

ROCS

The role of ontology in coordination systems

Sascia Virruso

`sascia.virruso@studio.unibo.it`

Project developed in the course of
Multi Agent Systems

April 1, 2010

Abstract

The *ROCS* project aims to explore the complex world of ontologies starting from the definition to the description of its relevance in the world of Multi Agent Systems (MAS). For long time every work focuses on the syntax of language more than the semantics that it has. Now, with the introduction of new programming paradigms (like the agents), the growing of networks, the heterogeneity of data and informations, it is necessary to move toward a system where semantics has a crucial role. In the following report it's going to be described how an ontology can be represented (RDF, OWL, Wordnet) and various methods useful to compare them (W4, Hmatch, SOSTuS).

Contents

1	What is an ontology	2
1.1	Definition	2
1.2	Ontologies in MAS	3
2	Ontology representations	4
2.1	RDF, RDFS & OWL	5
2.1.1	RDF	5
2.1.2	RDFS	6
2.1.3	OWL	7
2.2	WordNet	9
2.3	W4 model	9
2.3.1	The model	9
2.3.2	Data representation	10
3	Comparing ontologies	11
3.1	Hmatch algorithm	11
3.1.1	Ontology representation	11
3.1.2	Linguistic features	12
3.1.3	Contextual features	12
3.1.4	Matching process	12
3.1.5	H-match algorithm	12
3.2	W4 comparison	13
4	SOSTuS project	15
4.1	Semantic Tuple Matching	15
4.2	Adoption of WordNet	16
4.3	Organization Based on Soft Matching	16
4.4	Semantic Relationships	18
4.5	Semantic and Lexical Matching	20

Chapter 1

What is an ontology

1.1 Definition

In philosophy *ontology* is a philosophical discipline—a branch of philosophy that deals with the nature and the organisation of reality defined for the first time in the Science of Being (Aristotle, *Metaphysics*, IV, 1).

In computer science and information science, an ontology is a formal representation of the knowledge by a set of concepts within a domain and the relationships between those concepts.

A definition of the word ontology in computer science was created by Thomas Gruber in [3] saying that “An ontology is a specification of a conceptualization.” Behind this words there is the concept that an ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. This definition is consistent with the usage of ontology as set-of-concept-definitions, but more general. And it is certainly a different sense of the word than its use in philosophy. An ontology can be seen as a set of definitions of formal vocabulary, related in some way. Ontologies are often equated with taxonomic hierarchies of classes, but ontologies need not to be limited to these forms. Ontologies are also not limited to conservative definitions, that is, definitions in the traditional logic sense that only introduce terminology and do not add any knowledge about the world (Enderton, 1972) . To specify a conceptualization one needs to state axioms that do constrain the possible interpretations for the defined terms.

Ontology is an engineering artifact constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary. Ontology describes a formal specification of a certain domain in terms of

- shared understanding terms of a domain of interest
- formal and machine manipulable model of a domain of interest

In these terms ontology describes both the syntax of terms and relationships between concepts. The structure of an ontology typically has two distinct components:

- Names for concepts in the domain

- Background knowledge/constraints on the domain

In that way new terms can be formed by combining existing ones and meaning (*semantics*) of such terms is formally specified.

1.2 Ontologies in MAS

The concept of ontology is important both in centralized and in distributed systems where the heterogeneity of data and the continuous flow of knowledge let people need to categorize it and make reasoning about that. In order to solve these kind of problems, people often use MAS implementing KBS.

Knowledge based systems are artificial intelligent tools working in a narrow domain to provide intelligent decisions with justification. Knowledge is acquired and represented using various knowledge representation techniques rules, frames and scripts. The basic advantages offered by such system are documentation of knowledge, intelligent decision support, self learning, reasoning and explanation[9].

In order to obtain a consistent and uniform system, people often use a common ontology after a preprocess step of mapping various concepts (deriving from different applications or systems, in different format, etc.) on a common ontology properly defined.

Chapter 2

Ontology representations

An ontology can be explicitly specified in different types of languages:

- Graphical notations
 - Semantic networks
 - Topic Maps
 - UML
 - RDF
- Logic based
 - Description Logics (e.g., OIL, DAML+OIL, OWL)
 - Rules (e.g., RuleML, LP/Prolog)
 - First Order Logic (e.g., KIF)
 - Conceptual graphs
- Probabilistic/fuzzