants to check if a term or a lemma exists in the underlying reference ontology writes it in the field of the ***Synonym's meaning(s)*** panel obtaining by default all the related senses in the noun syntactic category.

Like is shown in figure 3.4 we insert for example the lemma *airplane*. If it already exists, all its current meanings are displayed; on the contrary case an error message appears. In the example case we obtain only one meaning.

The list of current associated senses is ordered from most to least frequently used with the most common sense put in the top position. Frequency of use is determined in WordNet by the number of times a sense is tagged in the various semantic concordance texts. Senses that are not semantically tagged follow the ordered senses. In particular all the synset_offset s in database files index.pos are output in sense number order with sense 1 first in the list.

For each founded element the sense number for the inserted lemma, the meaning definition and the creator's code of the bound between the lemma and this particular sense are reported.

The creator's code is an information extremely important since it gives the possibility to identify who has extended a synset, modifying its synonyms, or who has



Enter the synonym: airplane

Synonym's syntactic category: (●) noun ( ) verb ( ) adj ( ) adv

senseNr	Sense	ext
1	an aircraft that has fixed a wing and is powered by propelle...	wn

an aircraft that has fixed a wing and is powered by propellers or jets; "the flight was delayed due to trouble with the airplane"

```
WordNet is structured in synsets so to start browsing and extending it you shall enter this program by click on:
- create a new synset or
- search for a particular synset or
```

Figure 3.4: ***Synonym's meaning(s)*** panel

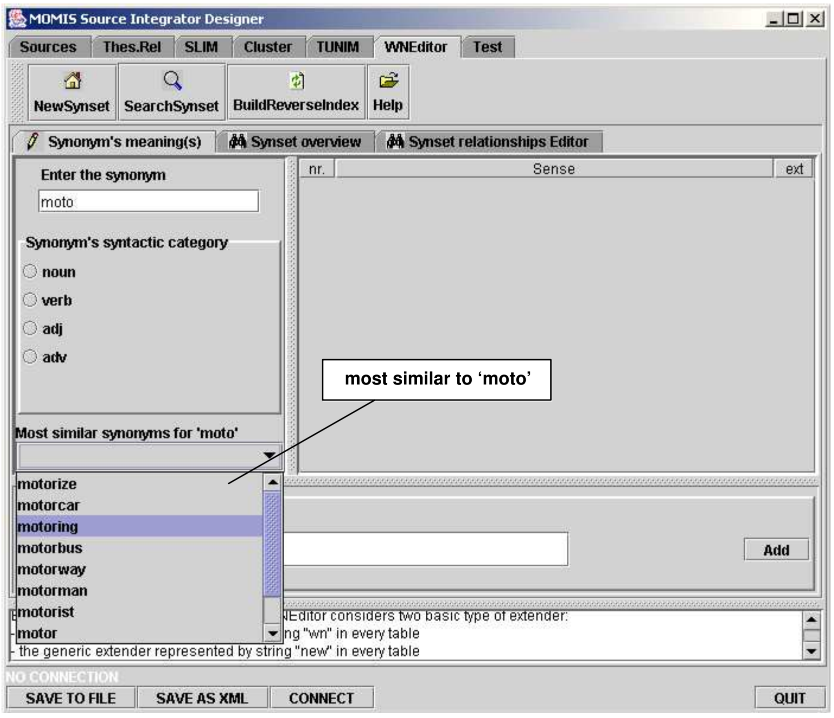
added a meaning to a specific lemma. In particular, when the bound is not created by a designer belonging to the official WordNet's group, the code associated to this extender is *new* and appears red colored.

WNEditor gives the possibility to cancel through a contextual menu a *new* created sense from the set of meanings currently displayed. Two important things should be noted in this case:

- we cannot remove an original WordNet sense associated to the particular synonym
- all the sense numbers are reassigned

The second point is the most important one since, as it will be shown in the following chapters, every couple like *(synonym, sense number)* must uniquely identify a synset in the reference ontology, in order to make any source annotation possible. On the other hand, reassigning sense numbers to a lemma could mean that all the pre–existing sources annotations made on those meanings must be reviewed to remain consistent with the underlying ontology.

Moreover, during this operation the user could receive a warning message in the particular case that the meaning to be cancelled is the only one associated to the lemma as well. This implementation choice is due to the impossible case of the existence of a floating word form in the ontology, i.e. a synonym without a sense.
This last assumption is not a limitation to the changes' possibilities since **WNEditor** allows the designer to modify a *new* meaning in every moment without turning to the sense's deletion and reinsert.



NEditor considers two basic type of extender:
ng "wn" in every table
- the generic extender represented by string "new" in every table

Figure 3.5: Exploiting Levenshtein distance in the ***Synonym's meaning(s)*** panel

Moreover, considering that the user is not requested to know exactly the word form of the lemma he wants to search, **WNEditor** is also able to execute an *approximate search of the particular lemma*.
A list of most similar existing terms to the inserted one appears to help the designer in the selection. Referring for example to figure 3.5 we have inserted the lemma "moto" that doesn't exist in the English language: **WNEditor** helps us in getting the right form by listing all most similar synonyms to the inserted one.

Going into details the similarity criteria implemented to reach this target is the Levenshtein distance whose definition is as follows: **Levenshtein distance (LD)**

is a measure of the similarity between two strings, which it can be referred to as the source string (s) and the target string (t). The distance is the number of deletions, insertions, or substitutions required to transform s into t.

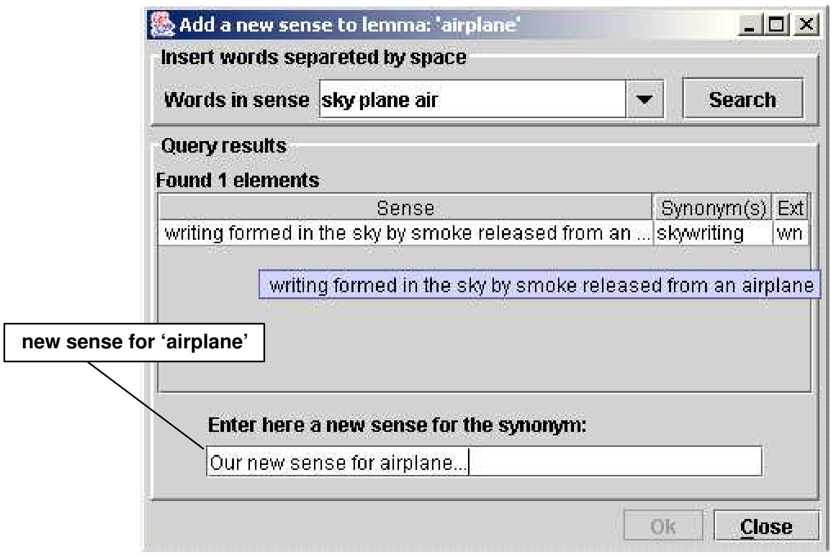
Making a simple example, if s is *test* and t is *test*, then LD(s,t) = 0, because no transformations are needed. The strings are already identical. If s is *test* and t is *tent*, then LD(s,t) = 1, because one substitution (change *s* to *n*) is sufficient to transform s into t. The greater the Levenshtein distance the more different the strings are. This method to compute the similarity between strings is named after the Russian scientist Vladimir Levenshtein, who devised the algorithm in 1965. This metric is also sometimes called *edit distance*.

Table 3.1 illustrates all the necessary steps to implement the edit distance algorithm.

Step	Description
1	Set n to be the length of s. Set m to be the length of t. If n = 0, return m and exit. If m = 0, return n and exit. Construct a matrix containing 0..m rows and 0..n columns
2	Initialize the first row to 0..n. Initialize the first column to 0..m
3	Examine each character of s (i from 1 to n)
4	Examine each character of t (j from 1 to m)
5	If s[i] equals t[j], the cost is 0. If s[i] doesn't equal t[j], the cost is 1
6	Set cell d[i,j] of the matrix equal to the minimum of: a. The cell immediately above plus 1: d[i-1,j] + 1. b. The cell immediately to the left plus 1: d[i,j-1] + 1. c. The cell diagonally above and to the left plus the cost: d[i-1,j-1] + cost
7	After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell d[n,m]

Table 3.1: The seven steps to perform the edit distance algorithm

Obviously we had to threshold in some way the edit distance value resulting between two terms, the input one and any other word form containing that string. Actually it was necessary to output only *most* similar strings to the given one: to perform this target the threshold value for the distance was assumed equal to the length of the inserted string.


Add a new sense to lemma: 'airplane'
Insert words separeted by space
Search
Query results
Ok
Close
new sense for 'airplane'

Words in sense: sky plane air

Found 1 elements

Sense	Synonym(s)	Ext
writing formed in the sky by smoke released from an ...	skywriting	wn

writing formed in the sky by smoke released from an airplane

Enter here a new sense for the synonym: Our new sense for airplane...

Figure 3.6: ***add a new sense to this synonym*** panel

The ***add a new sense to this synonym*** panel indicated in figure 3.6 gives the possibility to add a meaning to any existing searched lemma.

Referring to figure 3.6, the designer is able to extend the actual set of senses related to this current synonym by add to it an existing one or a new created one. In the first case the designer chooses a sense from a list of candidates obtained by inserting one or more keywords that are presumably in the definition of the meaning he is looking for; in the second one the designer is requested to write the English gloss explicitly. The designer has to click on the *GetSenseToAdd* button to perform these two operation.

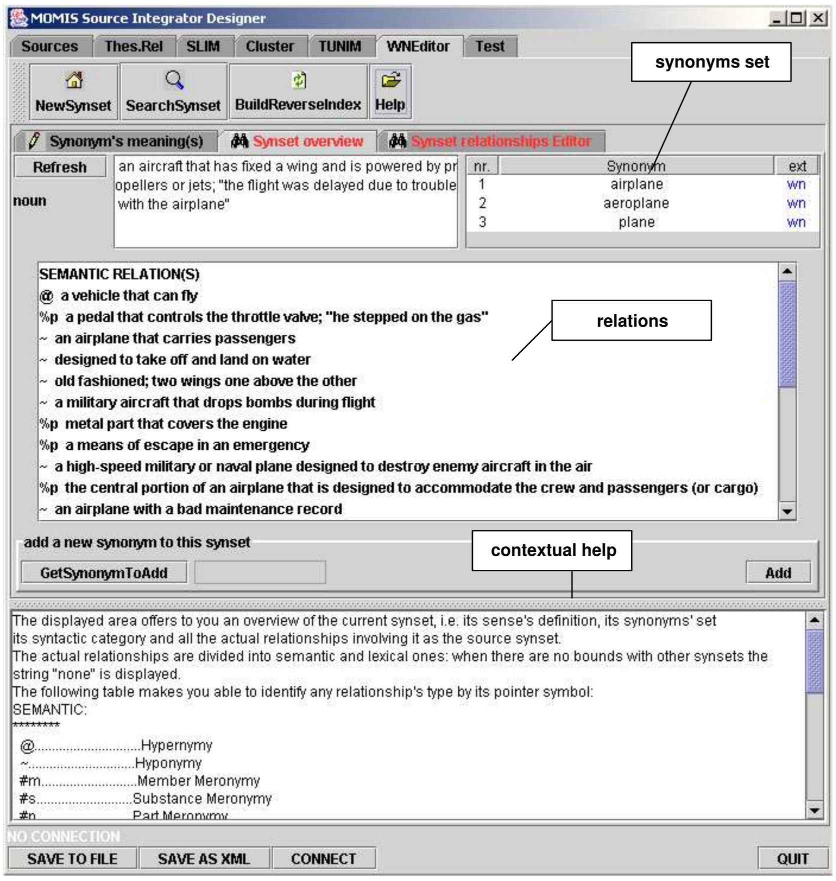
For example, as reported in figure 3.6, suppose we want to add a new sense to the lemma "airplane" but we don't have clear in mind what meaning. . . so we perform a search of it by the keywords: *sky, plane, air*. However, we are then not satisfied of the result set of meaning so we decide to insert a new one for lemma "airplane". The new added meaning is then displayed among the existing ones when we confirm our choice.

It should also be noted that by double clicking on any sense among them associated to the current lemma, we set it automatically as the **master synset** of our

work session. Moreover, supposing we have already select the **current synset** , by a contextual menu we can set one of our senses as the target synset.
In the first case ***Synset overview*** panel becomes active, while in the second one ***Synset relationships Editor*** gets the focus.

3.2.4 Having an overview of the master synset

When the ***Synset overview*** panel becomes active, showing its title in red color, the designer is able to view all properties of the selected current synset beyond its spatial collocation within the ontology.



an aircraft that has fixed a wing and is powered by pr
opellers or jets; "the flight was delayed due to trouble
with the airplane"

nr.	Synonym	ext
1	airplane	wn
2	aeroplane	wn
3	plane	wn

SEMANTIC RELATION(S)
@ a vehicle that can fly
%p a pedal that controls the throttle valve; "he stepped on the gas"
~ an airplane that carries passengers
~ designed to take off and land on water
~ old fashioned; two wings one above the other
~ a military aircraft that drops bombs during flight
%p metal part that covers the engine
%p a means of escape in an emergency
~ a high-speed military or naval plane designed to destroy enemy aircraft in the air
%p the central portion of an airplane that is designed to accommodate the crew and passengers (or cargo)
~ an airplane with a bad maintenance record

The displayed area offers to you an overview of the current synset, i.e. its sense's definition, its synonyms' set
its syntactic category and all the actual relationships involving it as the source synset.
The actual relationships are divided into semantic and lexical ones: when there are no bounds with other synsets the string "none" is displayed.
The following table makes you able to identify any relationship's type by its pointer symbol:
SEMANTIC:
****
@............................Hypernymy
~............................Hyponymy
#m..........................Member Meronymy
#s..........................Substance Meronymy
#p..........................Part Meronymy

Figure 3.7: ***Synset overview*** panel

Considering we have selected the **master synset** by double clicking on the first meaning of the lemma “airplane”: this sense will be the **current synset** we refer to in the following.

Referring to figure 3.7, the meaning's definition and its syntactic category are displayed in the top–left panel's corner. Near these data is the list of all the actual synonyms in the **current synset** : for each of them the order number within the synset, the name and the code of who has created the bound between that particular synonym and the **master synset** are reported.

By double clicking on one of the synset synonyms we set it automatically in the ***Synonym's meaning(s)*** obtaining all its actual senses in the syntactic category of the **master synset** .

Notice that we can also remove any new created synonym by exploiting a contextual menu on them. In this case all the lemma numbers for the **current synset** are reassigned.

Both existing lexical and semantic relations involving the **current synset** as the first member of a relation (i.e. the source synset) are displayed in the central area. Notice that when a relation is a symmetric or a reflexive one it compares only one time in the list, considering that the **current synset** remains the source synset of the relation, i.e. its first member.

For each displayed semantic relation the pointer symbol, indicating the relation's type, the target synset's gloss and all target synonyms are reported, while for each lexical one the **master synset** 's synonym involved in the relation, the pointer symbol, the dot noted string like *target synset's gloss.target synset's synonym* and all the target synonyms are displayed.

Since the information about the type of a relation is displayed as internally represented, the designer can exploit an useful contextual help in which every symbol is associated to a relationship's name (see figure 3.7 for more details).

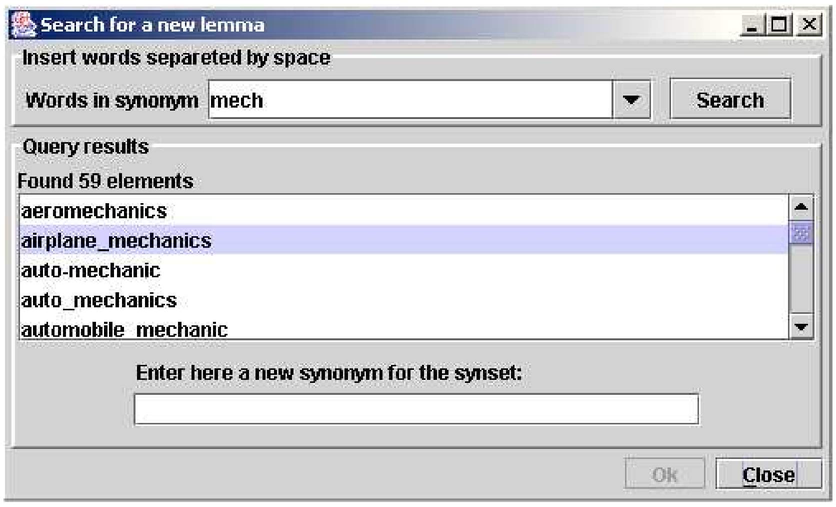
Giving to the designer the possibility to have an immediate view of all actual relations involving the **current synset** **WNEditor** allows a more flexible and easy WordNet browsing.

In particular, the designer who wants to modify the existing relations doesn't need to search for the target synsets involved: actually by double clicking on an element of the list the ***Synset relationships Editor*** panel becomes automatically active displaying the **master synset** as the first member of the selected relation and the related target synset as the second one. For further information about this panel, please see section 3.2.5.

If the designer wants to set one of the actual target synsets as the main synset, he can do this by exploiting the contextual menu *set as main synset*.
Each time the **current synset** 's actual state is modified by, for example, deleting some of its relations or adding to it one or more new synonyms, the designer can view the up–to–date state by clicking on the *Refresh* button.

We also give to the designer the possibility to edit the gloss of the actual **master synset** when it is not an original WordNet one. To confirm the new definition of the meaning the designer has to click on the *Update* button.

This operation seems a very simple one but the algorithm that commit the new sense has also to update the state of the reverse index, however this is done in a transparent way to the designer and takes some seconds depending on the change's entity. For more details refer to section 3.3.1.


Search for a new lemma
Insert words separeted by space
Words in synonym
Search
Query results
Enter here a new synonym for the synset:
Ok
Close

mech

Found 59 elements

aeromechanics
airplane_mechanics
auto-mechanic
auto_mechanics
automobile_mechanic

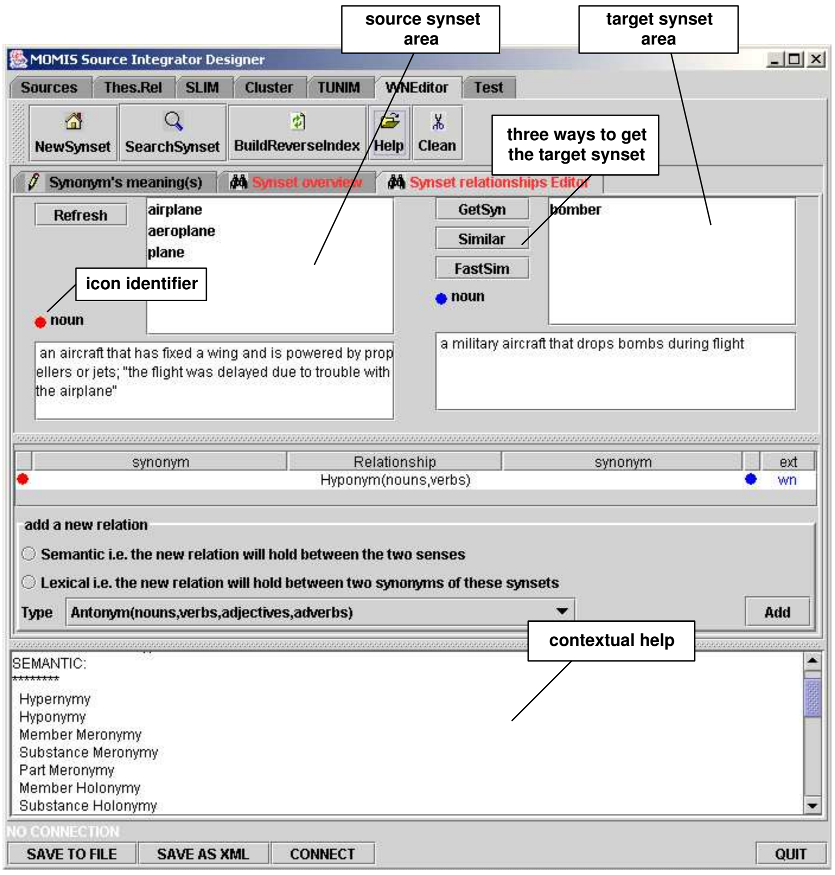
Figure 3.8: Extending the **current synset** 's synonyms set

The designer is also able to extend the **master synset** set of synonyms by adding to it an existing reference ontology's lemma or a new created one.
Referring to the form in figure 3.8 obtained by click on *GetSynonymToAdd* button, the designer can chose a lemma from a list resulting from a keywords based search, or he can write a new one in the appropriate field placed in the bottom of the panel.

For example, after inserting the keyword "mech" a list of similar synonyms is displayed. Then confirming one choice the just added synonym is placed among the preexisting ones with an *ext* field value equal to **new**.

3.2.5 Relation(s) editor

Through the ***Synset relationships Editor*** panel the designer is able to view, to analyze and to modify the actual state of relations involving the **current synset** as the source synset, i.e. the first member of any relation.



airplane
aeroplane
plane

bomber

● noun

an aircraft that has fixed a wing and is powered by propellers or jets; "the flight was delayed due to trouble with the airplane"

● noun

a military aircraft that drops bombs during flight

	synonym	Relationship	synonym		ext
●		Hyponym(nouns,verbs)		●	wn

Semantic i.e. the new relation will hold between the two senses

Lexical i.e. the new relation will hold between two synonyms of these synsets

Type Antonym(nouns,verbs,adjectives,adverbs)

```
SEMANTIC:
********
 Hypernymy
 Hyponymy
 Member Meronymy
 Substance Meronymy
 Part Meronymy
 Member Holonymy
 Substance Holonymy
```

Figure 3.9: ***Synset relationships Editor*** panel

Referring to figure 3.9 the north part of this panel is divided in two equal areas that are the *logic representation of the source synset and of the target synset*, respectively, from left to right.

As indicated in figure 3.9, the **master synset** chosen for this tutorial:

```
[ an aircraft that has fixed a wing and is powered by
  propellers or jets; "the flight was delayed due to
             trouble with the airplane" ]
```

is set in the left part while the target synset:

```
[ a military aircraft that drops bombs during flight ]
```

is set in the right part.
Notice that the target synset appears also in figure 3.7: we have double clicked on it making it automatically the target synset.
Then, what we expect from figure 3.7 is an hyponym relation, that is effectively displayed.

Notice also that for both the relation's members the actual synonyms' list, the syntactic category and an icon identifier (a red or a blue ball) are displayed. In the bottom area we can view all the existing relations between these selected synsets.

Considering now we want to change the target synset and search for another one to check any existing relation between the **current synset** and it. We can perform this search since the second member of any relation (i.e. target synset) is selectable as much as designer likes.
In particular, three different ways are available to find out from here another target synset:

- by an explicit search of the target sense's definition using one or more keywords based search
- by exploiting the semantic related to the current meaning considering all its words, i.e. the full method cited in 2.2.1
- by exploiting the semantic related to the current sense considering part of its words selectable by the designer explicitly, i.e. the partial method cited in 2.2.1

Due to their own relevance in this work, the three over mentioned methods are described in the next section (section 3.3).

Considering now that a target synset has already been selected then all actual relations between it and the **current synset** are automatically loaded. If no relations exist among these two members a warning message is displayed.

As figure 3.9 shows, to avoid the writing of both members' meanings the two icon identifiers are reported instead of them.

For each *semantic relation* only the icon identifiers, the relation's name and the code of who has created that bound are displayed, while for each *lexical relation* are also reported the two synonyms between which the relation holds.

Moreover, each relation created by the designer can be deleted. When deleting a reflexive or a symmetric relation, also the correspondent one is automatically cancelled.
Notice that *in a very moment, a synset could be isolated from the remaining ontology's network not being related to any other synset*.
As indicated in figure 3.9, the bottom area of this panel allows the designer to create new lexical or semantic relations involving the **master synset** as the first member and the selected target synset as the second member. To reach this target the designer is requested to select the relation's type, from a displayed list reporting all WordNet relations, and the relation's category, i.e. lexical or semantic.
The designer can exploits a contextual and useful help by click on *Help* button in the toolbar.

It is worth noting that each selected configuration is automatically checked by **WNEditor** to guarantee the respect of all the relations' rules introduced by the WordNet's official group.
When try to insert wrong relations, the designer will receive a clear explanation as an error message.
We remember also that to build a new lexical relation, the two synonyms to be related must be selected from each synset respectively.

3.3 Three methods for getting the target synset

According to the introduction in section 2.2 of chapter 2, here we deepen the implemented techniques to perform the search of the second member in a relation where the first one is the **current synset** . In this section, we will refer in particular to the exploitation of all the terms in each sense's definition from a semantic point of view.
Two different situations can arise during the target synset's search phase:

- the designer has rather clear in mind what is the sense's definition he is looking for and he doesn't consider much useful possible suggestions from the machine

- the designer has a less clearer view of the wished sense than he had in the previous situation and allows the machine to suggest him some solutions

In the first case the designer should perform a keywords based search of the target sense by clicking on *GetSyn* button. After viewing the result set of elements, the designer is able to set one of them as the actual target synset simply by double clicking on it.
In the second case **WNEditor** puts as disposal two buttons to perform a similarity search of the target synset, i.e. a search to get a set of synsets similar in a certain degree to the current one; the exploited algorithms implement the similarity functions cited in section 2.2.1.

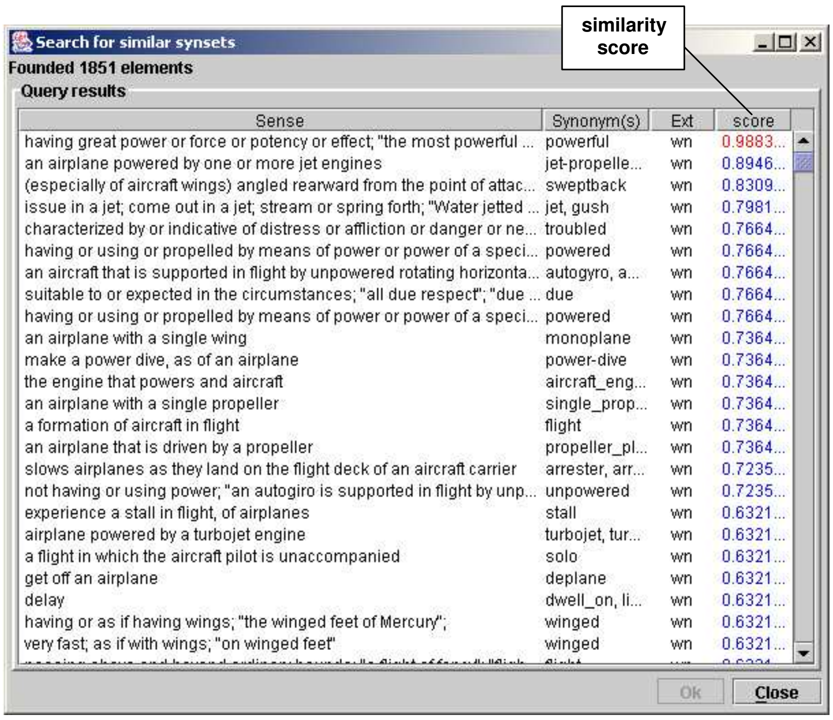
Through the *Similar* button we are able to perform the search by exploiting all the word forms existing in the current sense's definition after demorphing it and this is exactly the full method cited in section 2.2.1, while through the *FastSim* button we can exploit only a subset of the gloss' terms referred in the following as "keywords": this method is the partial one cited in section 2.2.1.
Concerning to these two last methods, as it is shown in the next section, for each suggested synset is also displayed a measure of the *similarity degree* between the current sense and the selected one.
Starting from this point it appears more clear how the designer can exploit all the suggestions coming from the machine.
Offering him the possibility to have a preview bounded to the most similar synsets, among all existing ones in the ontology, makes the building relations' phase more easy and flexible for the generic designer.

However, I would like to underline that *the similarity search technique remains an heuristic method and all the data it returns in output remain only suggestions that must be tested and evaluated by the designer.*

The main target related to the use of particular information retrieval techniques is to take the first step in the direction of a more larger exploitation of the semantics within the **MOMIS** integration system.

In section 3.3.1 a simple application example of a similarity search process is illustrated.

3.3.1 Searching most similar synsets



Founded 1851 elements

Query results

Sense	Synonym(s)	Ext	score
having great power or force or potency or effect; "the most powerful ...	powerful	wn	0.9883...
an airplane powered by one or more jet engines	jet-propelle...	wn	0.8946...
(especially of aircraft wings) angled rearward from the point of attac...	sweptback	wn	0.8309...
issue in a jet; come out in a jet; stream or spring forth; "Water jetted ...	jet, gush	wn	0.7981...
characterized by or indicative of distress or affliction or danger or ne...	troubled	wn	0.7664...
having or using or propelled by means of power or power of a speci...	powered	wn	0.7664...
an aircraft that is supported in flight by unpowered rotating horizonta...	autogyro, a...	wn	0.7664...
suitable to or expected in the circumstances; "all due respect"; "due ...	due	wn	0.7664...
having or using or propelled by means of power or power of a speci...	powered	wn	0.7664...
an airplane with a single wing	monoplane	wn	0.7364...
make a power dive, as of an airplane	power-dive	wn	0.7364...
the engine that powers and aircraft	aircraft_eng...	wn	0.7364...
an airplane with a single propeller	single_prop...	wn	0.7364...
a formation of aircraft in flight	flight	wn	0.7364...
an airplane that is driven by a propeller	propeller_pl...	wn	0.7364...
slows airplanes as they land on the flight deck of an aircraft carrier	arrester, arr...	wn	0.7235...
not having or using power; "an autogiro is supported in flight by unp...	unpowered	wn	0.7235...
experience a stall in flight, of airplanes	stall	wn	0.6321...
airplane powered by a turbojet engine	turbojet, tur...	wn	0.6321...
a flight in which the aircraft pilot is unaccompanied	solo	wn	0.6321...
get off an airplane	deplane	wn	0.6321...
delay	dwell_on, li...	wn	0.6321...
having or as if having wings; "the winged feet of Mercury";	winged	wn	0.6321...
very fast; as if with wings; "on winged feet"	winged	wn	0.6321...
[illegible]	[illegible]	[illegible]	[illegible]

Figure 3.10: Similarity search for the target synset

Consider we want to find out all the existing similar synsets to the current one: being the **current synset** already selected from the examples in the previous sections, it is sufficient to click on *Similar* button in ***Synset relationships Editor*** panel.

Figure 3.10 shows the resulting set of elements: for each of them we can estimate its *similarity* with the current synset by reading the meanings and the associated score values.

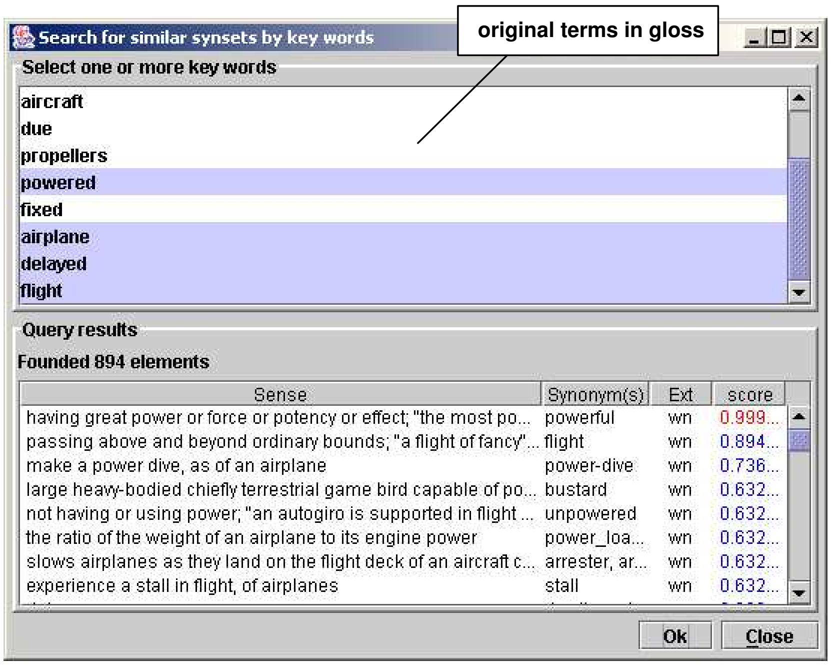
By clicking on this last table's header the founded synsets are ordered from the most similar to the least one, where that corresponding to the highest score is red colored.

Furthermore, by double clicking on one of them we set it automatically as the second member of any existing relation with the **master synset**.

It is worth noting that, after having removed all common english terms from the current sense's definition, this search task will take some seconds depending on the remaining number of words in the gloss.

To avoid any confusion and in the respect of a user friendly interface, **WNEditor** displays a progress bar informing the designer on the process' advancing state. However, **WNEditor** makes the designer able to interrupt this batch operation at every moment.

Suppose now we want to perform a more faster but less accurate search task, then we can use the **keywords based similarity search**.


Search for similar synsets by key words
original terms in gloss
Select one or more key words
Query results
Ok
Close

aircraft
due
propellers
powered
fixed
airplane
delayed
flight

Founded 894 elements

Sense	Synonym(s)	Ext	score
having great power or force or potency or effect; "the most po...	powerful	wn	0.999...
passing above and beyond ordinary bounds; "a flight of fancy"...	flight	wn	0.894...
make a power dive, as of an airplane	power-dive	wn	0.736...
large heavy-bodied chiefly terrestrial game bird capable of po...	bustard	wn	0.632...
not having or using power; "an autogiro is supported in flight ...	unpowered	wn	0.632...
the ratio of the weight of an airplane to its engine power	power_loa...	wn	0.632...
slows airplanes as they land on the flight deck of an aircraft c...	arrester, ar...	wn	0.632...
experience a stall in flight, of airplanes	stall	wn	0.632...

Figure 3.11: Similarity search for the target synset

To perform this operation, the designer is requested to select only some words of the current meaning's definition.

Referring to figure 3.11 in the top of the area all the master sense's definition words are listed except the most common English terms (stopwords).

Notice that in the case the current gloss is composed only by stopwords, all its original terms will be reported in this list, as better explained in section 3.3.2.

In the example we have selected only the following words: "powered", "airplane", "delayed", "flight". The resulting set of elements is of 894 synsets: as in the previous method we can read a score number for each of them and by double

clicking on one element we automatically set it as the current target synset.
Notice that the designer is able to perform a multi selection intervals in the terms list (as reported in figure 3.11) by using a keyboard–mouse combination (CTRL + ALT + mouse event).

All the functions studied and implemented to perform the similarity search are very selective, since the most part of the returned meanings are associated to very low score values.
However, being part of this work an experimental evaluation on their results, we have stated to not threshold through the GUI interface the resulting scores, in order to make the designer aware of these functions selectivity.

3.3.2 The reverse index exploited by the similarity functions

Even if all implementation details are deeply explained in the chapter 4, we would like to introduce here the auxiliary data structure exploited by the similarity search task, implemented as a *reverse index*.
To build the reverse index we had to perform a two steps operation:

1. building the set of all terms used by the actual reference ontology to define the senses' glosses
2. indexing of terms. Each element of the set obtained at point1 is associated with a list containing all the meanings in whose definitions the current term appears one or more times

In this way it is possible, given a term, to obtain all the synsets that actually contain it in their definitions.
To perform the operation at point1 each sense's definition within the ontology is demorphed in a three phases process:

1. the meaning's gloss is separated into individual words
2. the previous output is compare to a list of the most common English stop-words (named the stoplist). All terms that appear in that list are removed and all duplicates are kept in order to compute the correct gloss' length
3. all duplicates in the previous output are removed. Each remaining word is submitted to a stemming process. The final result is the *cleaning* of the beginning definition

It is now clear *why the designer is requested to update and create meanings using English terms: actually to guarantee a correct working process for the similarity search it is necessary to make use of an English stop list and of a stemming operation based on the English language*.

Moreover, to enjoy up–to–date similarity search's results it should need to maintain up–to–date the reverse index. To reach this target, as will be explained in chapter 4, every time the designer modifies or inserts a new meaning the related changes must be committed on the reverse index.

Referring to the figure 3.1, the designer can build from scratch the reverse index as he likes, by clicking on the *BuildReverseIndex* button in the toolbar. This task may takes at least 12 minutes, depending on the size on the reference ontology in that particular moment.

Having now clear in mind that the similarity search sets its basis on the over mentioned string cleaning operation and taking into consideration that there are at least 99600 different senses within the ontology, it will probably appear more right the choice of eject the use of term weighting's techniques, mentioned in section 2.2.
On the contrary case, not only the already mentioned strings' cleaning operation, but also the overall computation of the terms' weights would have been necessary for each similarity search task.

Chapter 4

The WNEditor tool: software architecture

This chapter illustrates all non visible software specifications concerning to the developed library underlying the application GUI, with a panoramic view of the exploited instruments.
The choice of used software tools was made considering *code's portability* and resulting environment's robustness. It must be considered the primary role played by these aspects using an information integration system, **MOMIS**, freeware and portable on each platform.
To reach these targets, some tests were made on different software tools combinations. All the developed code is available online at **http://sparc20.ing.unimo.it/**.

4.1 How WNEditor exploits WordNet

The first problem to face was how to use WordNet's data files to be able to extend the WordNet ontology itself.
The first explored solution (came to my mind but later rejected) made use of ASCII files to store the WordNet's extensions. This choice would be too much heavy due to the complexity of the data structures resulted to keep the relations among original WordNet's files and any new extension.

At the end we decided to adopt the *relational technology* to represent WordNet, creating a database named *momiswn*. This is a classic way of storing data, but at the same time it is an innovative way to represent an ontology.

The following step set the basis for the database logic and physic project and for

the choice of the DBMS application to implement it.
The resulting database should have been able at the beginning to contain only the WorNet lexicon ontology without any extension. To deepen the technical aspects please refer to the following sections.

4.1.1 Development of the database *momiswn*

Starting from the WordNet's structure built on a data file and an index file for each syntactic category (see chapter 1 for further explanations) the database project was required to extrapolate all the needed information from these files.

In order to guarantee a sharp distinction between the original WorNet's data and the extended ones, database *momiswn* should have contained the information about the owner of a specific modification on the reference ontology.
This particular information about any extender not only allows us to distinguish between different ontology extensions but also makes us able to guarantee the security on the database accessing.
It was intentionally avoided to create a controlled access to the database most due to two reasons:

- the used tool impossibility to manage a set of extenders dynamically, while the best solution is to delegate the whole access control task to the DBMS directly

- the permission of deal with concurrent inconsistent extensions of the same reference ontology is not a problem in a local perspective where our **MOMIS** system is isolated from the world, but, when we think about a distributed cooperation among more integration systems and then among more inconsistent ontology's extensions, it would seem a better solution to exploit only one nominal extender for each reference ontology involved, i.e. for each system

The actual database *momiswn* contains two types of extender, the original, named *wn*, and the generic one, called *new*.

At the beginning the database was built using Hypersonic SQL tool available online at **http://hsqldb.sourceforge.net/internet/hSql.html**.
This DBMS is not able to manage directly the foreign keys and eventual SQL store procedures, on the other hand:

- it is an open source java database that results completely portable

- it exploits standard SQL syntax and a JDBC interface
- it is compact in-memory (less than 100 KB)
- it is free to use and re-distribute

Moreover, Hypersonic SQL has got three operating modes:

- In-Memory (non-persistent, using the memory only)
- Standalone (with logging to disk, and maybe data written to disk)
- Client/Server (multiple computers/applications can access the same database)

Using the Hypersonic SQL DBMS, all tables can be created so that the records are stored on disk and only some records are cached in memory.
This is done by creating tables using 'CREATE CACHED TABLE' instead of 'CREATE TABLE'. This feature allows to use big tables where the records need too much space to fit into memory.
Indexes of cached tables are also saved to disk, so the size of the database is not limited by the main memory. However access to cached tables is slower than access to the ones already in memory.

Although these positive features we had some *OutOfMemory Exception* problems in the database's loading phase that brought us to the conclusion to opt for another DBMS.

In this critical moment we had to select a different DBMS maintaining at the same time the use of the tool named *Torque* to make the application able to exploit the underlying database's resources.
The choice would be fallen on the last *SAP* DBMS product named *SapDb* but during the code writing this tool wasn't supported by Torque yet.

But what is Torque exactly?
Torque is a persistence layer that generates all the database resources required by current application and includes a runtime environment to run the generated classes. Torque was developed as part of the Turbine Framework in Jakarta Project and it is available online at **http://jakarta.apache.org/turbine/torque/**.

In the end the chosen DBMS was MySQL, completely tested by Torque's developers and last but not least an open source and freeware tool.

4.1.2 How *Torque* works

Torque uses a single *XML database schema* to generate:

1. the SQL for the target database
2. Torque's Peer-based object relational model representing the XML database schema
3. an HTML document describing the database can be generated to obtain a browseable version of the database schema

Torque can also exploit an ant build file (build-torque.xml) which could be added to the project. *Apache Ant* is a Java based build tool, it is kind of like *make* without *make*'s wrinkles. *Apache Ant* is available online at **http://jakarta.apache.org/ant/**. It defines the following targets:

- **sql** generates SQL source from an XML schema describing a database structure
- **doc** generates html or xml documentation for xml schemas
- **create-db** generates simple scripts for creating databases on various platforms
- **datadtd** generates data DTD from an XML schema describing a database structure
- **datadump** dumping data from db into XML
- **datasql** generates SQL source from an XML data file
- **jdbc** generates an XML schema of an existing database from JDBC metadata
- **om** generates output by using Velocity
- **insert-sql** inserts a SQL file into its designated database
- **sql2xml** generates an xml schema from an sql schema
- **id-table-init-sql** generates the initialization sql for the id table

Moreover Torque's runtime environment includes everything to use the generated ObjectModel/Peer classes and includes a jdbc connection pool. This means that Torque is able to manage a set of connections to the underlying database, for example in a multiple users environment.

In figures 4.1, 4.2 and 4.3 is reported part of the XML file **momiswn-schema.xml** used by Torque tool to generate automatically the SQL related to the database schema and all Java classes.
In particular the descriptions of the main tables, WN_SYNSET, WN_LEMMA and WN_RELATIONSHIP respectively are shown.

Then running Torque with Ant compile we obtain the generation of the *object model* which will produce Java source files to be used to represent the *momiswn* database.
These Java classes allow the developer to create, edit, delete, and select objects that represent rows in database's tables.
Moreover, as already mentioned, Torque will generate all SQL code needed to create the database from scratch in a classical way.

The generated object model consists of **four classes for each table in the schema**.
For example, the *WN_EXTENDER* table, defined in *momiswn-schema.xml*, will result in the following four classes:

WnExtender
WnExtenderPeer
BaseWnExtender
BaseWnExtenderPeer

Where WnExtender and WnExtenderPeer are subclasses of BaseWnExtender and BaseWnExtenderPeer respectively.

The two Base classes (BaseWnExtender and BaseWnExtenderPeer) contain Torque generated logic and *should not be modified* by the developer since Torque will overwrite all made changes if it happen to generate the object model again via Ant.
Then, any business logic was added in the WnExtender and WnExtenderPeer classes, where WnExtender represents exactly an object row in this table.

```
<table name="WN_SYNSET" description="WN synset">
    <column
        name="WN_SYNSET_ID"
        required="true"
        primaryKey="true"
        autoIncrement="true"
        type="INTEGER"
        description="Synset Id"/>
    <column
        name="BYTE_OFFSET"
        required="false"
        type="INTEGER"
        description="Synset offset"/>
    <column
        name="SYNTACTIC_CATEGORY"
        required="true"
        type="INTEGER"
        description="Syntactic category"/>
    <column
        name="WORD_CNT"
        required="false"
        type="INTEGER"
        description="Total number of synonyms
        in this synset"/>
    <column
        name="GLOSS"
        required="true"
        type="LONGVARCHAR"
        description="Synset gloss"/>
    <column
        name="WN_EXTENDER_ID"
        required="false"
        type="INTEGER"
        description="Origin of this record (who is the extender)"/>
    <foreign-key foreignTable="WN_EXTENDER"
    name="WN_SYNSET_EXTENDER_FK" onUpdate="none" onDelete="none">
       <reference local="WN_EXTENDER_ID" foreign="WN_EXTENDER_ID" />
    </foreign-key>
</table>
```

Figure 4.1: Table WN_SYNSET as created in the file momiswn–schema.xml

```
<table name="WN_LEMMA" description="WN lemma">
    <column
        name="WN_LEMMA_ID"
        required="true"
        primaryKey="true"
        autoIncrement="true"
        type="INTEGER"
        description="Lemma Id"/>
    <column
        name="LEMMA"
        required="true"
        type="VARCHAR"
        size="254"
        description="Name of lemma"/>
    <column
        name="SYNTACTIC_CATEGORY"
        required="true"
        type="INTEGER"
        description="Syntactic category"/>
    <column
        name="SENSE_CNT"
        required="false"
        type="INTEGER"
        description="Nr. of senses of this lemma"/>
    <column
        name="WN_EXTENDER_ID"
        required="false"
        type="INTEGER"
        description="Origin of this record(who is the extender)"/>
    <unique name="UNIQUE_LEMMA_SYNTACTIC_CATEGORY">
          <unique-column name="LEMMA"/>
          <unique-column name="SYNTACTIC_CATEGORY"/>
    </unique>
    <foreign-key foreignTable="WN_EXTENDER"
        name="WN_LEMMA_EXTENDER_FK"
        onUpdate="none" onDelete="none">
       <reference local="WN_EXTENDER_ID"
        foreign="WN_EXTENDER_ID" />
    </foreign-key>
</table>
```

Figure 4.2: Table WN_LEMMA as created in the file momiswn–schema.xml

```
<table name="WN_RELATIONSHIP" description="Relationship
  between synsets">
    <column
        name="WN_RELATIONSHIP_ID"
        required="true"
        autoIncrement="true"
        primaryKey="true"
        type="INTEGER"
        description="WnRelationship Id"/>
    <column
        name="WN_SOURCE_SYNSET_ID"
        required="true"
        type="INTEGER"
        description="Source synset Id"/>
    <column
        name="WN_TARGET_SYNSET_ID"
        required="true"
        type="INTEGER"
        description="Target synset Id"/>
    <column
        name="WN_SOURCE_LEMMA_NUMBER"
        required="true"
        type="INTEGER"
        description="Number of lemma in the source synset
        in the case of lexical relationship.
        If 0 then this is a semantic relationship"/>
    <column
        name="WN_TARGET_LEMMA_NUMBER"
        required="true"
        type="INTEGER"
        description="Number of lemma in the target synset
        in the case of lexical relationship.
        If 0 then this is a semantic relationship"/>
    <column
        name="WN_RELATIONSHIP_TYPE_ID"
        required="true"
        type="INTEGER"
        description="Relationship type Id"/>
    <column
        name="WN_EXTENDER_ID"
        required="false"
        type="INTEGER"
        description="Who made this relationship between
        these synsets or their lemmas"/>
    <foreign-key foreignTable="WN_SYNSET"
    name="WN_RELATIONSHIP_SOURCE_SYNSET_FK" onUpdate="none" onDelete="none">
       <reference local="WN_SOURCE_SYNSET_ID" foreign="WN_SYNSET_ID" />
    </foreign-key>
    <foreign-key foreignTable="WN_SYNSET"
    name="WN_RELATIONSHIP_TARGET_SYNSET_FK" onUpdate="none" onDelete="none">
       <reference local="WN_TARGET_SYNSET_ID" foreign="WN_SYNSET_ID" />
    </foreign-key>
    <foreign-key foreignTable="WN_RELATIONSHIP_TYPE"
    name="WN_RELATIONSHIP_TYPE_FK" onUpdate="none" onDelete="none">
       <reference local="WN_RELATIONSHIP_TYPE_ID"
       foreign="WN_RELATIONSHIP_TYPE_ID" />
    </foreign-key>
    <foreign-key foreignTable="WN_EXTENDER"
    name="WN_RELATIONSHIP_EXTENDER_FK" onUpdate="none" onDelete="none">
       <reference local="WN_EXTENDER_ID" foreign="WN_EXTENDER_ID" />
    </foreign-key>
</table>
```

Figure 4.3: Table WN_RELATIONSHIP in the file momiswn–schema.xml

The Peer classes (i.e. WnExtenderPeer and BaseWnExtenderPeer) wrap their associated database tables and provide static methods to manipulate those tables such as doSelect, doInsert, and doUpdate.
Moreover Peers make use of a DatabaseMap class that holds internal data about the relational schema. It is used internally by Peers to discover information about the database at runtime.
There is exactly one DatabaseMap for each relational database that developer has to connect to. DatabaseMaps are constructed by classes called MapBuilders.

Data Objects (i.e. WnExtender and BaseWnExtender), on the other hand, wrap individual rows within those tables and provide getters/mutators methods for each column defined in those tables as well as the *save()* method.

Both Peer and Data Objects have a one-to-one mapping to a table defined in the database schema.

To query the database exist the so called Criteria objects.
They are an abstraction of the criteria of an SQL query: so a query to database can be done both with a criteria object and a raw SQL query.
In particular all written code to manipulate database's objects uses frequently Criteria objects.

4.1.3 E/R schema for the database *momiswn*

In figure 4.4 is shown the E/R schema of the database *momiswn*.
In particular, as file *momiswn-schema.xml* reports, we decided to exploit the *autoIncrement method*, put at disposal by Torque, to let all tables' primary keys be generated by MySQL automatically.
Every primary key is always represented by a string like *WN_tableName_ID* where *tableName* is the owner table.

The following paragraphs are a summarized description of all the tables in figure 4.4.

Table *WN_EXTENDER*

This table contains all the extenders that can actually modify the WordNet ontology. Each extender has a name (field *NAME*) and a description (field *DESCRIPTION*), that is for example the URL of his web site. As over mentioned, *momiswn*

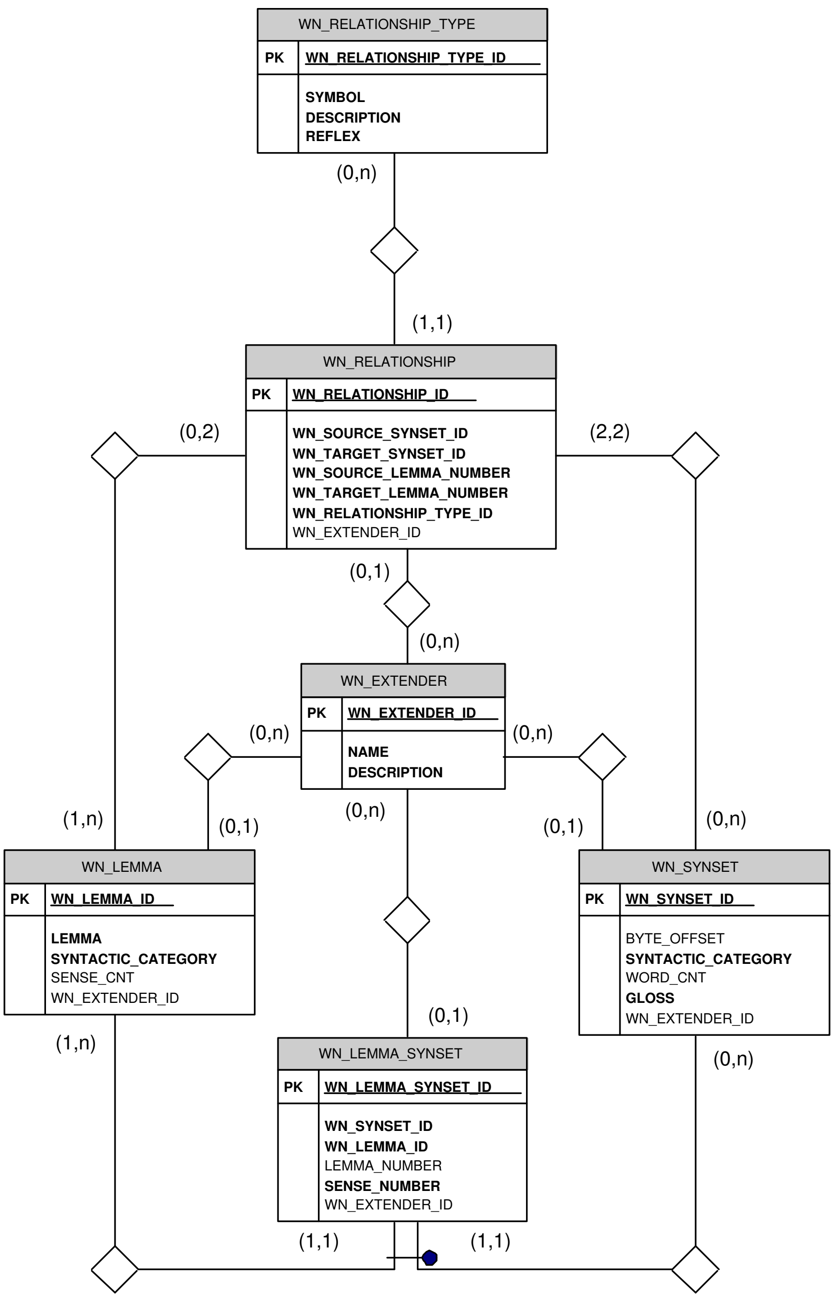


Figure 4.4: Database momiswn's E/R schema

contains two categories of possible extenders, those belonging to the WordNet official group, named *wn* and those not, called *new*.

Table *WN_LEMMA*

This table contains all the synonyms actually present in the reference ontology. Each synonym has a name (field *LEMMA*), the belonged part of speech (field *SYNTACTIC_CATEGORY*), the actual number of senses (field *SENSE_CNT*) and the creator's identifier (field *WN_EXTENDER_ID*).

Table *WN_SYNSET*

This table contains all the synsets existing in the reference ontology. According to the WordNet's synset organization each synset is identified in *momiswn* by the couple *(byte_offset, syntactic category)*. The *BYTE_OFFSET* field can however be null for all new synsets, i.e. those added to the ontology by the generic extender.
Moreover, each ontology synset should have a definition of its sense (field *GLOSS*), a counter starting from 1 indicating the number of synonyms it actually contains (field *WORD_CNT*) and its creator's identifier (field *WN_EXTENDER_ID*).

Table *WN_LEMMA_SYNSET*

This table represents the membership relation that holds between an ontology's synset and all its actual synonyms. In particular, each bound contains the synset's numeric identifier (field *WN_SYNSET_ID*), the identifier of a particular synonyms (field *WN_LEMMA_ID*), the order number of this synonym within the synset, starting from 1 (field *LEMMA_NUMBER*), this sense number for that synonym (field *SENSE_NUMBER*), starting from 1, and the bound's creator identifier (field *WN_EXTENDER_ID*).
It is worth noting that a synset that contains any synonyms won't be contained in this table.

Table *WN_RELATIONSHIP_TYPE*

This table contains all possible type of relations defined in the WordNet lexical ontology. To have a more specific description of WordNet's relations categories, please refer to chapter 1.
Each relation type has a machine readable symbol (field *SYMBOL*), a description to be understood by humans (field *DESCRIPTION*) and a flag (field *REFLEX*) indicating if the current type of relation is a reflexive one.

Table *WN_RELATIONSHIP*

This table represents the bound between any pair of synsets that are in relation. In particular, it is structured according to the definition of a relation pointer given in the WordNet lexicon ontology. To have a more specific description of pointer's definition, please refer to chapter 1.4.

Each relation must contain the identifiers of both the source and the target synset
field *WN_SOURCE_SYNSET_ID*
field *WN_TARGET_SYNSET_ID*

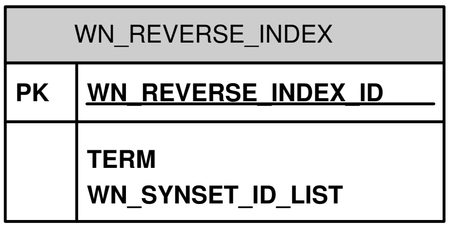
the identifier of the current relation's type
field *WN_RELATIONSHIP_TYPE_ID*

and two order numbers to determine the synonyms in source and target synset among with the relation holds
field *WN_SOURCE_LEMMA_NUMBER*
field *WN_TARGET_LEMMA_NUMBER*

It is worth noting that if and only if current relation is a lexical one the two order numbers are different from zero. At last every relation has to report its creator's identifier (field *WN_EXTENDER_ID*).

Table *WN_REVERSE_INDEX*

The table *WN_REVERSE_INDEX* shown in figure 4.5 was created as an auxiliary data structure to be exploited by the similarity search algorithms, as already introduced in section 3.3.2.



WN_REVERSE_INDEX	
PK	**WN_REVERSE_INDEX_ID**
	TERM **WN_SYNSET_ID_LIST**

Figure 4.5: Table WN_REVERSE_INDEX

This table implements the reverse index used by these algorithms to find out all most *similar* synsets to the current one. Its *xml* definition can be found in figure 4.6.

```
<table name="WN_REVERSE_INDEX"
  description="Reverse index built on the synset's glosses">
    <column
        name="WN_REVERSE_INDEX_ID"
        required="true"
        autoIncrement="true"
        primaryKey="true"
        type="INTEGER"
        description="WnReverseIndex Id"/>
    <column
        name="TERM"
        required="true"
        size="254"
        type="VARCHAR"
        description="Single term in one gloss (unique)"/>
    <column
        name="WN_SYNSET_ID_LIST"
        required="true"
        type="LONGVARCHAR"
        description="WnSynsetIds separated by , "/>
    <unique name="UNIQUE_TERM">
          <unique-column name="TERM"/>
    </unique>
</table>
```

Figure 4.6: Table WN_REVERSE_INDEX as created in the file momiswn–schema.xml

This data structure is a very well known indexing technique in information retrieval: each entry in this table is made of a term (field *TERM*) and a list of necessary data (field *WN_SYNSET_ID_LIST*) representing all the synsets identifiers in which definition this term appears, each synset gloss' length and the position(s) of the term within the gloss.

The state of the reverse index identifies at every moment the set of terms used within the reference ontology to build the sense's definitions.

It is worth noting that every time a designer modifies the current synset's gloss, the reverse index is consequently updated.

In particular there are four ways to update the reverse index according to its pre–existing state:

1. if the designer adds a term in the current synset's gloss that doesn't exist in the reverse index yet, the developed algorithm creates the new table's entry like (new inserted term, current synset's identifier)
2. if the designer adds a term in the current synset's gloss that already exists in the reverse index, the developed algorithm inserts current synset's identifier in the entry corresponding to this added term
3. if the designer deletes a term in the current synset's gloss that is associated to at least one another synset, then the developed algorithm deletes the bound between this term and the current synset's identifier
4. if the designer deletes a term in the current synset's gloss that is associated to this synset only, then the developed algorithm deletes this term's entry from table

Please remember that this updating process is automatically started every time the designer of the extension process confirms the changes introduced about the current synset's gloss.

This implementation's choice was made to guarantee better performance of the similarity search's process given that it is founded on reverse index's actual state.

4.1.4 Loading the database *momiswn*

Exploiting the object model generated by Torque, it is possible to insert data in *momiswn* using only Java code.

Remembering that object model associates four classes to each database table, consider for example the case of the table WN_SYNSET. The four corresponding Java classes are reported in figure 4.7.

The two *Base classes*, BaseWnSynset and BaseWnSynsetPeer, contain Torque-generated logic and they should not be modified since Torque will overwrite any change if it happen to generate the object model again.

This property is valid for all generated *Base classes*, like it's reported in section 4.1.2. Consequently, the whole business logic that had to add, was placed in WnSynset and WnSynsetPeer classes.

From my point of view, this coding did by hand allowed the most flexible solution in querying, updating and loading the database.

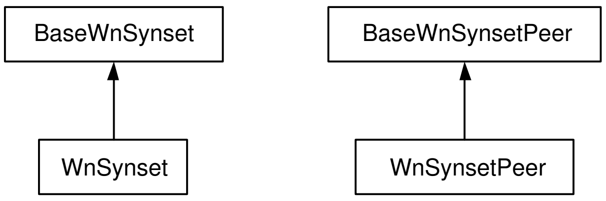


Figure 4.7: Generated Java Objects corresponding to table WN_SYNSET

All code needed to perform the database loading is implemented in the file *GlobalLoader.java*.

The table 4.1 shows for each database table, the corresponding Java object and the time (with the format hour:minutes) requested to load it on a Windows2000 workstation with 256MB of RAM memory and a processor frequency equal to 1.6 GHz. The tables are ordered like in the loading process.

Nr.	Table Name	Java Object	Time
1	WN_EXTENDER	WnExtender	0:0
2	WN_LEMMA	WnLemma	0:15
3	WN_SYNSET	WnSynset	0:7
4	WN_LEMMA_SYNSET	WnLemmaSynset	0:26
5	WN_RELATIONSHIP_TYPE	WnRelationshipType	0:0
6	WN_RELATIONSHIP	WnRelationship	8:0
7	WN_REVERSE_INDEX	WnReverseIndex	0:12

Table 4.1: Correspondences between tables and Java Objects

To load the whole WordNet lexical ontology in *momiswn* all its data and index files had to be parsed. To do so I modified the preexisting library's java files *mDataRecord.java* and *mIndexRecord.java* whose task is to parse each line of data and index files respectively.

To load table WN_SYNSET all WordNet's data files, i.e. NOUN.dat, VERB.dat, ADJ.dat and ADV.dat, were used, while to load table WN_LEMMA all index files, i.e. NOUN.idx, VERB.idx, ADJ.idx and ADV.idx, were parsed. To load table WN_RELATIONSHIP both the data and the index files were exploited, actually the most time consuming loading is associated to this particular table.

To have a more specific description of these mentioned files refer to chapter 1.

The designer can build from scratch the reverse index data structure as he likes, as already explained in chapter 3. This task may take at least 12 minutes, depending on the size of the extended ontology in that particular moment.

At the beginning, the loading algorithm used for table WN_REVERSE_INDEX, for each change in any tuple committed it directly on the database.
This way to perform the loading task took about 50 minutes on the same machine cited above.
On the other hand, since the designer of any ontology extension should often use this loading task through the GUI application **WNEditor**, 50 minutes seemed a too much waiting time.
In order to optimize this loading task, another solution was adopted considering a caching method for the whole table: only sometimes during the task the tuples are saved on the database.
To reach this target a main memory management based on the *Least Recently Used* method was implemented, exploiting the Java data structure, called *LRUMap*, in the Torque library package **org.apache.commons.collections**.
Follows the Java code defining our LRUMap; it was placed in the file *WnReverseIndexPeer.java*.

```
static class MyLRUMapForWnReverseIndex extends LRUMap {
    public MyLRUMapForWnReverseIndex(int size) {
       super(size);
    }
    protected void processRemovedLRU(Object key,
                                     Object value){
       WnReverseIndex wnReverseIndex = (WnReverseIndex)value;
       try {
          wnReverseIndex.save();
       } catch (Exception e) {
          String id = "" + wnReverseIndex
          .getWnReverseIndexId();
          System.out.println(" Error saving wnReverseIndex."
           + id );
          e.printStackTrace(System.out);
       }
    }
 }
```

We now briefly explain the working process exploiting this data structure.
Each row of the LRUMap table is indexed by a key element (unique in the table), which is any term of any sense definition in our case. To each term the correspondent WnReverseIndex object is associated, as shown in figure 4.8.

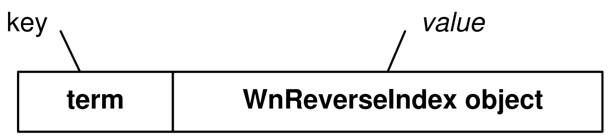


Figure 4.8: Object row of the LRUMap

For each term in any synset gloss the algorithm checks if the correspondent entry is already existing in cache, otherwise it tries to get the term from the database table directly. If term already exists in WN_REVERSE_INDEX then the algorithm reloads the related tuple in cache (it means this tuple was extracted from memory being the least recently used), on the contrary case it puts a new entry in the LRUMap.
Then by calling method *addSynsetId* on the *WnReverseIndex* object we ensure the changes on the database.
After parsing every synset in the reference ontology, the cache map contains the whole WN_REVERSE_INDEX table, since at every step any *value* grows. At the end, we have only to commit the whole table on the database transferring all cached data.
By exploiting this implementation choice an improvement of 76% in the reverse index loading time was reached.
Updating reverse index with a *right* frequency it is a very good way to exploit at best the similarity search process, then we suggest to any designer to consider this operation.

4.1.5 Statistics on the database *momiswn*

Following data tables represent a global view of *momiswn*'s state after loading the whole WordNet ontology, including also statistics on it.
Table 4.2 shows the distribution of records for each database's table. Consider that the dump file, generated with command
mysqldump momiswn > momiswn.dump
takes about 52 MB of memory.

Table Name	Records number
WN_EXTENDER	2
WN_LEMMA	129625
WN_SYNSET	99642
WN_LEMMA_SYNSET	173941
WN_RELATIONSHIP_TYPE	17
WN_RELATIONSHIP	238452
WN_REVERSE_INDEX	37208

Table 4.2: Distribution of all Database's records among its Tables

Table 4.3 shows the records distribution for each syntactic category in database's tables WN_LEMMA and WN_SYNSET.

Table Name	Records number	Syntactic category
WN_LEMMA	94503	noun
WN_LEMMA	10348	verb
WN_LEMMA	20199	adjective
WN_LEMMA	4575	adverb
WN_SYNSET	66025	noun
WN_SYNSET	12127	verb
WN_SYNSET	7003	adjective
WN_SYNSET	10912	adjective satellite
WN_SYNSET	3575	adverb

Table 4.3: Distribution of records for each syntactic category

Now doing some queries on command line environment offered by MySQL, it is possible to review WordNet from a statistic point of view. Doing a join between lemma table and lemma_synset table we obtain the total number of synonyms related to the noun synsets: query result is 116364. Supposing that each noun synset contains at least one synonym, the average number of synonyms for each noun synset is 1.7624233 (116364/66025). Doing the same operation for every syntactic category we obtain the table 4.4.

We can now compute the number of antonym relations involving a noun synset as the source one: query result is 1849. Doing the same query for each relation's type and for each syntactic category the table 4.5 is built.

Average number of synonyms	Synset syntactic category
1.7624233	noun
1.8201534	verb
1.2780237	adjective
1.9181634	adjective satellite
1.5885315	adverb

Table 4.4: Mean number of synonyms for each Database's synset

Note that to represent the relation's type also machine readable symbols are reported.

This following query allows to get the total number of relations involving a noun synset as the source one query result is 175201. Supposing that each noun synset is involved at least in one relation as the source synset, the average number of relation per noun synset is 2.653555 (175201/66025). Doing the same operation for every syntactic category that results in table 4.6.

The method *getMaxMin* in *WnReverseIndexPeer.java* was implemented to compute the maximum, the minimum, the mean and the variance value for all glosses' lengths in the database. To reach this target the algorithm considers only non stop words in every sense definition, except when the meaning is made only by stopwords. Table 4.7 reports the computed values.
Notice that, since the distribution variance value is lower than the 10% of the range of all possible values (48), we can assume that glosses' lengths have a less marked dispersal behaviour. This last consideration allows us to assume the mean value as a good index for the distribution.

Figure 4.9 is the histogram of the occurrences of glosses' lengths within the ontology: an important gathering can be seen around the mean value 6.162632.
Figure 4.10 is the same histogram but in a logarithmic scale: in this one also occurrences of the most long glosses can be seen.

It should be noted that the minimum value of gloss length is 1 since in a gloss made only by stopwords all the original terms are considered. Actually, the minimum number of non stop terms in a particular synset's gloss is zero when the definition is made only by stop words: in this case the demorphing algorithm must consider all the gloss original terms, submitting them to the stemming process.

Relation's name	symbol	noun	verb	adj	adj.sat.	adverb
Antonymy	!	1849	1031	4108	0	720
Hypernymy	@	66910	11536	0	0	0
Hyponymy	˜	66910	11536	0	0	0
Member meronymy	#m	11849	0	0	0	0
Substance meronymy	#s	709	0	0	0	0
Part meronymy	#p	6883	0	0	0	0
Member holonymy	%m	11849	0	0	0	0
Substance holonymy	%s	709	0	0	0	0
Part holonymy	%p	6883	0	0	0	0
Attribute	=	650	0	650	0	0
Entailment	*	0	427	0	0	0
Cause to	>	0	224	0	0	0
Also see	ˆ	0	631	2712	0	0
Verb group	$	0	523	0	0	0
Similar to	&	0	0	10974	10927	0
Participle of verb	<	0	0	90	0	0
Pertainymy	\	0	0	4021	0	3141

Table 4.5: Relations' distribution for each syntactic category

The figure 4.11 shows the demorphing algorithm *cleanString* in the Java class *StringCleaning*; this code restores all the original gloss terms when only stop-words have been found.

This implementation choice is due most to one reason: if we don't consider also these particular glosses we will never able to obtain them from a similarity search task, since their stopwords are not stored in the reverse index.

To compute the minimum and the maximum number of synsets and how many synsets on the average correspond to each term of the reverse index the method *getAverageListSize* in *WnReverseIndexPeer.java* was implemented.

Table 4.8 shows all the resulting values. It should be noted that the mean value represents the mean occurrence frequency of a term in the reference ontology definitions.

Average number of relations	Synset syntactic category
2.653555	noun
2.136389	verb
3.220762	adjective
1.001374	adjective satellite
1.08	adverb

Table 4.6: Mean number of relations for each Database's synset

Min	Max	Mean	Variance
1	48	6.162632	3.635852

Table 4.7: Statistic values about the glosses' lengths

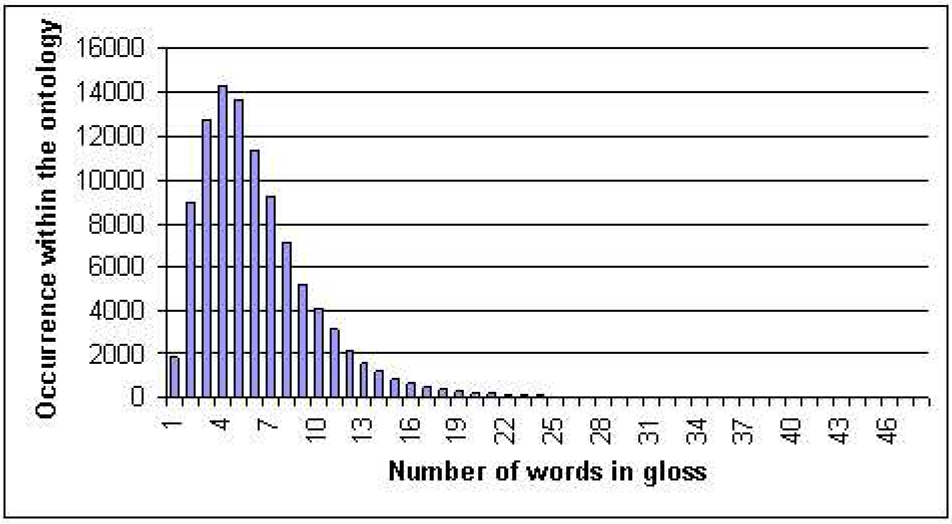


Figure 4.9: Histogram representing the occurrences of the glosses' lengths

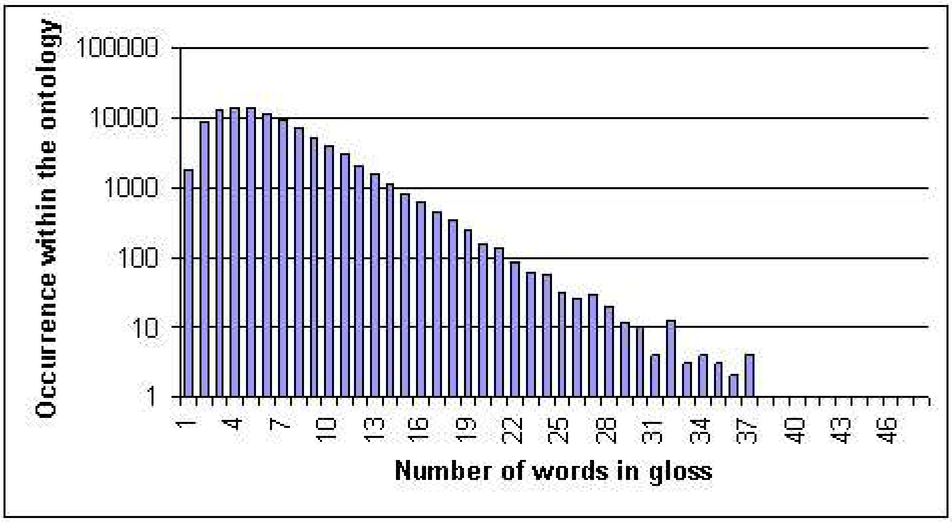


Figure 4.10: Logarithmic histogram representing the occurrences of the existing glosses' lengths

Min	Max	Mean
1	2964	15.8564

Table 4.8: Statistic values on the reverse index's terms

4.2 Evaluation on similarity search software

After illustrating the interaction between **WNEditor** and WordNet, we focus now on the implementation of the similarity functions introduced in section 2.2.1, assuming in particular the same notation given two definitions: y is the number of shared words between the two definitions and x is the number of words in the shorter one.

Table 4.9 is an analysis of the cases related to the first six values assumed by variable x, showing the relative range values of the three proposed similarity functions.

When a similarity search is performed on a demorphed synset's gloss g, three independent and fundamental aspects are to be considered as discriminating factors for any result set effectiveness:

1. considering the number of duplicated terms in g

```
/**
* Method to clean a given string
* @param stringToClean: string to be cleaned
*/
public void cleanString(String stringToClean)  {
  PorterStemmer stemmer = new PorterStemmer();
  try {
    _stopTable = makeStopTable(ENGLISH_STOP_WORDS);
    _gloss = stringToClean;
    StringTokenizer _strt =
    new StringTokenizer(_gloss,"' '\'\";,.:()!{}");
    while(_strt.hasMoreTokens()){
      _termsInGloss.addElement((_strt.nextToken()).toLowerCase());
    }
    /*
    * stopwords cycle:
    * if this word is not in the stop list it must be saved
    * in _termsToStem vector
    */
    for(int i=0;i<_termsInGloss.size();i++){
      if( (_stopTable.get(_termsInGloss.elementAt(i)))==null ){
        _termsToStem.addElement(_termsInGloss.elementAt(i));
      }
    }
    /*
    * the gloss is made only of stop words
    */
    if(_termsToStem.size()==0){
      for(int i=0;i<_termsInGloss.size();i++){
        _okTerms.addElement
        (stemmer.stem((String)(_termsInGloss.elementAt(i))));
      }
    }
    /*
    * stemming cycle
    */
    else{
      for(int i=0;i<_termsToStem.size();i++){
        _okTerms.addElement
        (stemmer.stem((String)(_termsToStem.elementAt(i))));
      }
    }
    _numOfWords = _okTerms.size();
    /*
    * _okTerms can contains also duplicate words:
    * build the hashmap _toDbMap
    */
    for(int i=0;i<_numOfWords;i++){
      _toDbMap.add(_okTerms.elementAt(i));
    }
  } catch (Exception e)  {
    e.printStackTrace(System.out);
  }
}
```

Figure 4.11: Method *cleanString*

var x	var y	$f_0(x,y)$	$f_1(x,y)$	$f_2(x,y)$
1	0	0	0	0
1	1	1	0.63	0.63
2	0	0	0	0
2	1	0.5	0.39	0.39
2	2	1	0.63	0.86
3	0	0	0	0
3	1	0.33	0.28	0.28
3	2	0.66	0.48	0.73
3	3	1	0.63	0.95
4	0	0	0	0
4	1	0.25	0.22	0.22
4	2	0.5	0.39	0.63
4	3	0.75	0.52	0.89
4	4	1	0.63	0.98
5	0	0	0	0
5	1	0.2	0.18	0.18
5	2	0.4	0.32	0.55
5	3	0.6	0.45	0.83
5	4	0.8	0.55	0.95
5	5	1	0.63	0.99
6	0	0	0	0
6	1	0.16	0.14	0.15
6	2	0.33	0.28	0.48
6	3	0.5	0.39	0.77
6	4	0.66	0.48	0.93
6	5	0.83	0.56	0.98
6	6	1	0.63	0.99

Table 4.9: Different cases for the three similarity functions according to the first six x values

2. considering the occurrences of terms within all the glosses to which g is compared

3. considering the relative positions of terms within all the glosses to which g is compared

Concerning to the first factor, if we state to maintain in gloss g also the duplicated terms, it is the same as assume that a duplicated term is *more important* than those which compare one time only within the gloss. This assumption is a very strong one since in this way *we relate the generic sense of a term to the whole meaning expressed by* g.

As far as the second aspect is concerned, if we count the occurrences of a term within any gloss that is compared to g, we assume in this way that the greater the occurrence is the more important this gloss is with the respect of g.
For example, if g contains the word *bread* and the algorithm finds a gloss that contains *bread bread bread* and any other term, then considering the occurrences of *bread* means that this will result in a three times match.
On the other hand, we could count only a one time match, i.e. we don't consider that *bread* compares more than one time; in this case we simply assume that, in g, *bread* is relevant as any other g term.
Another implementation choice could be to consider the occurrences only for the g terms that compare more than one time, while counting always one occurrence for any other g term. Then, referring to our example, if g contains *bread* two times, then we should consider the three occurrences; if g contains *bread* only one time, then we should consider one occurrence.

Concerning to the third factor it is not properly adapt to our immediate goal, i.e. to find out a set of similar meanings starting from g.
This fact is due most to this reason: we can exploit the *relative nearness between two query terms* when we treat them as keywords to be inserted in a search engine.
This implementation choice is part of the future work related to this thesis: when a search engine, exploiting our similarity functions, will be created to get the set of meanings containing specific query keywords, then we will be able to consider also the relative positions of those keywords within any founded synset.
This last operation will be a very critical task since not always who inserts a set of keywords also indicates a right and clever order.
However, it is very important to underline that the actual project of the database already considers the positions of term within a gloss, storing them in the reverse index data structure.

	one occurrence	more occurences
no duplicated	case 1	case 2
duplicated	case 3	case 4

Table 4.10: Four possible cases

According to the last considerations about the third factor, we want now to show the four possible output of a similarity search when we combine pair of the first two factors. In order to create an example, we have chosen an arbitrary gloss, referred in the following as the *query gloss/definition*. In particular, the ontology's gloss we will use is:

```
g.[ a landing (as the wheels touch the landing field);
especially of airplanes]
```

Notice that this meaning contains a duplicated term: "landing". The remaining terms after removing stopwords are six if we consider the duplicated one:

```
[ landing, wheels, touch, landing, field, airplanes ]
```

In table 4.10 the four cases we will analyze are reported. In the following, for each case, the resulting scores distribution and some significant returned glosses are shown.
Notice that, for every similarity search, we have exploited the function $f_2(x, y)$ and that the number of found elements is 1119 in each case.

Case 1

As we can see in figure 4.12, the most significant meaning with the respect to the query one is exactly that associated to the highest score.
Moreover, it can be said that all the elements associated to a score value less than 0.77 are more weakly related to the query sense. In particular all these meanings represent the 89% of the whole result set.

Case 2

As we can see in figure 4.13, the most significant meaning with the respect to the query one is exactly that associated to the highest score. Moreover, the glosses

Elements nr.	Percentage	score value
1	0.08	0.89
2	0.17	0.77
1	0.08	0.73
2	0.17	0.63
2	0.17	0.55
16	1.42	0.48
23	2.05	0.39
76	6.79	0.28
104	9.29	0.22
149	13.31	0.18
743	66.39	0.15

Table 4.11: Case 1: scores distribution

associated to scores like 0.89, 0.77, 0.73 are the same as those in the case 1. What is different now is that this case results are influenced by the occurrences of any term which appear also in the query meaning.

In particular, according to this computation way, two more glosses are here associated to higher score numbers than in the case 1; but none of them is significant with the respect of the query synset since the computed occurrences regard the terms “touch” and “field” which are less related to a landing…

As in the first case analyzed, it can be said that all the elements associated to a score value less than 0.77 are more weakly related to the query sense. In particular all these meanings represent the 84% of the whole result set.

Case 3

As we can see in figure 4.14, the most significant meaning with the respect to the query one is exactly that associated to the highest score.

What we immediately notice is that in this case the meanings containing the root of “landing” are associated to scores values greater than in the first two cases analyzed. This fact is due to the consideration of the duplicated term “landing”, in the query sense, two times when searching for matches.

However, case 3 remains rather selective as the previous ones concerning to the higher score values, but it causes a more dispersal distribution of meanings when

```
g.[  the approach to a landing field by an airplane ]

score.[ 0.89460075 ]

g.[  a landing in which all three wheels of the aircraft touch the
ground at the same time ]

score.[ 0.77686983 ]

g.[  a wheel located under the nose of an airplane that is part of
the plan's landing gear ]

score.[ 0.77686983 ]

g.[  an airplane that can land on or take off from water ]

score.[ 0.73640287 ]

g.[  wheel somebody or something ]

score.[ 0.63212055 ]

g.[  get off an airplane ]

score.[ 0.63212055 ]
```

Figure 4.12: Case 1: output glosses

it goes down with score values.

It can be said that the majority of meanings associated to scores less than 0.73 are weakly related to the query one; all these senses are the 95% of the whole result set.

Case 4

Even in this case, like in the third one, all the meanings containing the root of “landing” are associated to scores values greater than in the first two cases analyzed. This fact is due to considering the duplicated term “landing” two times when searching for matches.

However, as we can see in figure 4.15, the most significant meaning with the respect to the query one is now associated to the second highest score.

This new situation is due to the root of the term “landing” which compares three times in the gloss corresponding to the highest score; clearly this meaning is that corresponding to the highest occurrence of “landing” among them in the result set.

Elements nr.	Percentage	score value
1	0.08	0.89
5	0.44	0.77
1	0.08	0.73
3	0.26	0.63
4	0.35	0.55
65	5.8	0.48
23	2.05	0.39
76	6.79	0.28
103	9.2	0.22
147	13.13	0.18
691	61.75	0.15

Table 4.12: Case 2: scores distribution

Elements nr.	Percentage	score value
1	0.08	0.98
1	0.08	0.95
2	0.17	0.93
4	0.35	0.86
1	0.08	0.83
14	1.25	0.77
32	2.85	0.73
47	4.2	0.63
60	5.36	0.55
283	25.29	0.48
19	1.69	0.39
44	3.93	0.28
59	5.27	0.22
90	8.04	0.18
462	41.28	0.15

Table 4.13: Case 3: scores distribution

```
g.[  the approach to a landing field by an airplane ]

score.[ 0.89460075 ]

g.[  a landing in which all three wheels of the aircraft touch the
ground at the same time ]

score.[ 0.77686983 ]

g.[  a wheel located under the nose of an airplane that is part of
the plan's landing gear ]

score.[ 0.77686983 ]

g.[  make physical contact with, come in contact with; "Touch the
stone for good luck"; "She never touched her husband"; "The two
buildings almost touch" ]

score.[ 0.77686983 ]

g.[  the magnetic field of a planet; the volume around the planet
in which charged particles are subject more to the planet's
magnetic field than to the solar magnetic field ]

score.[ 0.77686983 ]

g.[  an airplane that can land on or take off from water ]

score.[ 0.73640287 ]

g.[  move along on or as if on wheels or a wheeled vehicle ]

score.[ 0.63212055 ]

g.[  get off an airplane ]

score.[ 0.63212055 ]

g.[  wheel somebody or something ]

score.[ 0.63212055 ]
```

Figure 4.13: Case 2: output glosses

```
g.[  the approach to a landing field by an airplane ]

score.[ 0.9816844 ]

g.[  an airplane that can land on or take off from water ]

score.[ 0.95021296 ]

g.[  a landing in which all three wheels of the aircraft touch the
ground at the same time ]

score.[ 0.93051654 ]

g.[  a wheel located under the nose of an airplane that is part of
the plan’s landing gear ]

score.[ 0.93051654 ]

g.[  landing an aircraft ]

score.[ 0.86466473 ]

g.[  get the lay of the land ]

score.[ 0.86466473 ]

g.[  make a forced landing ]

score.[ 0.86466473 ]

g.[  land tortoises ]

score.[ 0.86466473 ]

g.[  of airplanes: land on the underside without the landing gear
]

score.[ 0.8347011 ]

g.[  the path that is prescribed for an airplane that is preparing
to land at an airport; "the traffic patterns around O’Hare are
very crowded"; "they stayed in the pattern until the fog lifted" ]

score.[ 0.77686983 ]

g.[  an unscheduled airplane landing that is made under
circumstances (engine failure or adverse weather) not under the
pilot’s control ]

score.[ 0.77686983 ]

g.[  an airstrip outline with lights to guide an airplane pilot in
landing ]

score.[ 0.77686983 ]
```

Figure 4.14: Case 3: output glosses

Due to the last reason, this case seems to be not as accurate as the others. However, the next meanings are the same of those retrieved in case 3.

Moreover, it can be said that the majority of meanings associated to scores less than 0.73 are weakly related to the query one; all these senses are the 93% of the whole result set.

Elements nr.	Percentage	score value
1	0.08	0.99
1	0.08	0.98
2	0.17	0.95
21	1.87	0.93
4	0.35	0.86
1	0.08	0.83
16	1.42	0.77
32	2.85	0.73
48	4.28	0.63
60	5.36	0.55
293	26.18	0.48
19	1.69	0.39
44	3.93	0.28
58	5.18	0.22
89	7.95	0.18
430	38.42	0.15

Table 4.14: Case 4: scores distribution

The analysis of the similarity search on the particular chosen definition suggests that in each case a sharp distinction between two sets of meanings, the good ones and the others, can be seen.

Moreover, the last two cases seem to exploit a more fine granularity in select the meanings associated to the higher score values, retrieving more significant elements in the same score than the two previous methods.

In the end, considering the whole set of cases and the analyzed data not reported due to the lack of space, it can be said that the distinction among case 1, case 2 and case 3, case 4, is more important than one among case 1, case 3 and case 2, case 4.

This means that, for a better retrieval, we should consider also the duplicated terms

```
g.[  owning or consisting of land or real estate; "the landed
gentry"; "landed property" ]
score.[ 0.9975212 ]

g.[  the approach to a landing field by an airplane ]
score.[ 0.9816844 ]

g.[  an airplane that can land on or take off from water ]
score.[ 0.95021296 ]

g.[  yield crops, of land: "This land crops well" ]
score.[ 0.9592378 ]

g.[  a landing in which all three wheels of the aircraft touch the
ground at the same time ]
score.[ 0.93051654 ]

g.[  an emergency landing under circumstances where a normal
landing is impossible (usually damaging the aircraft) ]
score.[ 0.93051654 ]

g.[  the act of coming down to the earth (or other surface); "the
plane made a smoothe landing"; "his landing on his feet was
catlike" ]
score.[ 0.93051654 ]

g.[  a wheel located under the nose of an airplane that is part of
the plan's landing gear ]
score.[ 0.93051654 ]

g.[  landing an aircraft ]
score.[ 0.86466473 ]

g.[  get the lay of the land ]
score.[ 0.86466473 ]

g.[  make a forced landing ]
score.[ 0.86466473 ]

g.[  land tortoises ]
score.[ 0.86466473 ]

g.[  of airplanes: land on the underside without the landing gear ]
score.[ 0.8347011 ]

g.[  slows airplanes as they land on the flight deck of an
aircraft carrier ]
score.[ 0.77686983 ]

g.[  the path that is prescribed for an airplane that is preparing
to land at an airport; "the traffic patterns around O'Hare are
very crowded"; "they stayed in the pattern until the fog lifted" ]
score.[ 0.77686983 ]
```

Figure 4.15: Case 4: output glosses

in the starting gloss. Moreover, considering the occurrences of a term within the retrieved elements, can lead to a non significant meanings with the respect to the query one, but improves the retrieval returning new elements.

Mainly for these reasons, **WNEditor** actually implements the case 4 algorithm. It is also worth noting that, since the application of these functions is a new experiment in exploiting the semantic related to every ontology's meaning definition, it has to be analyzed in its whole totality, so we have stated to not threshold the resulting score values.

However, being the definition match a pure semantic match and not a structural one, it is very subjective in many cases to say if a returned meaning is effectively related to the query one: a designer can judge a synset a good one for the relation he's thinking about, but a bad one for another relation type. Thus, a much more serious effectiveness evaluation should firstly annotate some significant WordNet glosses and exploits a more than one expert's judgment in order to state if the retrieved definitions are effectively related to the query one.

Moreover, the need of more than one expert is an important requirement since more persons may give different importance (similarity score) to the same definitions.

Since the similarity search goodness test requires expert people also to annotate the WordNet glosses, we have presented a particular case study, extracted from a set of arbitrary glosses examined, in order to support our hypothesis, judging it a well formulated one.

Considering the *case 4* the most appropriate one from a theoretical point of view and in order to demonstrate that the choice of function $f_2(x, y)$ is the best one, table 4.15 shows the scores distribution when we perform the same similarity search by exploiting functions $f_0(x, y)$ and $f_1(x, y)$ respectively.

Both $f_0(x, y)$ and $f_1(x, y)$ return the same number of elements (1119) of function $f_2(x, y)$, moreover they create exactly the same scores distribution. Clearly this is not a surprising situation since both the functions depend on the ratio y/x.

Differently from $f_2(x, y)$, $f_0(x, y)$ and $f_1(x, y)$ have a *less fine granularity* in selecting the most similar definitions, then, all the meanings $f_2(x, y)$ associates to different and incremental scores, in the other two functions are associated to the max score value only.

For these reasons and considering the output data of the particular case study, $f_2(x, y)$ can be assumed as the *best similarity function among them proposed in this thesis*.

Elements nr.	Percentage	$f_0(x,y)$ score	$f_1(x,y)$ score
9	0.8	1	0.63
1	0.08	0.8	0.55
53	4.73	0.66	0.48
1	0.08	0.6	0.45
81	7.23	0.5	0.39
60	5.32	0.4	0.32
337	30.11	0.33	0.28
58	5.18	0.25	0.22
89	7.95	0.2	0.18
430	38.42	0.16	0.14

Table 4.15: Case 4: scores distribution for function $f_0(x,y)$ and function $f_1(x,y)$

Efficiency

As far as the efficiency of a similarity search is concerned, we would like to point out that we have worked with two different situations.

First we implemented a search algorithm exploiting an heavy access to the database: however, as we expected, this solution was not enough efficient for an interactive tool like **WNEditor**.

The second solution we have proposed, and then adopted, moves the problem of accessing the underlying ontology towards the database loading task.

Since the efficiency of the search task using the *partial* method depends strongly on *how many and which keywords* were selected from the current gloss, however being less heavy of a full search, we focus here on the case of a *full* search task.

Any similarity search task's efficiency is affected by different factors, among which:

- *all physics machine's characteristics (the processor's frequency, the main memory size, the operating system...)*
- *the number of words in the current meaning*
- *the reverse index's frequencies of all the terms in the query definition*

Clearly the first factor is a well defined one: the developed Java2 code has run on a Windows 2000 workstation characterized by a processor's frequency equal to 1.6 GHz and by 256 MB of RAM memory. Further an unloaded system was considered.

The remaining two factors are the real parameters of the efficiency test problem. We now illustrate the difficulties generated by the first adopted solution, illustrating the primary roles of the parameters, then showing the improvements brought from the second one.

First solution

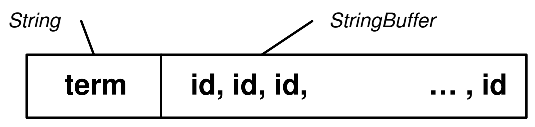


Figure 4.16: First Solution: reverse index record

In the first adopted solution we refer to a very poor reverse index data structure, where each row contains only the identifiers of the synset in whose definitions the particular term appears.
The figure 4.16 shows the record prototype where to each term corresponds a comma separated *StringBuffer*.
To compute a similarity score between the query gloss and any synset definition (referred as the target one), the algorithm needs the following data:

- the length of the target gloss
- the occurrence of a specific term within the target definition

Since those information are not existing in a such reverse index data record, in order to get them, we have to access the database table WN_SYNSET for each analyzed target gloss and since this operation is made by a select query, we expect only that particular synset is imported in the main memory, that is, it is a no sense to think about some exploitation of the *program's spatial locality*.
Moreover, we have also to compute those data since neither the corresponding WnSynset object contains them. The whole process can be represented as in figure 4.17.

The frequency f_i, associated to each query definition's term t_i, indicates how many times we have to access the database to retrieve the above data.

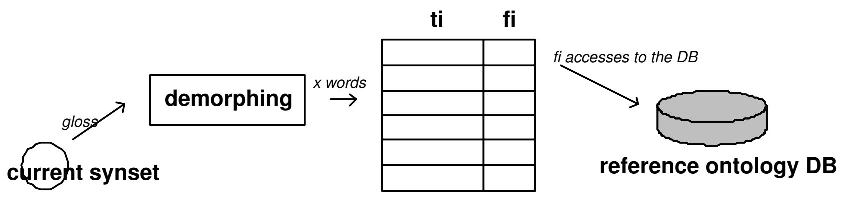


Figure 4.17: Database access in a similarity search task

According to these last considerations, it can be said that any task efficiency depends on two parameters:

- the length of the query definition, referred as L_q
- each query definition term frequency, referred as f_i

To understand which were significant data to test the efficiency of the similarity search we had to refer to the statistics computed in section 4.1.5 about the ontology definitions' lengths.

In particular, from table 4.7, we can get the minimum, the maximum and the mean value of all glosses' length; in this way we assume specific values for variable L_q.

Then, considering those particular values for the definition length, we searched in the ontology for glosses with different term frequencies.

Since the terms' frequency has a rather high variance with the respect to the possible values range, we cannot assume the mean value in table 4.8 as a good distribution index.

On the other hand, to give an idea of how much frequent the terms of a gloss are with the respect of other specific glosses, a sort of relative index, we refer to the *mean terms frequency* in a definition, defined as:

$$f_m = \frac{\sum_{i=1}^{L_q} f_i}{L_q}$$

Table 4.16 shows the waiting time values related to part of the analyzed cases, due to the lack of space we don't report all of them but only some among the most significant ones.

What can immediately be seen is that the waiting time is strictly influenced from both the variables L_q and f_m.

Both the glosses 96150 and 77500 led to a null waiting time: actually the first has a rather high f_m value but it has the minimum length while the second one is both short and with low f_m value.

synset id	var L_q	var f_m	time (sec)	stopwords
96150	1	74	0	
3	6	371.66	55	
5407	48	51.75	210	
77898	5	52.6	4	all
45682	11	82.09	23	all
64709	12	235	107	
77500	3	5.33	0	all
382	23	245.6	382	

Table 4.16: First Solution: waiting time values

Considering the three highest f_m values, i.e. 371.66, 235, 245.6, it should be noted that the more is L_q and the more is the waiting time. In particular the more variable L_q exceeds the mean value and the more is the time one.
Concerning the synset 3 and 45682 the waiting time to compute the first is about two times than the one used by the second synset: this can be explained considering that even if the 45682 L_q value is rather high, its f_m value is much more less than the 3 one.

In the couple 5407, 64709, the waiting time to compute the first gloss is about two times than the one used by the second synset, but in this case the gloss length is the most significant factor given that the 64709 f_m value is more greater than the 5407 one.
However, the majority of the waiting times reported and of them examined at all resulted unacceptable, considering also that we could improve them.

Second solution

In order to usefully exploit the reverse index, we stated to further enrich this data structure, putting in it, at its loading, also the information needed by the similarity search algorithm.
To reach this target, a new record prototype for the index was implemented (see figure 4.18).
In the new record type, for each synset, not only the identifier is reported, but also the definition length (including the duplicated terms) and the position of the index term within that gloss.
Notice that if the index term compares more than one time in a particular defini-

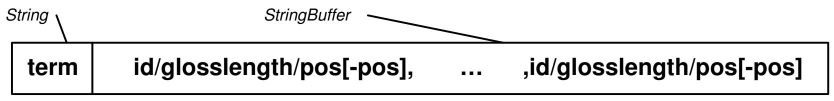


Figure 4.18: Second Solution: reverse index record

tion, we can account this information by counting the number of positions inserted in the reverse index.

synset id	var L_q	var f_m	time (sec)	stopwords
96150	1	74	0	
3	6	371.66	5	
5407	48	51.75	5	
77898	5	52.6	0	all
45682	11	82.09	1	all
64709	12	235	5	
77500	3	5.33	0	all
382	23	245.6	9	

Table 4.17: Second Solution: waiting time values

Table 4.17 shows the same data set used in table 4.16 submitting it to the new similarity search algorithm.
All the non null waiting times are decreased more than 95%.

The method *getSynsetIdAsNumberKey* in *WnReverseIndex.java* (see page 92) is used by the search algorithm in order to retrieve all the needed data from the reverse index; it is called for each term in the query gloss.

In order to optimize this retrieval task, the function works on three HashMap data structures to store the correspondences among any synset and its gloss length (*glossLengthMap* variable), among any synset and the index term (*occMap* variable), and, finally, among any synset and the occurrence of the index term in its definition (*termMap* variable).

Once these hash maps are loaded, the similarity search algorithm exploits them in order to compute the similarity score between any synset definition whose identi-

```
/**
 * @param glossLengthMap Key: synsetId, Value: gloss lenght
 * @param occMap Key: synsetId, Value: term occurrence
 * @param termMap Key: index term, Value: set of synsetId
 */
 public void getSynsetIdAsNumberKey(HashMap glossLengthMap
                                    , HashMap occMap
                                    , HashMap termMap
                                    ) throws TorqueException {
    wnSynsetIdListParser();
    for (Iterator i = _glossLength.keySet().iterator();
     i.hasNext(); ) {
       NumberKey id = (NumberKey)i.next();
       Integer lung = (Integer)_glossLength.get(id);
       if (lung != null) {
          glossLengthMap.put(id, lung);
          occMap.put(id, new Integer(getGlossOccurences(id)));
       }
    }
    termMap.put(getTerm(),_glossLength.keySet());
 }
```

fier appears in them and the query meaning.

Chapter 5

State of the Art

Assuming that any source's wrapper will be able to export not only the structure but also the annotation, i.e. the semantic of a source, then, every time a **MOMIS** system placed somewhere in the Network performs a source acquisition, it has to load the source's structure description but also its annotation.
Loading a source's annotation means that the particular **MOMIS** system should be able to reconcile the part of ontology used by source's creator to annotate the data with its own reference ontology, which could be only WordNet or WordNet and any other extension.
These above are the assumptions on which the next chapter is based. However, before face those problems, we offer, in this chapter, an overview on the actual state of the art regarding the ontologies management, from their alignment to their fully integration. Actually, many problems arise when we try to combine independently developed ontologies or when existing ontologies are adapted for new purposes.

5.1 The concept of ontology

The main part of the related work exploited in this thesis is concerning the concept of "ontology".
We will now give a general overview about this concept, referring to [Tam01]. The term "ontology" has been originally used in philosophy where it indicated the systematic explanation of Existence. More recently, this term has been used in Artificial Intelligence (AI) and Computer Science areas like knowledge engineering, knowledge representation, language engineering, information integration, information retrieval and extraction, agent–based system design and e–commerce. Last but not least, ontologies play a key role in the so called "Semantic Web";

interesting efforts in this area can be seen on web sites
http://www.ontoweb.org
http://www.daml.org
http://www.sewaise.org
http://www.ida.liu.se/ext/etai/seweb/.

Due to the huge number of areas that exploit the term ontology, it is clear that there is a no unique definition for it.
In particular, in AI the term "ontology" denotes an *engineering artifact* that is comprised of a *specific vocabulary* and of a set of explicit assumptions concerning the intended meaning of the words composing the vocabulary.
The AI notion of ontology is then *language dependent*, while in philosophy an ontology is assumed independent from the language used to described it, since it is intended as a particular system of categories accounting for a certain vision of the world [Gua98].

The most widely quoted definition of ontology was given by Tom Gruber in 1993:

> an explicit specification of a conceptualization

This definition sets its basis on the idea that the declarative formalization of the domain knowledge starts from the *conceptualization of the domain*, that is the identification of the objects that are hypothesised to exist in the world and the relationships between them.
According to Genesereth and Nilsson [GN87] a conceptualization is

> a triple consisting of universe of discourse, a functional basis set for the universe of discourse and a relational basis set

The universe of discourse is the set of objects on which the knowledge is expressed.

In [Gru93] and [GN87] an ontology is defined as the *quintuple*:

$$(\mathcal{C}, \mathcal{I}, \mathcal{R}, \mathcal{F}, \mathcal{A})$$

where

- $\mathcal{C}$ is the set of the *concepts*, i.e. the set of abstractions used to described the objects of the world

- $\mathcal{I}$ is the *individuals* set, i.e. the actual objects of the world. The individuals can be referred as the *instances of the concepts*
- $\mathcal{R}$ is the set of *relationships* defined on the set **C**. Each $R \in \mathcal{R}$ is an ordered n–pla $R = (C_1 \times C_2 \times \ldots \times C_n)$. For example `subconcept-of` is the pair (C_p, C_c) where C_p is the parent concept while C_c is the child one.
- $\mathcal{F}$ is the set of *functions* defined on $\mathcal{C}$. Each element $F \in \mathcal{F}$ is a function $F : (C_1 \times C_2 \times \ldots \times C_{n-1} \mapsto C_n)$. For example, the function `Price-of-flat` is a function of the concepts `Year`, `Location`, `Number-of-square-meters` and it returns a concept `Price`, i.e.

  `Year`$\times$`Location`$\times$`Number-of-square-meters`$\mapsto$`Price`
- $\mathcal{A}$ is the set of *axioms*, i.e. first logic order predicates that constraint the meaning of concepts, relationships and functions

Only some of these above components are needed by an ontology. For example, few ontologies include instances.
The simplest type of ontology is composed by the set of concepts $\mathcal{C}$ and the set of relationships $\mathcal{R}$ (lightweight ontology); this composition limits the knowledge that can be expressed about the domain of interest.
Concepts and instances are usually hierarchically organized in *ISA* hierarchies (exploiting the so called *inheritance*): if A is an ancestor of B, i.e. $A \mapsto B$, and $B \mapsto C$ then $A \mapsto C$.

[Gua97] offers the general concept of ontology, giving different definitions of it as follows:

> A (AI-) ontology is a theory of what entities can exist in the mind of a knowledgeable agent
>
> [WS93]

> An ontology for a body of knowledge concerning a particular task or domain describes a taxonomy of concepts for that task or domain that define the semantic interpretation of the knowledge
>
> [Alb93]

> An ontology is an explicit knowledge level specification of a conceptualization, which may be affected by the particular domain and task it is intended for
>
> [vHSW97]

> An ontology is an explicit, partial account of a conceptualization
>
> [GG95]

> An ontology is an explicit, partial specification of a conceptualization that is expressible as a meta level viewpoint on a set of possible domain theories for the purpose of modular design, redesign and reuse of knowledge-intensive system components
>
> [SWJ]

In particular, the author says a very important thing in my opinion: the nowadays distinction between terminological and knowledge–modelling ontologies is misleading, while the meaning of "richer internal structure" of the latter remains very vague.

Then I completely agree with the author when he says that there is no reason to hypothesize a distinction among ontologies on the basis of the "amount and type of structure of their conceptualization"; instead a distinction can be made on the bases of the degree of detail used to characterize a conceptualization.

A very detailed ontology gets closer in specifying the intended conceptualization; on the other hand it pays the price of a richer language.

A very simple ontology may be developed with particulars inferences in mind, to be *shared* among users which *already agree on the underlying conceptualization*.

Simple ontologies like lexicons can be kept "on–line", i.e. they are shareable ontologies (WordNet can be said to belong to this one category), while detailed ontologies, i.e. sofisticated theories on the meanings of the terms used in a lexicon, can be kept "off–line".

Moreover, depending on the subject of their conceptualization, ontologies can be distinguished in *application ontologies, domain ontologies, generic ontologies*, and *representation ontologies*. In a representation ontology, for example, the underlying conceptualization addresses representation primitives, like those defined in Ontolingua's Frame Ontology [Gru93]. A representation ontology can be said to be a meta–level ontology.

However, ontologies can be in general classified according to their different sizes, which range from the level of generality of the concepts they describe, to the type of knowledge they model. Then according to the level of generality used to descibe the domain, it is possible to distinguish the following types of ontologies [Gua98]:

- *Top level ontologies*

  this type of ontologies describes very general concepts or common sense knowledge such as space, time, matter, object, event, action, which are independent from a particular problem or domain

- *Domain ontologies*

  this type of ontologies describes the vocabulary related to a generic domain such as medicine

- *Task ontologies*

  this type of ontologies describes the vocabulary related to a generic task or activity such as diagnosis or selling

- *Application ontologies*

  this type of ontologies describes concepts depending both on a particular domain and on a particular task. They are often a specialization of both domain and task ontologies, corresponding to the role played by the domain entities when they perform certain activities

In [GG95] the authors identify seven most common interpretation of the term *ontology*. Since they are very "clever" and useful in order to understand the concept, we report them:

1. Ontology as a philosophical discipline
2. Ontology as an informal conceptual system
3. Ontology as a formal semantic account
4. Ontology as a specification of a conceptualization
5. Ontology as a representation of a conceptual system via logical theory
   - characterized by specific formal properties
   - characterized only its specific purposes
6. Ontology as a the vocabulary used by a logical theory
7. Ontology as a (meta–level) specification of a logical theory

5.2 Exploiting ontologies

Till now a panoramic on the concept of “ontology” has been shown, but how can we effectively exploit the ontologies ?
In order to make their exploitation useful we generally need to combine them in some way. On the other hand this is not a simple task since every online ontology is an independently developed object.
Combining ontologies, i.e. merging and integrating them, is one of the most important issue in the information area research thus many papers about it are available online.
The general and basic problem those papers face is how to improve and to make more automated the alignment of two different ontologies. This is not precisely our goal when we consider a complex scenario where a designer has to exploit the ontology extension of another one; actually we most have to *reconcile* a reference ontology with any its extension. However, the theory on ontologies alignment and merging processes is the first step from which we should start in order to offer a complete solution to our particular problem.
In the following we try to offer a panoramic view of the most well–known actual systems and methods in the ontologies merging, always considering their possible contribution to our particular case.
[Kle] summarizes the problems that may arise when trying to combine independent ontologies, examining several actual projects that aim at the ontology combination task.
The author underlines the strict correlation between the specification of ontologies on the Web and the “Semantic Web”. Then, in order to develop techniques for the “Semantic Web”, a lot of freely accessible domain specific ontologies would emerge: their *reuse* would be a very important task.
In particular suppose we wants to *reuse different ontologies together*: to reach this target we should then *integrate* them.
An ontologies integration means that the two ontologies are merged creating a new one ontology; however in many cases it could not be necessary since only an ontologies alignment is required.
Notice that both in integration and alignment tasks the two ontologies have to be aligned, that is *they have to be brought into mutual agreement*.
This alignment task is exactly what we need in our work to reconcile the reference–ontology with any its extension.
As [Kle] reports, to integrate two ontologies we have to *iterate* the following steps [DFRW00]:

- find the places in the ontologies where they overlap

- relate concepts that are semantically close via equivalence and subsumption relations (alignment process)
- check the consistency, coherency and non–redundancy of the result

The most critical phase in this process is the concepts alignment since it needs to understand the meanings of the concepts, i.e. it must exploit some semantic similarity functions.

Notice that in our case *we have to exploit WordNet synsets instead of simple concepts, thus we will need of a similarity metric to be applied on the whole synset, including its gloss, its synonyms' set and its relationships.*

Moreover, **aligning two ontologies implies changes to at least one of them**: changes to an ontology will result in a new version of it. Then supposing that the two ontologies are not in the same language, a translation is also required.

All the problems that may arise in combining two ontologies are due to the *mismatches* between them. The way how two ontologies can differ is reported in literature explaining many mismatches types. To have a more clear comprehension of the actual state of the art in ontology combining we should understand the mismatches that may affect this process.
In [Kle], the author first distinguishes between two level of mismatches:

1. *language or meta–model level*
2. *ontology or model level*

At the first level we find all the language primitives used to specify an ontology. This level mismatches hold between different mechanisms to define classes and relations. Four mismatches types can occur at this level: *syntax*, since different ontologies languages uses different syntaxes; *logical representation*, i.e. the difference in representation in logical notions; *semantics of primitives*, despite the fact that sometimes the same name is used for a language construct in two languages, the semantics may differ; *language expressivity*, i.e. the difference in expressivity between two languages: some languages are able to express things not expressible in other languages.

Second level's mismatches are differences in the way the ontology domain is modelled. Three mismatches type can occur at this level: *conceptualization mismatches*, i.e. semantic differences between the two conceptualizations of the domain in the ontologies; *explication mismatches*, i.e. different paradigms used to represent concepts such as time, causality, action ... and different ways in modelling concepts; *terminological mismatches*, i.e. the same concept is represented by different names, the meaning of a term is different in different contexts. Consider also that values in the ontologies can be encoded in different formats.

In our particular case only some mismatches occur. We can assume that no mismatch of the meta–model level can exist since any designer extends only WordNet, respecting all its features and any alignment process is made between WordNet and any its extension.
Instead, since model level mismatches can occur when two or more ontologies describing overlapping domain are combined, we can assume that these ones are possible in our case. In particular *conceptualization mismatches* and *terminological mismatches* are the most possible.

The first type regards mismatches in *scope*, i.e. when two classes seem to represent the same concept but don't have the same instances, and in *model coverage and granularity*, i.e. a mismatch in the part of the domain covered by the ontology or the level of detail to which that domain is modelled.
The second type regards mismatches arising when the same concept is represented by different names in the ontologies (synonym terms, i.e. a term mismatch) and when the meaning of a term is different in different context (homonym terms, i.e. concept mismatch).
Considering that it is unrealistic to hope that merging or alignment at the semantic level could be performed completely in an automated way, in the following are reported current approaches and techniques illustrated in [Kle] to solve language mismatches and to perform an ontology level integration.

5.2.1 Solving language mismatches

Although we have assumed that meta–model level mismatches are not possible in our particular case, we offer an overview of the current approaches on them.

- ***Superimposed Metamodel***

  [BD00] illustrates an approach to transforming information from one representation to another. They focus on information representation schemas that provide structural modeling constructs. Their goal is to represent information for a wide variety of model–based applications in a uniform way and to provide a mapping formalism from one representation to another.

  To reach this target, ontologies languages are each represented in a metamodel, named "superimposed metamodel", that is represented as RDF triples.

Mappings are then specified using *production rules* (if . . . then). Since each RDF triple is a simple predicate, mapping rules are specified using a logic–based language which allow to specify and implement the mappings.

If superimposed information from a source language is mapped to a target language, then it is possible to convert data from the source layer into data conforming to the target layer.

- ***Layered approach to interoperability***

  [MD00] refers to data interoperation using a layered approach as implemented in internet working.

  Their suggestion is to view Web–enabled information models as a series of layers: *syntax layer, object layer, semantic layer*.

  The last is a knowledge representation layer and deals with conceptual modeling. The object layer provides applications with an object–oriented view of their domain, while the syntax layer is responsible for "dumbing down" object–oriented information into document instances and byte streams.

- ***OKBC***

  Open Knowledge Base Connectivity [CFF+98] is a generic interface to knowledge representation systems (KRS). When specifying the API (Application Program Interface) for knowledge representation systems, assumptions are made on the representation used by that KRS. OKBC knowledge model make these assumptions explicit.

  An ontology language, i.e. a specific knowledge representation language, can be bound to OKBC by defining the mapping between the OKBC knowledge model and that language. Then the ontology's users can use the OKBC model instead of the specific language.

  It can be said that the interoperability achieved by using OKBC is at the OKBC knowledge model level.

  OKBC can then solve some mismatches at the language level, but semantic differences beyond the representation cannot be solved by providing mappings to the OKBC knowledge model.

- ***OntoMorph***

  OntoMorph [Cha00] is a system that facilitates ontologies merging and the generation of knowledge–base translators.

  It focuses at transformations to individual ontologies that are needed to align two or more ontologies. The merging's step of "relating concepts semantically close via equivalence and subsumption relations" is split into three:

1. design transformations to bring the sources into mutual agreement
2. editing or morphing the sources to carry out the transformations
3. taking the union of the morphed sources

OntoMorph facilitates the second step by transforming the ontologies into a common format with common names, common syntax, uniform modelling assumptions.

The first step implies the representation meanings' understanding and is therefore a less automatable task.

OntoMorph is then able to solve several problems at the language level; since it requires an executable specification of the transformation, the process can be repeated with modified versions of the original ontologies.

5.2.2 Ontology level integration

- ***SKC***

  Scalable Knowledge Composition project developed an *algebra* for ontology composition used in the **ONION** system is described in [MWK00].

  In this project contexts are defined to be the unit of encapsulation for ontologies while a rule–based algebra is used to compose novel ontological structures. Since the composition of independent and specialized ontologies makes the issue of semantic mismatch explicit, this algebra has also to support rules that resolve mismatches dynamically.

  The algebra operates on ontologies represented by nodes and arcs (term and relationships respectively) in a directed labelled graph.

  Each algebraic operator takes in input a graph of semistructured data transforming it into another graph. This guarantees that the algebra is composable.

  Algebra operations are knowledge driven using articulations rules. These rules can be both logical rules, i.e. semantic implications between terms across ontologies, and functional rules (dealing with conversion between terms across ontologies). The composition rules are partly suggested by expert and lexical knowledge.

  The most critical operation is the *intersection* since it identifies the terms having linkage among the domains (articulation).

  An *articulation ontology* contains the source ontologies' terms that are related and the relations among them: it can be seen as a specification of an alignment.

However, to compute and exploit the articulations requires expertise thus there are the domain experts, providing ontologies and processing rules for their sources and the articulation experts who have to understand what useful concepts appear in these sources' domains and how to combine them in order to support applications.

This separate specification facilitates repeated executions of the composition.

Algebra is then able to solve several conceptual and terminological mismatches. The main advantage in using an algebra to combine different ontologies is the *reusability* since the unified ontology is not a physical entity but a set of terms resulting from the application of the articulation rules. This approach ensures minimal coupling between the sources then sources can be developed and maintained independently.

We didn't choose this algebraic method to perform the reconciliation between WordNet and any its extension even if it could be a valid alternative to our proposal.

- ***Chimaera***

  Chimaera [DFRW00] is an ontology merging and diagnosis tool. The major supported tasks in ontology merging are:

  1. coalesce two semantically identical terms from different ontologies so the resulting ontology refers to them with the same name
  2. identify terms that should be related by subsumption, disjointness, instance relationships, providing support for introduce these relationships

  Chimaera generates name resolution lists that help the user in merging task suggesting terms from the different ontologies that are candidates to be merged or to have a taxonomic relationships not yet included in the merged ontology.

  This target is reached by using a number of heuristics. Then Chimaera can be used to solve mismatches at terminological level and it is also able to find some similar concepts that have different descriptions at the model level.

  Our approach in reconcile WordNet and its extensions exploits some heuristics to identify semantically related concepts and terms that also this tool uses. We could exploit this tool attaching and integrating it with **MOMIS** system and then "passing" to it the reference–ontology and the particular extension which has to be aligned. However we didn't explore this alternative way considering our case more simple than the one of integrating two different ontologies, moreover we don't need of a third integrated ontology.

- ***PROMPT***

  PROMPT (SMART) is an interactive ontology merging tool [NM00].

  It guides the user trough the merging process making suggestions, determining conflict and proposing conflicts solutions.

  For each knowledge–based operation in ontology merging or alignment the changes that PROMPT performs automatically, the new suggestions presented to the user, eventual conflicts are defined. When the user invokes an operation PROMPT crates members of these three sets based on the arguments to the specific invocation.

  The conflicts that may appear in the merged ontology as results of these operations are: names conflicts, dangling references, redundancy and inconsistency in the class hierarchy.

  Summarizing PROMPT gives iterative suggestions for merging and changing concepts.

  We didn't adopt this solution for the same reasons declared about Chimaera tool, being this last one similar to PROMPT.

- ***Common top level model***

  This approach exploits a common top level ontology in order to enable model level interoperability.

  For example the ABC project [BHL99] performs the interoperability of multiple metadata packages that may be associated with and across resources.

  These packages overlap and relate to each other. ABC is based on the assumption that many entities and relationships are so frequently encountered that they don't fall into the domain of any particular metadata vocabulary applying across all of them.

  ABC attempts to:

  1. formally define these common entities and relationships
  2. describe them in a logical model
  3. provide the framework for extending these common semantics to domain and application–specific metadata vocabularies

This approach can solve some interoperability but requires a manual mapping of the ontologies to the common ontology

In our case we don't need to map a particular ontology extension, related to a specific domain of interest, to a top level ontology: we have to reconcile our whole reference–ontology with that extension making it growing.

However the problem to realize a common top level ontology with standard semantics is a very important one thus follows one of the existing approaches to solve it. A comparable approach to that of a "common top level model" is the ***IEEE Standard Upper Level Ontology*** (SUO) [NP01]. This standard will specify the semantics of a general–purpose upper level ontology.
This IEEE project is based on the assumption that *the use of ontologies to specify semantics is a promising technique for software integration* actually creators of different components often assume they understand the terms in the same way but this is not the reality. Even the best documented code has implicit assumptions and ambiguity in the definition of terms.
The main SUO purposes are:

- Automated logical inference will able to exploit this standard to support knowledge based reasoning applications

- The large general–purpose standard ontology will provide the basis for middle–level domain ontologies and lower–level application ontologies

- The resulting ontology will be suitable for more restricted forms such as XML or database schema. In this way database developers will be able to define new data in terms of a common ontology

- Existing systems will be able to map existing data once to the common ontology

- Domain–specific ontologies that are compliant with SUO will be able to interoperate exploiting the common shared terms and definitions

Regarding the overall structure of the standard ontology notice that the ontology could be a single , consistent structure or a lattice of theories that are internally consistent but may be inconsistent with theories that are not part of the same path through the lattice. Following are reported as in [PN01] the main reasons to opt for each solution.

Why a single ontology

- A multiple ontology standard is more difficult to maintain since more than one knowledge model has to be supported
- A multiple ontology standard is more difficult to use
- The comparable effort to create a large ontology, Cyc, doesn't have alternative theories
- No one has yet shown that there are two logically inconsistent and equally valid theories that ought to be included in an engineering–focused ontology

Why a lattice of theories

- There are multiple incompatible and equally valid views on hoe to model the world. A single choice cannot be legislated
- Groups that have a viewpoint will not use an ontology that assumes a different viewpoints

Since the need of manage and compose ontologies to satisfy many practical application is growing, not a single, universal and well-maintained ontology but articulations among ontologies allowing correct interoperation among different domains, each with its own context are required. Actually a global agreement is hard to achieve and is achieved its consistency will be transient.

Another way to perform the ontology integration is by using a language that includes techniques (tags) for combining and integrating different ontologies.
Referring in particular to **SHOE** [HH00] we are in front of a technique that considers also the ontology versioning problem, one of the main feature in Web–based ontologies managing.
Even if there are numerous efforts to create semantic languages for the Web like **Ontobroker** project [DDES98], the Ontology Markup Language **OML** and the Conceptual Knowledge Markup Language **CKML** [Ken99], the Resource Description Framwork **RDF** [LS99], all of these don't sufficiently handle the notions of revising and integrating ontologies.
On the other hand, SHOE faces the problems associated with managing ontologies in a dynamic, distributed and heterogeneous environment such as the Web.

As reported in [HH00], to create a web language with semantics, XML must be extended with knowledge representation languages features (KR). However simply creating an XML syntax for traditional KR languages is insufficient.

The Simple HTML Ontology Extensions language (SHOE) is an *ontology–based knowledge representation language* that is embedded in web pages.

SHOE extends HTML with a set of knowledge oriented tags that provide structure for knowledge acquisition as opposed to information representation. SHOE associates meaning with this content by making each web page commit to one or more underlying ontologies. Interoperability is promoted through the ontologies sharing and reuse. In order to have a compatibility with existing Web–standards SHOE's syntax is defined as an application of SGML.

SHOE exploits a first order logic that *separates data from ontologies* and allows ontologies to provide different perspectives on the data. By mapping SHOE to this logic we can see how different types of revisions to an ontology can affect the way we reason with existing data sources.

In particular revisions that add categories or relations will have non effect, revisions that modify rules may change the answers to queries against the data sources, revisions that remove categories or relations may eliminate certain answers.

This knowledge could be used in weighting costs and benefit of any revision.

Moreover ideally integration is a byproduct of an ontology extension, but in a large and distributed environment the ontologies tend to diverge then the integration will need to be performed periodically.

As shown in [HH00] to incorporate the result of an ontology integration effort three ways are possible:

1. a new *mapping ontology* that extends all the existing ones is created; users of the integrated ontology should explicitly conform to the newly created one. The mapping ontology will contain rules that map concepts between the ontologies

2. *mapping revisions*: each ontology involved in the integration could be revised with the mutual relations to the other ontologies. Naming the ontologies O_1 and O_2 then O_1 is revised by $O_1^{'}$ while O_2 is revised by $O_2^{'}$. $O_1^{'}$ will contain rules that map O_2 objects to the O_1 terminology, while $O_2^{'}$ will do the reverse

3. a new *intersection ontology* is created; it will be extended by the already existing ontologies. Defining $O_1^{'}$ and $O_2^{'}$ as in the previous approach then the intersection ontology will be extended by them and will contain the intersection of concepts. $O_1^{'}$ and $O_2^{'}$ rename terms if necessary.

An application of SHOE could be the next step performed by **MOMIS** system in build more intelligent wrappers. Actual wrappers export only the structure description of a data source not considering in any way the related semantic. A possible and future approach could be as follows: each source creator should annotate its source providing some semantics, then the wrapper presiding that source would be able to export also the annotation. The way these semantics will be exported could be by using XML or any existing language supporting "knowledge tags".

At the end it is necessary to remember that the ontology integration including the alignment of concepts is a very difficult task requiring the understanding of the concepts meanings. Due to this reason, the entire task cannot be completely automated. As we have reported in this state of the art, at the *model level* we can only find tools that *suggest* mappings exploiting heuristics matching algorithms. These heuristics can be divided in two categories:

- linguistic based match: terms with the same word–stem... A deep use of these methods can be found in this works [KL94, Hov98]
- structural and model similarity.

The most interesting paper we have found about a different large–knowledge bases merging is [Hov98], where the author illustrates the process of aligning the principal ontology, SENSUS, and the new ontology MIKROKOSMOS. He also lists all the steps performed in the semi–automated ontology alignment process:

1. a set of heuristics that make initial cross–ontology alignment suggestions
2. a function for integrating their suggestions
3. a set of alignment validation criteria and heuristics
4. a repeated integration cycle
5. an evaluation metric

What we are going to show in the next chapter is a "synset–based reconciliation" method to be applied between WordNet and any its extension and that exploits a mix of heuristics providing a structural graph–based match.

Chapter 6

Reconciliation of different WordNet's extensions

Wandering from a local perspective of an integration system exploited in the previous chapters, the reader is here requested to think about a wider and distributed environment such as the Web. Starting from this more complex scenario many new and different problems to face in about data integration and ontology extensions' exploitation arise.

6.1 What is the meaning of extensions reconciliation?

To have a clearer view about the need of different extensions alignment please refer to the scenario summarized in figure 6.1.

In this example three different Internet nodes represent the systems considered. Going into details, each of these mentioned systems exploits **MOMIS** to perform data's integration. Imagine that a user working on SiDesigner tool in Stanford point wants to integrate two data sources coming from the other considered points, i.e. Bologna and Zurich. Most probably these over mentioned sources are about the same domain of interest, on the contrary case it would be a no sense to think about their integration.

It should be noted that each data source has been annotated within its system, i.e. both Bologna and Zurich's users have already perform that mapping process which allows a correspondence between source's data and the whole "knowledge" (lexicons, meanings, relations) put at disposal by the reference ontology (WordNet and all its extensions).

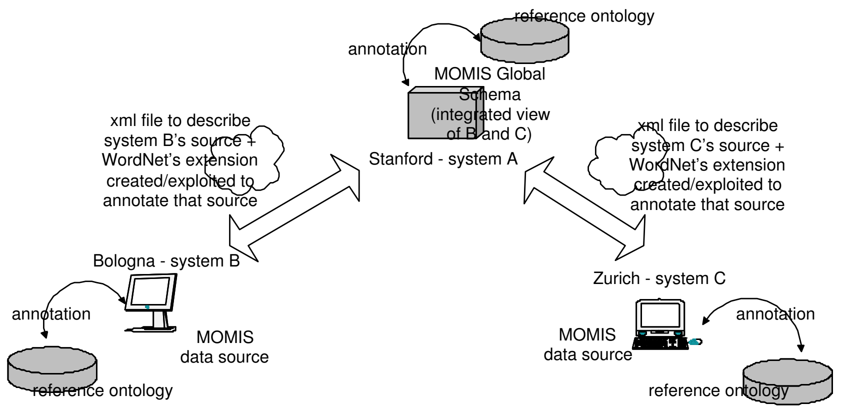

reference ontology
annotation
MOMIS Global Schema (integrated view of B and C)
Stanford - system A
xml file to describe system B's source + WordNet's extension created/exploited to annotate that source
xml file to describe system C's source + WordNet's extension created/exploited to annotate that source
Bologna - system B
annotation
MOMIS data source
reference ontology
Zurich - system C
MOMIS data source
annotation
reference ontology

Figure 6.1: Scenario for any WordNet's extensions reconciliation task

Moreover we know that to further improve the mapping operation each user can exploit the possibility born with this thesis to extend WordNet.

Having clear in mind the whole situation, then it is possible to understand why system A requests both the source data's structures and the WordNet's extensions used to annotate those data sources, to perform the integration process.

It is worth noting that once system A receives all these necessary data, it would contain not only its own extended ontology, i.e. WordNet plus any its extensions, but also the two other WordNet's extensions coming from Bologna and Zurich.

Concerning to this last consideration it may be said that **system A temporary loads extern WordNet's extensions into its own reference ontology**. In this way Stanford's user is able to exploit not only its own set of added concepts and relations but also the ones built by other persons in the world.

Clearly to improve the integration process by using extern WordNet's extensions *it is necessary to align those foreign data with preexistent ones*.

6.1.1 Problems to face in

To really be able to exploit different owner WordNet's extensions in a distributed environment such as the Web, it needs to face in three problems:

1. how to represent the information related to the ontology extensions
2. how to export the data including sources structures descriptions and all the ontology's extensions used to annotate those sources

3. how to perform the alignment and reconciliation among the particular reference ontology and any WordNet's extension

The following sections deepen these three factors offering solutions to them.

6.2 Data representation and export

6.2.1 Source export

MOMIS system makes use of the ODL$_{I^3}$ language to describe the structure of each data source returned by the correspondent wrapper. This particular way to defined a structure includes the definition of different elements like class and attributes. All these elements' names are mapped onto the **MOMIS** reference ontology, i.e. WordNet, during the source's annotation phase described in section 2.1.1.

Referring to the scenario shown in figure 6.1 suppose we are connected at Bologna access point and we are requested to send our annotated source *Tax_Position* to the Stanford Web point.

Considering the nowadays persistent need to maximize the portability of an integration system, it would seem to be more appropriate the exploitation of a very well-known and portable language such as the XML to export the ODL$_{I^3}$ data.

MOMIS is already able to automatically perform any import/export of data both in ODL$_{I^3}$ format and in XML one. Referring to figure 6.2, notice that *Tax_Position* can be said to have an only one class structure, correspondent to the tag named *Interface*, made of four different attributes identified by the tags named *Attribute*. As specified in section 2.1.1, all these class and attributes' names are to be annotated onto the reference ontology.

As we can see in figure 6.2, the user has annotated the source *Tax_Position* choosing the wordforms *university_student*, *name*, *faculty*, *tax_free*, *student_code* for class and attributes names respectively. It should be noted that all the information related to the mapping phase are placed in the tags named ***AttributeAdditionalInfo*** and ***InterfaceAdditionalInfo***.

The data required to get the exact position of an element within the reference ontology are those described in the tags *SlimNode* and *SlimNodeSenso* that define the word baseform and its selected sense number respectively.

These two data are necessary and sufficient to identify an ontology's specific synset due to the unique index built on the couple of lemma identifier and the particular sense number in table WN_LEMMA_SYNSET (see section 4.1.3 in chapter 4).

```
<Source
  name="Tax_Position"
  type="file"
  wrapperName="a"
  hostName="a"
  portNumer="1"
  >
  <Interface
     name="University_Student"
     persistent="false"
     >
     <extent name="University_Student" />
     <Key name="PrimaryKey" >
       <KeyAttribute name="student_code" />
     </Key>
     <Attribute
        name="name"
        type="string"
        >
       <AttributeAdditionalInfo>
           <SlimNode formaBase="name" >
             <SlimNodeSenso numero="1" />
           </SlimNode>
       </AttributeAdditionalInfo>
     </Attribute>
     <Attribute
        name="faculty_name"
        type="string"
        >
       <AttributeAdditionalInfo>
           <SlimNode formaBase="faculty" >
             <SlimNodeSenso numero="2" />
           </SlimNode>
       </AttributeAdditionalInfo>
     </Attribute>
     <Attribute
        name="tax_fee"
        type="long"
        >
       <AttributeAdditionalInfo>
           <SlimNode formaBase="tax_fee" >
             <SlimNodeSenso numero="-2" />
           </SlimNode>
       </AttributeAdditionalInfo>
     </Attribute>
     <Attribute
        name="student_code"
        type="long"
        >
       <AttributeAdditionalInfo>
           <SlimNode formaBase="student_code" >
             <SlimNodeSenso numero="-2" />
           </SlimNode>
       </AttributeAdditionalInfo>
     </Attribute>
     <InterfaceAdditionalInfo>
       <SlimNode formaBase="university_student" >
         <SlimNodeSenso numero="1" />
       </SlimNode>
     </InterfaceAdditionalInfo>
  </Interface>
</Source>
```

Figure 6.2: Example of an annotated schema source

6.2.2 Ontology extension export

Till now we have show the XML representation of an annotated source, however it should be possible to export also any WordNet's extension as an XML file together with the one describing the source structure.

Starting from the point that *it is a no sense to export also part of the original WordNet ontology*, we had to study a way to represent any ontology's extension as an XML file.

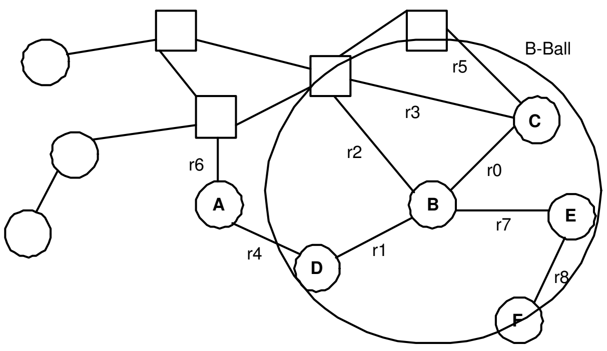


Figure 6.3: Ontology network's subpart

Figure 6.3 shows a subpart of the reference ontology: every *square point* represents an original WordNet's synset while every *round point* is a synset that extends the ontology since built by an integration designer.

According to tags *SlimNode* and *SlimNodeSenso* mentioned in the section 6.2.1, we are able to identify a specific synset within the underlying ontology for every annotated source element. Since from the annotation phase on the *Tax_Position* source we can find out at the most 5 different ontology synsets and due to the will to show a very simple example, we suppose we have identified only three different synsets referred in the following as the C, D, E round points of the figure 6.3.

However, *to perform an useful export of an extended ontology's subpart, we have to consider not only these annotated synsets but also the ones linked directly with any relation to some of them.*

To reach this target we use an algorithm that for each annotated synset explores the whole ontology network subpart centered in it with a radius equal to one–relation–link; in the following we refer to this area as the *ball* for that *head synset*.

According to the choice of exploring only the ball for every head synset, it should be noted that it is sufficient since we are interested in synsets related directly to the head ones. Actually the head synsets are the most useful information to be exported while their one–link–related synsets allow to have an overview of their position within the ontology adding useful information about their own context.

On the other hand, the designer who annotates is able to choose among a wide variety of meanings so when he focuses on a specific one, the whole remaining ontology's subpart is not a useful information referring to a particular data domain.

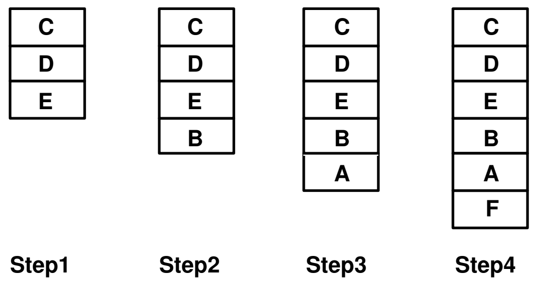


Figure 6.4: Algorithm steps to retrieve the whole subnetwork

To have a more clear view of the way the algorithm works see figure 6.4: at the first step the vector, called V, contains all the head synsets founded out by the annotation phase, i.e. C, D, E (notice that in this particular case no WordNet original synset was annotated). The algorithm is now ready to start the exploration of the balls centered in C, D, E respectively.

Since we have three starting points three steps follow:

- Step 2: Examining the C-ball the algorithm finds out the synset named B: V doesn't contain B yet so B is added to V
- Step 3: Examining the D-ball the algorithm finds out two synset named A and B: only synset A is added to V since B already exists
- Step 4: Examining the E-ball the algorithm finds out two synset named B and F: only synset F is added to V since B already exists

The resulting vector V is exactly the set of synset we want to export together with the annotated data source. In particular the XML file containing the extension's data will be like the one shown in figure 6.5.

Any couple of tags

```
<WnExtension >
    <WnSynset  gloss="gloss" id="synC" >
        <WnLemma name="lemmaC1" senseNumber="'senseNumberC1" />
        <WnLemma name="lemmaC2" senseNumber="senseNumberC2" />
        <WnLemma name="lemmaC3" senseNumber="senseNumberC3" />
    </WnSynset>

    ...

    <WnSynset gloss="gloss" id="synB" >
        <WnLemma name="lemmaB1" senseNumber="senseNumberB1" />
        <WnLemma name="lemmaB2" senseNumber="senseNumberB2" />
        <WnLemma name="lemmaB3" senseNumber="senseNumberB3" />
    </WnSynset >
    <WnRelation type="semantic" symbol="@"
        sourcesynset="synC"
        targetSynset="synB"
    </WnRelation >

    ...

    <WnRelation type="lexical" symbol="="
        sourceLemma="lemmaC2" sourceSenseNumber="senseNumberC2"
        targetLemma="lemmaB1" targetSenseNumber="senseNumberB1"
    </WnRelation >
</WnExtension >
```

Figure 6.5: Example of an XML ontology extension's representation

`<WnExtension > </WnExtension >`

identifies a specific area of the reference ontology: all the synset and any relation among them are then indicated with the tags

`<WnSynset > </WnSynset >`

`<WnRelation > </WnRelation >`

respectively, while each lemma of a synset is referred with tags

`<WnLemma > </WnLemma >`

Every synset is uniquely identified by the local synset identifier and by any couple *(lemma, senseNumber)*.
In order to show a simple example, figure 6.5 reports only two synset, B and C, considering there is also a relation among them (relation r0 in figure 6.3), however

all the synset of figure 6.4 will be in this file.

The file structure is as follows: after listing all the synset belonging to the particular ontology's subpart, also the existing relations among them are to be reported.
The field *type* in the *WnRelation* tag can assume the value *semantic* or the value *lexical* according to the specific relation category, while the field *symbol* is the internal representation of a relational pointer, as described in chapter 1.
In our example we have supposed that the r0 relation that holds between synset C and synset B is an hypernym one. Moreover we assume that a lexical attribute relation holds between the lemma "lemmaC2" and "lemmaB1" of synset C and B respectively. In order to identify a synset, any lexical relation exploits the unique couple *(lemma, senseNumber)*.

6.3 Reconciliation between the reference ontology and any extension

In this section the whole extensions alignment task is discussed.
Referring to the scenario shown in figure 6.1, suppose that the user at Stanford requires first an annotated source from system C, whose XML representation is shown in figure 6.6, and then another source from system B, whose structure is described in figure 6.5.
As explained in the previous section, Stanford's user receives an XML files from both the systems B and C containing these information:

- the annotated source (or a completely annotated and integrated schema). This information allows to identify a specific ontology's subnetwork composed by a set of synsets and relations among them. In the following this information will be referred as *source.xml*

- the XML representation of the particular ontology's area exploited in annotating the source at point1. In the following this information will be referred as *ext.xml*

Notice that these information are built coherently each other and with the respect of the local reference ontology mapping.
However, when spreading our view, it becomes clear that it is a no sense to think about a synset merely as a couple *(lemma, senseNumber)* since *senseNumber* is ordered according to a local representation.

```
<WnExtension >
    <WnSynset  gloss="gloss" id="synH" >
        <WnLemma name="lemmaH1" senseNumber="'senseNumberH1" />
        <WnLemma name="lemmaH2" senseNumber="senseNumberH2" />
        <WnLemma name="lemmaH3" senseNumber="senseNumberH3" />
    </WnSynset>

    ...

    <WnSynset gloss="gloss" id="synK" >
        <WnLemma name="lemmaK1" senseNumber="senseNumberK1" />
        <WnLemma name="lemmaK2" senseNumber="senseNumberK2" />
    </WnSynset >
    <WnRelation type="semantic" symbol="~"
        sourcesynset="synH"
        targetSynset="synK"
    </WnRelation >

    ...

</WnExtension >
```

Figure 6.6: XML ontology extension's representation for system C

So *we should refer to the complete synset objects, comprehensive of their gloss, their synonyms and the relations involving them.*

System A is now ready to load both the ontology's extensions within its own reference ontology. Going into details, every ***extension loading task*** can be viewed as a two steps process:

1. ***Screening***

   of the synset set reported in ext.xml. Every already existing synset within the reference ontology of the receiver system (Stanford in this case) is separated from the ones not existing yet. All the non–existing synset are *automatically loaded* in the receiver's reference ontology with all the relations among them and the WordNet original network. To reach this target three algorithms will run:

   ```
   WnSynset getSynset(WnSynset extSynset);
   ```

   which tests if the synset object *extSynset* in the ext.xml file is already existing within the reference ontology.

   ```
   int addLemma(WnLemma extLemma, WnSynset extSynset);
   ```

supposing that the synset object *extSynset* doesn't exist yet in the reference ontology, this method, after the synset is created, adds to it all its WnLemma objects, according to the ext.xml file.

`void createRel(Vector synsetVec, WnSynset extSynset);`

for all the synset that are in relation with the object *extSynset* according to the ext.xml file, the corresponding relations are built.

If the first method finds out in the reference ontology a "twin synset" for the *extSynset* object, this means that *extSynset* is already existing in the ontology. However, a human expert should confirm if the returned synset object is really "equal" to that in the extension; on the contrary case, also this *extSynset* synset will be treated like the non–existing ones.

The second method returns the new sense number within the receiver reference ontology for each added WnLemma, thus creating a local representation of the imported synset object.

The third method appends the non–existing ontology's extension part to WordNet. Notice that this task is performed in a unique step, since first of all, the non–existing synsets are created calling the methods *getSynset* and *addLemma* and then all those synset are related each other and with WordNet.

It is very important to note that by introducing new synset within the reference ontology in the above way, none of them can be directly related at this step to non original WordNet receiver's synset. Eventual relations with those last synset may be added, in the following, by aligning with the ontology all those extension synset assumed similar to existing ones.

2. ***Reconciliation***

   among the extension synset assumed similar to existing ones and those in the reference ontology.

Referring to our example the two extension's loading tasks that Stanford designer has to perform are the following:

- *About System C*

  Screening for the synset identified by *synH* and *synK* in the system C's ext.xml file. Supposing that none of them already exists within the system A reference ontology, Stanford designer performs an automated loading for the Zurich ontology extension.

- *About System B*

  Screening of the synset set reported in the Step 4 vector in figure 6.4. Now, supposing that the just loaded synset referred as *synK* is assumed the twin synset of that identified by *synC* in the system B ext.xml file, system A automatically loads only A, B, D, E, F synset with the relations among them.

  The conflict between the existing synset *synK* and *synC* must be resolved trough a reconciliation process.

6.3.1 Similarity criteria to test the existence of a query synset

Till now we have only discussed on the main phases concerning to the extension's loading task. But what is very important and fundamental is how we can assume that a generic synset in any extern extension already exists in the reference ontology: once we have resolved this problem we can worry about the conflicts resolution.
Referring to the example in figure 6.1 the question would be: *how system A can divide the set of synsets, reported in Bologna and Zurich's ext.xml files, into two categories: the existing in the system A reference ontology and the non–existing ones?*
This is a very complex problem and we will try to offer a practical solutions to it. In particular consider we have to test if the synset identified by *synC* already exists in the system A reference ontology; in the following the synset whose existence has to be tested will be referred as the *query synset*.
Notice that all the considerations about the similarity criteria exploited to test the query synset set the basis for different implementation of the getSynset method.

Since we are treating complete objects like WnSynset, first of all we have to preface the alignment heuristics known in literature we had exploited to study this problem:

- ***Text matches***

  1. Matches performed on concepts names (*name match*). Having two concepts names in the same language, the assumption on which this technique is based is that similar enough names should suggest that the developers consider the underlying concepts similar
  2. Matches performed on concepts definitions ((*definition match*), as first used in [KL94]). Having two concepts definitions in the same lan-

guage, the base assumption is that similar-enough natural language definitions should also provide some evidence of concepts similarity

- ***Hierarchy matches***

  these techniques exploit the physical structure of the ontology. For example the use of *ambiguity filters on shared superconcepts*, i.e. when a concept (synset in our case) can be aligned to more than one alternative, only those whose superconcepts are already somewhere aligned with the superconcepts of the query one are considered

The first and most immediate solution we propose, to the existence test problem, takes into consideration only *exact matching techniques*.

Performing the test in this way we should only verify if any synset with the *same* gloss as that of *synC* or the *same* set of synonyms as *synC* already exists. Obviously both the glosses and the set of lemmas are demorphed before the matching algorithm runs. In this case the word *same* means that the exploited match technique will return a positive answer only when two compared lemma or strings are exactly and syntactically equal.

On the other hand, considering that it could be very rare that two synset belonging to two different reference ontologies are defined in the same manner or have a different sense definition but the same set of synonyms or both the previous cases, this exact matching would be useful in very few situations so it was discarded.

Due to the last considerations we have decided to think about an *approximate matching technique* that would have considered not only a *syntactic affinity* between two synsets but also a *semantic and structural* one.

The criteria used to say if a synset already exists in the reference ontology needs to compared the query synset *object* with any candidate one, thus considering:

1. *the two senses' definitions*

2. *the two set of synonyms*

3. *the physical positions within the ontologies network*

Known these three information, we may be able to decide if the query synset belonging to the extern extension is already existing within the reference ontology.

Now we formally define the parameters of the problem. Given a query synset $Qsyn$ with all its components derived from the correspondent ext.xml file:

- the sense's definition, named $Qgloss$ and its length L_{Qgloss}

- the set of n synonyms: $Qlemma_1, \ldots, Qlemma_n$
- the set of m relations involving it as the source synset: $Qrel_1, \ldots, Qrel_m$

we can theoretically compare it to any other synset of the reference ontology. We refer to this last one as $Tsyn$ (target synset). Even the generic target synset is composed of a gloss, named $Tgloss$ with a length L_{Tgloss} , a set of p synonyms, $Tlemma_i$ and a set of q relations, $Trel_j$.

The first operation needed to test the existence of $Qsyn$ in the reference ontology is a two steps one:

1. $Qgloss$ is submitted to a demorphing algorithm that removes all the English stopwords and then stems the remaining words. In the particular case that $Qgloss$ is composed only by stopwords, all its original terms are kept. This step produces a terms' vector indicated in the following as $termVec$
2. The vector $synVec$ contains all $Qsyn$ synonyms. Since we are treating extensions of WordNet all the composed lemmas are surely built by replacing the space character with the underscore one, then we have not to separate each composed lemma $Qlemma_i$ into individual words.

The output of the first step, $termVec$, allows us to get all the synset objects in whose definitions at least one of $termVec$ term appears. Notice that we get all these synset by performing an *exact match* on the gloss' terms.

To reach this target we exploit the reverse index data structure placed in the receiver system's reference ontology.
The result of this operation is a well defined set of synset elements; by doing so we have not to start our search considering the whole ontology. This last consideration is a very important one due to the heaviness of the alignment process between the reference ontology and any extension: in this way, we don't need to compute the similarity functions between all the possible couple of synset.
This implementation choice doesn't seem to limit the reconciliation task if we consider that, most probably, *two definitions of the same concept may share at least one significant word.* The consequence of this assumption is that a very small part or zero of relevant synset will not be compared to the query one.

The output of the second step, $synVec$, allows us to extract from the ontology a specific set of lemma objects. To reach this target, every $synVec$ element could be submitted to a *syntactic similarity function* that should compare it with all the

existing lemma in the reference ontology; however, this operation, seems to be a too much onerous and time consuming one.
Thus, the adopted solution is to search each of the $synVec$ lemma within the table WN_LEMMA placed in the receiver system's reference ontology; in this way we exploit an *exact match* to find out all the synset that share at least one synonym with the query one.

In order to get all those synset objects we have to select all the rows in table WN_LEMMA_SYNSET containing the founded lemma identifiers.

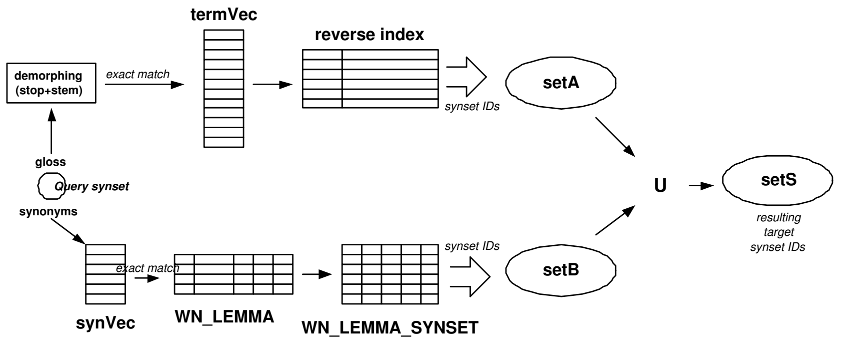


Figure 6.7: Reference ontology synsets we have to explore

Starting from this point we have delimited the part of the reference ontology we need to explore to test the existence of the query synset.

In figure 6.7 this synset retrieval process is summarized; it should be noted that $card\{setA\}$ is in general much greater than $card\{setB\}$. Moreover, to get the whole set of synset to be compared with the query one, we use the union operator obtaining $setS = setA \cup setB$ and not the intersection one since at this step we have to consider *every synset at the same relevance*: clearly a synset belonging to both $setA$ and $setB$ has an higher probability to be more similar than others to the query one.

Assuming that we need to extract only one synset from $setS$, the question is how can we select it ?
We have then to compare the query synset $Qsyn$ with each target synset $Tsyn$ belonging to $setS$ by applying a similarity function on each pair of objects; the target synset associated to the highest similarity score will be considered the *best fit* for the query one (i.e. the twin synset).

It should be noted that in this way *a twin synset for the query one is always found*, except in the rare case that $card\{setS\} = 0$.

To avoid this wrong situation we should impose a ***threshold value*** on the similarity function; in this way also when $card\{setS\} \neq 0$ it will be possible that none twin synset is retrieved.

More precisely

if no synset in the reference ontology exceeds the similarity threshold we can assume the query synset doesn't exist yet; on the contrary case, if one or more synset exceed that value, the one corresponding to the highest similarity score is assumed the best fit for the query synset. In the last case we are able to assume that $Qsyn$ already exists within the reference ontology.

The overall similarity function should be based on a metric including a *semantic contribute* from the similarity between the two meaning definitions and a *syntactic contribute* that considers the similarity between the two sets of synonyms; thus, we may call it also a "combination function".

Possible ways to implement these two contribution functions are shown in the sections 6.3.3 and 6.3.4.

6.3.2 Synset–based reconciliation

In this section the reconciliation process between the reference ontology and a particular extension is discussed. Consider that when we start to perform the reconciliation we have already created all the non existing query synset and all the relations involving them, thus we have modified the reference ontology.

Consider now that the generic ***combination similarity function*** introduced in 6.3.1 could be defined in the following two ways:

$$f_{SIM}(Qsyn, Tsyn) = \alpha f_{SEM}(Qsyn, Tsyn) + \beta f_{SYN}(Qsyn, Tsyn)$$

or

$$f_{SIM}(Qsyn, Tsyn) = \gamma f_{SEM}(Qsyn, Tsyn) * f_{SYN}(Qsyn, Tsyn)$$

$\forall \quad \alpha, \beta \in [0, 1]$
$\forall \quad \gamma \in]0, 1]$

where $f_{SEM}(Qsyn, Tsyn)$ is the semantic contribution while $f_{SYN}(Qsyn, Tsyn)$ is the syntactic one.

Then we will have to set the coefficients and threshold values in order to perform the alignment between any extension and the reference ontology. In general, all these values will be set a priori starting from previous and successful alignment processes. This operation will be possible most of all when the performed reconciliation processes will make available correctly aligned data: if this should be the case then an *automated training* of the combination function coefficients will be possible.

It is necessary to point out that any adopted combination formula has to satisfy the following requirements:

- it must increase with increasing values of the semantic and syntactic similarity functions
- it should mitigate the syntactic match's tendency to grow large quickly. This target can be reached for example by computing its root square value or by dividing it for a constant coefficient. This particular requirement should not be imposed in our case, in which more synonyms match and the more the synset are related
- it must returns a nonzero value if at least one of the composing functions returns a nonzero value
- it must normalize the heuristics' results

After computing the similarity score between the $Qsyn$ and each $Tsyn$ belonging to $setS$, we can exploit a *score–table* associated to the particular $Qsyn$ where at every target synset object is associated the corresponding combination function score value.
By extracting the synset associated to the highest score exceeding the threshold we are able to ***get the best fit for*** $Qsyn$.

Clearly a human supervisor is requested to confirm this suggestion: if this should be the case then $Qsyn$ is assumed already existing within the reference ontology.
To reach this target a new GUI (graphical user interface) application should be developed in order to show to the designer the best target fit extracted from every query synset's $setS$.
Then, *the designer would be requested to confirm the proposed match*: each $Tsyn$ confirmed is assumed the twin synset in the reference ontology for the particular

$Qsyn$, on the contrary case $Qsyn$ will be created from scratch together with all its relations.

To be more precise, what will be shown to the designer is a *list of candidate twin synset*; each of them is the synset associated to the highest similarity score for a particular query synset of the ext.xml file.

Then, through a checkbox selection on each twin synset, the designer may confirm the match between any pair of synset (a query and the corresponding twin one). Every time a twin synset is confirmed "equal" to the related query one, we may imagine that these two objects are now superimposed in the ontology network.

Referring for example to figure 6.8, three query synset Q, Q' and Q'' "were fixed" by the designer on the corresponding twin synset of the reference ontology, i.e. T, T' and T'' respectively.

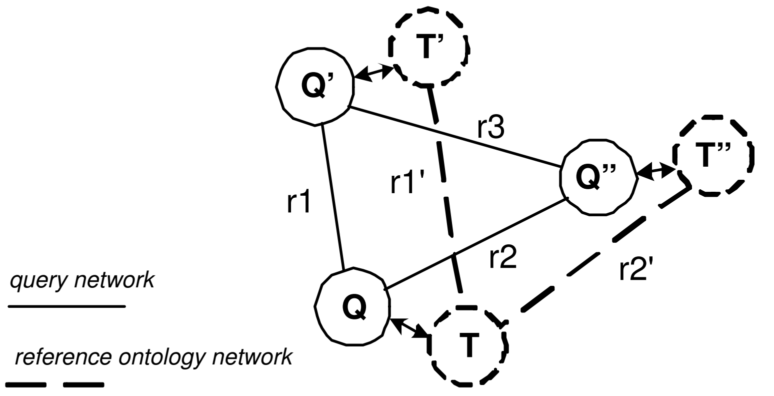


Figure 6.8: Comparison between two ontology subnetworks

After fixing some extension synset on the reference ontology, the designer has to start the conflict resolution algorithm. This last one has to perform a mapping between the relations graph associated to the "query network", i.e. the network that relates each others the query synset, and the underlying relations graph in the reference ontology.

It should be noted that both the "reference ontology network" and the "query network" can be heavily modified by this mapping process through inferencing new or different relations. Clearly any mapping run could be totally automated but the resulting graph should be confirm by the designer owner of the reference ontology.

For example, referring to figure 6.8, if we accept the mappings proposed by the algorithm between r_1 and r_1' and between r_2 and r_2', then we also get a *new* relation, of the type of r_3, among the reference synset T' and T''.

In this way, the reconciliation algorithm allows to extend the knowledge expressed by the reference ontology by inferring new or different relations among concepts.

The last consideration is a very important one since introduce to a wider research in the direction of "exploiting other people's annotation in order to directly enrich our knowledge".

Since the designer is able to assume any query synset existing or not, it is clear that, *according to the particular selected configuration of the "query network"*, the mapping step will lead to very different configurations of the reference ontology. Due to this reason the application GUI should give to the designer the possibility to concurrently choose among different query synset mappings, i.e. among different query networks.
It should be noted that *how many couples and which couples* of synset will be constant features for every query network configuration since the similarity metric used to compare a pair of synset and the number of query synset in the XML extension representation don't change during a reconciliation task.

Once the designer is able to choose among the algorithm different proposals, he has also to select the most appropriated one in order to *reconcile* the extension with his reference ontology.

But how the algorithm extracts a unique mapping for every query network configuration ?
In order to solve any conflicting relation, the reconciliation algorithm has to try different configuration and then to output the *best* one.
The criteria exploited by the algorithm to find out the best graph match configuration are based on these two features:

- *relations type*

- *relations strength*

The first requirement means that the algorithm shouldn't match, for example, a ***meronym*** relation on a ***hyponym*** one.
The second requirement concerns the case in which the algorithm should choose between exclusive and alternative mappings. For example, referring to figure 6.9, we assume that the three query synset of the query network were fixed by the designer on the three concepts in graph1 and graph2.

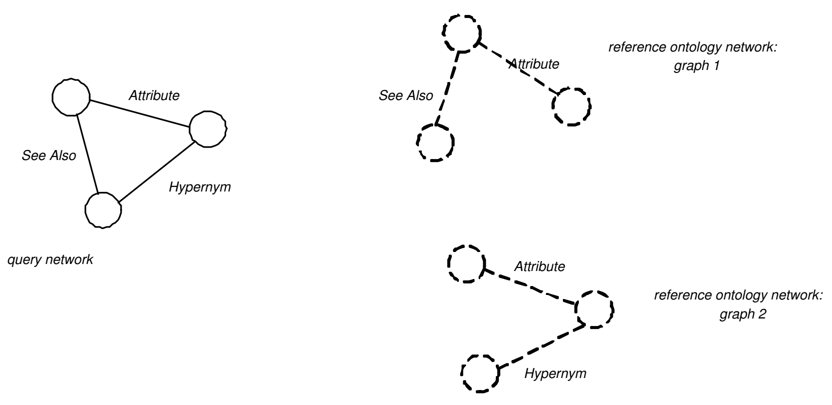


Figure 6.9: Relation strength in a reconciliation process

The reconciliation algorithm has now to retrieve the best match between the two reference ontology graphs. In particular, the best fit should be graph2, since it represents at all more important relations among concepts than graph1. In fact we may assume that an hypernym relation is *more important* than the *See Also* or *Attribute* ones.

Then supposing that is rather a no sense to match different relations each other, we start from the assumption that any relation has to be matched on the same relation type. At this point it is necessary to attribute a numeric weight to a relations match, in order to have an idea of how much the query graph is *similar* to the reference ontology network at all.
The last requirement is necessary since the algorithm may try different configurations within the reference ontology, so it will have to select the best one among them.

As we have attributed a weight to the similarity among concepts, we should now express in numbers the similarity among relations graph.

In order to support the last assumption, we have to define an *importance hierarchy among all the WordNet relations*, based for example also on the frequency of use.
To reach this target, part of the future work related to this thesis may be to associate a fixed numeric weight to each relation in the ontology; every weight will belong to $[0, 1]$; in this way we will exploit a graph matching algorithm which will consider also the *importance among relations*.

The exploitation of relations strength is to be interpreted as a refinement of the standard matching algorithm, based only on a match performed on the relations' type.

In order to extract a unique strength value when comparing two relations graphs, as an index of relations match goodness, the algorithm will have to balance the different relations.

To reach this target analytic functions from a 2D domain (the two base relations strength values) to 1D one (the resulting strength value) will be used.

In particular, if stating to compare also different type of relations, for example a specialization with a generalization one, we should use a "pessimistic" function in order to obtain a low strength value.

On the other hand, when combining the same relations each other or with transitive ones, the algorithm should exploit an "optimistic" function to enforce the resulting value, clearly according to the importance hierarchy among relations.

Then, it can be said referring to figure 6.9 that the resulting graph match among *See Also* and *Attribute* (the graph 1) relations is lower than one among *Attribute* and *Hypernym*, (graph 2) since the last relation is assumed stronger than a *See Also* one.

It should also be noted that from the reported example we can add a new relation on the reference ontology: a hypernym relation is added if considering the graph1 match, while a See Also relation is added when considering the graph2 match.

Remembering that the reconiliation algorithm has two parts:

- the match algorithm that recognizes the corresponding synsets and relation (this is the subject of this section).
- the merging phase that fills the refernce ontology with information from the part of the ontology to be reconciled (new synsets and new relations).

in this case the query network is the ontology to be reconciled and the relations not existing in the selected target graph will be added in the merging phase.

Clearly all the above considerations may be the basis for future research and implementations.

6.3.3 Syntactic Similarity function

The syntactic similarity function should return a numeric score denoting the syntactic similarity between two synsets, i.e. a number representing how much the two set of synonyms are related each others.

In order to exploit an efficient computation of the syntactic similarity degree between two synsets we could rely on a syntactic similarity function that will assume only *binary values* and will be based on *exact matches*.
In particular, if at least one of the $Qsyn$ synonyms appears (i.e. we are using an exact match) in the $Tsyn$ synonyms set, then the returned value will be 1; on the contrary case the the returned value will be 0.
This immediate approach could seem a bit restrictive but we have to consider that the combination function has also a semantic contribution.

However, we may opt for a binary function that returns 1 even when at least one synonym of the query synset is *like* one in the target synset.
Clearly the use of an *approximate match* will bring to a major heaviness in the computation, on the other hand it will allow to get more synsets associated to higher scores than in the first case.
To perform this type of approximate match defined in literature as a *text match*, we focus mainly on two different approaches, even if many string matching techniques exist as illustrated in [Nav01, NBYST01, CN02]:

1. a Levenshtein distance based match

2. a name match as suggested in [Hov98]

In the following paragraphs both these methods are explained.

Edit distance based match

To have a clear view on the working way used by the Levenshtein distance, please refer to section 3.2.3 in chapter 3.
Consider we have to match two terms $Qlemma_i$ and $Tlemma_i$ belonging to the query synset and the generic target one in the corresponding $setS$ respectively.

Moreover, consider we are not interested in extracting only lemmas that are exactly equal to the query one $Qlemma_i$, but we want to exploit a *string match technique allowing errors*.

Due to the last consideration, we assume a certain percentage of precision, referred in the following as s where $s \in [0, 1]$. If, for example, $s = 0.8$ it can be said we allow an error in the 20% of matching cases.

Now calling the lengths of $Qlemma_i$ and $Tlemma_i$ L_{qi} and L_{ti} respectively and indicating the Levenshtein distance computed value as $e.d.(Qlemma_i, Tlemma_i)$, two ways of implementation are possible:

1. the system computes automatically the integer maximum number of errors allowed (k) as follows:

   $k \quad = \quad round[s * \max(L_{qi}, L_{ti})]$

   Then if $e.d.(Qlemma_i, Tlemma_i) \leq k$ the two lemma are similar according to the edit distance criteria.

2. the system computes directly the distance value. If

   $$\frac{e.d.(Qlemma_i, Tlemma_i)}{\max(L_{qi}, L_{ti})} \quad \leq \quad s$$

   then the two lemma are similar according to the edit distance criteria.

The syntactic similarity function $f_{SYN}(Qsyn, Tsyn)$ should return a nonzero value if at least one match exists between the two synonyms sets, while should return zero if any match exists.

We assume that only one match, exact or approximate, is sufficient to relate the two synonyms sets, since referring to the ontology statistics reported in chapter 4 the average number of lemmas for each synset is included in the interval $[1, 2]$.

Clearly this last consideration is a rather arbitrary one, but we think that the considered average value is a good statistic index to decide if two set of synonyms are "similar". In particular the integration system should compute some specific statistics on its reference ontology with a certain frequency, obtaining useful values for computation tasks.

Name match

The name match technique suggested in [Hov98] compares $Qlemma_i$ and $Tlemma_i$ considering decreasing substrings of $Qlemma_i$, chopping off the left. Then, the resulting similarity score between the two lemmas is computed taking into consideration:

- the number of letters matched

	limousine	**vine**	**morphine**
cuisine	0	0	0
uisine	0	0	0
isine	0	0	0
sine	4	0	0
ine	3	3	3
ne	2	2	2
e	1	1	1

Table 6.1: Example 1: Name match example considering a non composed query word

- if the words are exactly equal or if end of match coincides

We should then threshold this syntactic score: if it exceeds the imposed limit, the two lemmas are similar according to this criteria.

As already explained in the previous paragraph, the syntactic similarity function $f_{SYN}(Qsyn, Tsyn)$ should return a nonzero value if at least one positive name match exists between the two synonyms sets of $Qsyn$ and $Tsyn$, while should return zero if any match exists.
For example, in [Hov98] the author defines the resulting function by adding to the square number of letters matched 20 points if the words are exactly equal or 10 points if and of matches coincides.

The following examples should help us to have a more clear view of this name match technique.

Example1:
Assuming $Qlemma_i$ = "cuisine" and three target lemmas $Tlemma_i$, "limousine", "vine", "morphine", the following table 6.1 reports the number of matches considering decreasing substrings of $Qlemma_i$.

By computing the name match scores according to the proposed function we obtain:

1. cuisine–limousine $= 4^2 + 10$
2. cuisine–vine $= 3^2 + 10$

3. cuisine–morphine $= 3^2 + 10$

Example2:
This example shows how this technique treats the composed lemmas. Assuming $Qlemma_i$ = "free world" and one composed target lemma $Tlemma_i$ = "percent of world population", the table 6.2 reports the number of matches considering decreasing substrings of $Qlemma_i$ for each its subword.

	percent	**of**	**world**	**population**
free	0	0	0	0
ree	0	0	0	0
ee	0	0	0	0
e	1	0	0	0
world	0	0	5	0
orld	0	0	4	0
rld	0	0	3	0
ld	0	0	2	0
d	0	0	1	0

Table 6.2: Example 2: Name match example considering a composed query word

By computing the name match score according to the proposed function we obtain:

1. free world–percent of world population $= 5^2 + 20 + 1^2$

6.3.4 Semantic Similarity function

The semantic similarity function is the most important index we can exploit to determine if the definitions of two concepts effectively express the "same" concept. This function should implement the match type defined in literature as the *definition match*.
In particular we focus mainly on the following approaches:

1. a vector space model based match
2. a LSI model based match

3. a definition based match as suggested in [Hov98]

What can be said immediately is that the third method doesn't require term weighting techniques as the first and the second one. The following paragraphs illustrate these three approaches.

Vector Space model based match

Supposing both the $Qgloss$ and the target one $Tgloss$ have been already submitted to a demorphing process, including the remove of stop words phase and the stemming one, these two senses' definitions can be represented as two vectors of terms.

In reality any vector contains not only the set of the remaining terms, but also the weights associated to them. Actually, this match type requires an algorithm that is able to assign a numeric weight to all the remaining terms within each meaning's definition.
Such algorithm will run automatically and it will use a well known term weighting technique (to have an idea of the working process of the weight assigning phase refer to the section 6.3.5).

Given two vectors v_q and v_t representing $Qgloss$ and $Tgloss$ respectively, this definition match type determines the *similarity degree* between the two concepts by computing the *correlation* among them, i.e. the cosine of the angle between the two vectors.
More precisely, defining:

$$\begin{aligned} v_q &= (w_{q1}, \ldots, w_{qx}) \\ v_t &= (w_{t1}, \ldots, w_{ty}) \end{aligned}$$

where

- $x = L_{Qgloss}$
- $y = L_{Tgloss}$
- w_{qi} is the weight associated to the term numbered i within the gloss $Qgloss$
- w_{ti} is the weight associated to the term numbered i within the gloss $Tgloss$

the correlation formula adopted in the vector space model to get the measure of the similarity between v_q and v_t is the following:

$$\frac{\sum_{i}^{shared} w_{qi} * w_{ti}}{\sqrt{\sum_{i=1}^{L_{Qgloss}} w_{qi}} * \sqrt{\sum_{i=1}^{L_{Tgloss}} w_{ti}}}$$

where $shared$ is the number of common terms between $Qgloss$ and $Tgloss$.

The numerator is the sum of the weights' products for all the couple of terms shared in both definitions, while the denominator is the vectors' norms product: this last one allows a normalized similarity score.
The semantic similarity function $f_{SEM}(Qsyn, Tsyn)$ values should assume exactly this correlation score that varies continuously in $[0, 1]$.
The use of this method should improve the search for the most similar synset to the query one but it is worth noting that the exploitation of automated term weighting techniques could make the alignment process very heavy.

LSI model based match

The **LSI** technique, where LSI means **L**atent **S**emantic **I**ndexing, is taken into consideration since it seems more accurate than a classical vector model based match.
Actually, summarizing the contents of definitions in sets of index terms may cause that many unrelated definitions will be returned or that relevant definitions that don't contain any of the indexed term will not be retrieved.
LSI method extracts from a definition the concepts it represents. This match considers that the ideas expressed in a definition are more related to the concepts described in it than to the index terms used in its description.
To test if a target definition is relevant to a query one, LSI perform a *matching between the concepts* expressed in both the meanings.

The idea on which this model is based is to map each definition vector into a lower dimensional space where concepts are the coordinates.
If we consider a definition as a vector whose terms correspond to the space coordinates, we can say that LSI projects this vector onto a latent semantic subspace, i.e. with a minor dimensionality, where the main axis are the concepts.

A concept is here defined as a set of terms that frequently occurs within a definition: in this way LSI perform first a clustering of the terms and then a clustering of the definitions. Moreover it should reduce the distance among definitions belonging to the same cluster while should increase the distance among definitions belonging to different clusters.

This method is similar to the one described in the previous paragraph 6.3.4 since it starts from two vectors representing the stemmed definitions where each terms

is associated to a numeric weight.
Each *shared* term will have a nonzero weight in both the definitions, while any other term will have a zero weight in the definition that doesn't contain it.

After having computed all terms weights, the LSI based match builds the so called *weights' matrix* $\mathbf{W}$ $[x \times y]$ where $x = card\{Qgloss \cup Tgloss\}$ and y is the number of definitions examined, i.e. 2; the rank of W is $r = min(x, y)$.
The core algorithm of an LSI method is the computation of the singular value decomposition of W. In the following this operation will be referred as $\mathbf{SVD}$. The SVD technique applied on matrix W decomposes it univocally into a three matrixes product:

$$W = \Gamma \times \Delta \times D^T$$

where

- $\mathbf{\Delta}$ is the diagonal $[r \times r]$ matrix of singular values (eigenvalues–concepts matrix)
- $\mathbf{\Gamma}$ columns are the eigenvectors of $W \times W^T$. It denotes the similarity degree between a term and a concept. Its dimension is $[x \times r]$
- $\mathbf{D}$ columns are the eigenvectors of $W^T \times W$. It denotes the similarity between a definition and a concept. Its dimension is $[y \times r]$

If only the κ largest singular values of Δ are kept along with their corresponding columns in Γ and D^T, the resulting matrix

$$W_\kappa = \Gamma_\kappa \times \Delta_\kappa \times D_\kappa^T$$

is exactly the matrix of rank κ that is closest to the original W in the least square sense; $\kappa < r$ is the dimensionality of a reduced concept space.

We should choose κ in order to balance two effects:

- to fit all the structure in the real data κ should be large
- to filter out all the non relevant representational details κ should be small

SVD method gets the similarity existing between two concepts $Qgloss$ and $Tgloss$ by computing this matrix:

$$W_\kappa^T W_\kappa = (D_\kappa \times \Delta_\kappa) * (D_\kappa \times \Delta_\kappa)^T$$

where the element (i, j) quantifies the relationship between the definition i and the definition j.

Definition match

This matching technique is exactly the same discussed in the chapter 4, so for any detail about it please refer to the section 4.2.

6.3.5 Term weighting: the theory

As already explained two definitions $Qgloss$ and $Tgloss$ can be seen a two vectors of terms. A more formal representation is obtained by including in each vector all possible content terms allowed in the system and adding term weights assignments.
Thus, naming w_{qk} (w_{tk}) the weight of term t_k in vector v_q (v_t) representing $Qgloss$ ($Tgloss$) and supposing t terms in all are available for defining meanings, the two mentioned vectors would be:

$$\begin{aligned} v_q &= (t_0, w_{q0}; t_1, w_{q1}; \ldots; t_t, w_{qt}) \\ v_t &= (t_0, w_{t0}; t_1, w_{t1}; \ldots; t_t, w_{tt}) \end{aligned}$$

We can assume only binary weights and then say that w_{qk} (w_{tk}) is 0 if term t_k doesn't appear in the definition $Qgloss$ ($Tgloss$) and that w_{qk} (w_{tk}) is 1 if term t_k appears in the definition $Qgloss$ ($Tgloss$).
In this way the similarity between $Qgloss$ and $Tgloss$ can be computed as the number of terms jointly assigned to both the definitions, i.e.

$$\sum_{k=1}^{t} w_{qk} * w_{tk}$$

To further improve this similarity computation, *term weights in decreasing term importance order could be assigned*, that resulting in weights allowed to vary continuously between 0 and 1, where 1 is assigned to the most important terms and 0 to the less ones.

To use normalized weights assignments, any weight should depend on the other weights in the same vector: to reach this target a vector length normalization could be exploited. If we rewrite the previous similarity formula considering also a vector length normalization, we obtain

$$\frac{\sum_{k=1}^{t} w_{qk} * w_{tk}}{\sqrt{\sum_{k=1}^{t} (w_{qk})^2} * \sqrt{\sum_{k=1}^{t} (w_{tk})^2}}$$

A vector matching system performing a comparison between $Qgloss$ and many other definition $Tgloss$ vectors provides ranked retrieval output in decreasing order of the computed similarity.

Now the main question to face is how can we compute the terms' weights. The next paragraph offers an overview about this problem.

6.3.6 Specifying the weights

The main goal of a term weighting system is the enhancement of retrieval effectiveness. To evaluate retrieval performances usually recall and precision features are used.

Recall is the ratio of the number of relevant retrieved definitions to the total number of relevant definitions in the collection.

Precision is the ratio of the number of relevant retrieved definitions to the total number of retrieved definitions.

Generally the recall function seems to be better served by using high–frequency terms, i.e. terms occurring in many definition of the collection. The precision factor, on the other hand, appears best served by using highly–specific terms that are able to discriminate the relevant definitions among the non relevant ones. To satisfy the recall and precision requirements we have to observe three considerations about weights' specification:

1. Term frequently mentioned within the definitions appear to be useful to the recall function so a *term frequency* **tf** factor measuring the occurrences of any term in the definitions is defined

2. When the high frequency terms are not concentrated in few definitions but are sparse in the whole ontology–collection then all definitions tend to be retrieved, affecting in this way the precision function. The *inverse definition frequency* **idf** index is introduced to favor terms concentrated in few definitions. idf is typically assumed as $\log \frac{N}{n}$ where N is the total number of definitions while n is the number of definitions in which the term appears.

   Considerations about the term discrimination make clear that the *best terms should have high term frequency but low overall ontology–collection frequency.*

   According to this fact a measure of a term's importance could be **tf**$\times$**idf**

3. This last factor appears useful in systems with widely varying vectors lengths. Actually it could be the case that short definitions tend to be represented by short term vectors, while much larger terms' sets correspond to the longer

definitions. When a large number of terms is used within the definitions, the case of matches between query definition and target definition is most probable and therefore the longer definitions have a better chance of being retrieved than the short ones. But all relevant definitions should be treated as equally important for retrieval purposes: to reach this target a normalization factor can be included within the term weighting formula to equalize the length of the definitions vectors.

In the respect of these explained requirements the most general formula to obtain the weight w_{qi} for the i–term of the definition vector v_q is:

$$w_{qi} = [0.5 + \frac{0.5 * \mathbf{tf}}{\max \mathbf{tf}}] * \log \frac{N}{n}$$

where

the first factor is the normalized term frequency, where $\max \mathbf{tf}$ is the frequency of the most frequent term within the vector v_q and constant 0.5 serves to lie this factor between 0.5 and 1.

This formulation was suggested in [SB87, Sal89] to determine the weights exactly in the case of short query vectors where each term is important; moreover the ontology–collection frequency component $\log N/n$ is the better one since it very similar to the probabilistic term independence factor. For all these reasons we could assume this term weighting formula for our particular case.

Chapter 7

Conclusions and future work

In this document, we have presented an approach to make possible the extension of the WordNet ontology.
The need to extend the reference ontology in an integration information system is due to the actual way of sources annotation, which can be very restrictive when particular and specific concepts are absent within the ontology.
The proposed approach was implemented by extending the **SI-Designer** interface of the **MOMIS** system. The developed tool **WNEditor** is an application GUI (graphical user interface) which exploits an underlying Java library created in order to make possible the WordNet extension.

This work is based on a wide and interdisciplinary scientific literature; in particular, we have exploited the semantics associated to the definitions of the reference ontology concepts in order to make more easy the extension phase.
To reach this target we have studied and implemented information retrieval techniques that allow to extract from the ontology all the similar synset to a fixed one.

By using the possibility born with this thesis to extend WordNet, the generic designer connected to a **MOMIS** system will be able to exploit his own new concepts, but also those introduced or aligned in the reference ontology by other designers.

Future research in **MOMIS** will face the problem of the automatic integration of annotated data sources, where we will study techniques to eliminate (at least in some applications) the role of the integration designer.
To automate the integration we will move to the annotation capabilities of wrappers and we will extend the ODL$_{I^3}$ language and data structures to support anno-

tations. This means that in the future anyone who wants to publish his data source on an integration **MOMIS** system, will have also to annotate those data before publishing them.

The basic assumption of this requirement is that *nobody has more clear in mind the semantic associated to a source than its own creator*.

Moreover, two important advantages could be exploited from a *source side annotation*:

- the Semantic Web agents, which exploit concepts–based searches, will be able to find out also this data source by relying on its annotation
- the particular data source will be annotated from its creator one time only in its life cycle. This implies a total reusability of any source annotation

Once we will get this *source side annotation* in a portable format, we will also be able to fully exploit the reconciliation between that annotation, expressed as a micro–ontology, and any **MOMIS** system reference ontology, i.e. WordNet plus any its extension, by applying the algorithms proposed in this work.

Furthermore, we will consider using Mobile Agents as tool to discover *interesting data sources*.

This thesis has also faced the problem of an *intelligent data retrieval*, by introducing in a particular information integration system, information retrieval techniques in order to enhance a deeper exploitation of the semantics related to concepts. The preliminary study done and all the proposed similarity functions will have to be reviewed, refined and evaluated in order to get a more and more better data retrieval.

The **MOMIS** project was financed for years 2001 and 2002 by the *D2I: Integration, Warehousing, and Mining of Heterogeneous Data Sources* project of the Italian MURST ministry.

Bibliography

[Alb93] M. Alberts. *YMIR: an ontology for engineering design*. PhD thesis, University of Twente, 1993.

[BBC$^+$00] Domenico Beneventano, Sonia Bergamaschi, Silvana Castano, Alberto Corni, R. Guidetti, G. Malvezzi, Michele Melchiori, and Maurizio Vincini. Information integration: The momis project demonstration. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 611–614. Morgan Kaufmann, 2000.

[BCV99] S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record Special Issue on Semantic Interoperability in Global Information*, 28(1):54–59, 1999.

[BCVB00] S. Bergamaschi, S. Castano, M. Vincini, and D. Beneventano. Semantic integration of heterogeneous information sources. *Data & Knowledge Engineering*, 2000.

[BD00] S. Bowers and L. Delcambre. Representing and trasforming model-based information. In *First Workshop of the Semantic Web at the Fourth European Conference on Digital Libraries, Lisbon, Portugal, September 18-20*, 2000.

[BHL99] D. Brickley, J. Hunter, and C. Lagoze. ABC: A logical model for metadata interoperability, October. 1999. Harmony discussion note, see http://www.ilrt.bris.ac.uk/discovery/harmony/docs/abc/.

[BN94] S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Journal of Applied Intelligence*, 4:185–203, 1994.

[BYRN] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley.

[CAV00] S. Castano, V. De Antonellis, and S. De Capitani Di Vimercati. Global viewing of heterogeneous data sources. *IEEE Transactions on Knowledge and Data Engineering*, 2000.

[CDL02] Diego Calvanese, Giuseppe DeGiacomo, and Maurizio Lenzerini. A framework for ontology integration. In I. Cruz, S. Decker, J. Euzenat, and D. McGuinness, editors, *The Emerging Semantic Web Selected Papers from the First Semantic Web Working Symposium*. IOS Press, 2002.

[CFF$^+$98] V.K. Chaundhri, A. Farquhar, R. Fikes, P.D. Karp, and J.P. Rice. OKBC: a programmatic foundation for knowledge base interoperability. In *Proceedings of the Fifteenth American Conference on Artificial Intelligence (AAAI–98), AAAI Press/The MIT Press, Madison, Wisconsin*, pages 600–607. Morgan Kaufmann, 1998.

[Cha00] H. Chalupsky. OntoMorph: A translation system for symbolic knowledge. In B. Selman A.G. Cohn, F. Giunchiglia, editor, *Principles of Knowledge Representation and Reasoning. Proceedings of the seventh international conference (KR2000), San Francisco, CA, 2000*, pages 471–482. Morgan Kaufmann, 2000.

[CHS$^+$94] M.J. Carey, L.M. Haas, P.M. Schwarz, M. Arya, W.F. Cody, R. Fagin, M. Flickner, A.W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J.H. Williams, and E.L. Wimmers. Towards multimedia information system: The garlic approach. Technical report, IBM Almaden Research Center, San Jose, 1994.

[CN02] Edgar Chavez and Gonzalo Navarro. A metric index for approximate string matching. In *Proceedings of LATIN'02. LNCS 2286*, pages 181–195, 2002.

[DDES98] D.Fensel, S. Decker, M. Erdmann, and R. Studer. Ontobroker: How to enable intelligent access to the www. Technical report, AI and Information Integration, Technical Report WS-98-14, 36-42, Menlo Park CA, 1998. AAAI press.

[DF] Ying Ding and Dieter Fensel. Ontology library systems: The key for successful ontology reuse. In *Proceedings of the the 1st Semantic web working symposium, Stanford, CA, USA, July 30th-August 1st, 2001.*

[DFKO02] Ying Ding, Dieter Fensel, Michel Klein, and Borys Omelayenko. The semantic web: Yet another hip? *Data and Knowledge Engineering*, 2002.

[DFRW00] D.L.McGuinness, R.E. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In B. Selman A.G. Cohn, F. Giunchiglia, editor, *Principles of Knowledge Representation and Reasoning. Proceedings of the seventh international conference (KR2000), San Francisco, CA, 2000*, pages 483–493. Morgan Kaufmann, 2000.

[Fou] European Commission US National Science Foudation. Research challenges and perspectives of the semantic web, sophia antipolis, france, 3-5 october, 2001. Workshop report and recommendations. http://www.ercim.org/EU-NSF/.

[GG95] N. Guarino and P. Giaretta. Ontologies and knowledge basis towards a terminological clarification. In N. Mars, editor, *Towards very large knowledge bases: knowledge building and knowledge sharing, IOS Press, Amsterdam*, pages 25–32. Morgan Kaufmann, 1995.

[GKD97] M. R. Genesereth, A. M. Keller, and O. Duschka. Infomaster: An information integration system. In *Int. Conf. ACM SIGMOD-97*, pages 539–542, Tucson, Arizona, 1997.

[GN87] M.R. Genesereth and N.J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.

[Gru93] T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[Gua97] Nicola Guarino. Understanding, building, and using ontologies: A commentary to ”using explicit ontologies in kbs development”, by van heijst, schreiber, and wielinga. *International Journal of Human and Computer Studies*, 46:293–310, 1997.

[Gua98] N. Guarino. Formal ontologies and information systems. In N. Guarino, editor, *Proceedings of FOIS 98, IOS Press, Amsterdam*, 1998.

[HH00] Jeff Heflin and James Hendler. Dynamic ontologies on the web. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000), AAAI/MIT Press, Menlo Park, CA*, pages 443–449, 2000.

[Hov98] E.H. Hovy. Combining and standardizing large-scale, practical ontologies for machine translation and other uses. In *Proceedings of the 1th International Conference on Language Resources and Evaluation (LREC), Granada, Spain, May 28-30*. AAAI Press, 1998.

[Hul97] R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *ACM Symp. on Principles of Database Systems*, pages 51–61, Tucson, Arizona, 1997. ACM.

[JW] Jan Jannink and Gio Wiederhold. Ontology maintenance with an algebraic methodology: a case study. In *Proceedings of 1999 AAAI workshop on Ontology Management, Orlando FL, July 1999.*

[Ken99] R.E. Kent. Conceptual knowledge markup language: The central core. In *Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management*, 1999.

[KFvHH] M. Klein, D. Fensel, F. van Harmelen, and I. Horrocks. The relation between ontologies and XML schemas.

[KL94] Kevin Knight and Steve K. Luk. Building a large-scale knowledge base for machine translation. In *Proceedings of the 12th National Conference on Artificial Intelligence, Menlo Park, CA, USA, July 31-August 4*, pages 773–778. AAAI Press, 1994.

[Kle] Michel Klein. Combining and relating ontologies: an analysis of problems and solutions. In *Workshop on Ontologies and Information Sharing, IJCAI'01, Seattle, USA, August 4-5, 2001.*

[LHL01] Tim Berners Lee, James Hendler, and Ora Lassila. The semantic web. May 17 2001. http://www.scientificamerican.com.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 251–262. Morgan Kaufmann, 1996.

[LS99] O. Lassila and R. Swick. Resource description framework RDF model and syntax. W3C (world-wide web consortium). 1999. At http://www.w3.org/TR/REC-rdf-syntax-19990222.html.

[Mal00] Giovanni Malvezzi. Estrazione di relazioni lessicali con wordnet nel sistema momis. Master's thesis, Università di Modena e Reggio Emilia, Modena, Italy, 2000. Available from http://sparc20.ing.unimo.it, Written in Italian.

[MD00] Sergey Melnick and Stefan Decker. A layered approach to information modeling and interoperability on the web. In *Electronic proceedings of the ECDL 2000 Workshop on the Semantic Web, Lisbon, Portugal, September 21*, 2000.

[Mil95] A.G. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995. Wordnet Web site: http://www.cogsci.princeton.edu/ wn/.

[MWK00] P. Mitra, G. Wiederhold, and M. Kersten. A graph-oriented model for articulation of ontology interdependencies. In *Proceedings of Conference on Extending Database Technology (EDBT 2000), Konstanz, Germany, March*, 2000. Also, Stanford University Technical Note, CSL–TN-99-411, August, 1999.

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[NBYST01] Gonzalo Navarro, Ricardo Baeza-Yates, Errki Sutinen, and Jorma Tarhio. Indexing methods for approximate string mathcing. *IEEE Data Engineering Bulletin, Special issue on "Managing Text Natively and in DBMSs"*, 24(4):19–27, 2001.

[NM00] N. Fridman Noy and M.A. Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000), Austin, TX*. AAAI/MIT Press, 2000.

[NP01] Ian Niles and Adam Pease. Origins of the IEEE standard upper ontology. In *Working Notes of the IJCAI-2001 Workshop on the IEEE Standard Upper Ontology*, 2001.

[Pea01] Adam Pease. Evaluation of intelligent systems: The high performance knowledge bases and IEEE standard upper ontology projects.

In *Proceedings of the 2001 workshop on Measuring Performance and Intelligence Of Intelligent Systems (PERMIS'2001)*, 2001.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Int. Conf. on Data Engineering, ICDE'95*, pages 251–260, Taipei, Taiwan, 1995.

[PN01] Adam Pease and Ian Niles. IEEE standard upper ontology: A progress report. 2001. IEEE SUO Working Group on Standard Upper Ontology, see http://suo.ieee.org/.

[Sal89] Gerard Salton. *Automatic Text Processing*. Addison-Wesley, 1989.

[SB87] Gerard Salton and Chris Buckley. Term weighting approaches in automatic text retrieval. November 1987. Supported in part by NSF under grants IST 83-16166 and IRI 87-02735.

[SWJ] Schreiber, Wielinga, and Jansweijer. The kaktus view on the 'o' word. In *Proceedings of IJCAI95 Workshop on Basic Ontological Issues in Knowledge Sharing. Montreal, Canada, 1995.*

[Tam01] Valentina Tamma. *An Ontology Model supporting Multiple Ontologies for Knowledge sharing*. PhD thesis, University of Liverpool, October 2001.

[vHSW97] G. van Heijst, A.T. Schreiber, and B.J. Wielinga. Using explicit ontologies in kbs development. *International Journal of Human–Computer Studies*, 45:184–292, 1997.

[Wie94] Gio Wiederhold. An algebra for ontology composition. In *Proceedings of 1994 Monterey Workshop on Formal Methods, U.S. Naval Postgraduate School, Monterey, CA, September*, pages 56–61, 1994.

[WJ98] Gio Wiederhold and Jan Jannink. Composing diverse ontologies. Technical report, Stanford University Technical Report, August, 1998.

[WS93] B. J. Wielinga and A. Th. Schreiber. Reusable and sharable knowledge bases: a european perspective. In *Proceedings of First International Conference on Building and Sharing of Very Large-Scaled Knowledge Bases. Tokyo, Japan Information Processing Development Center*, 1993.

Index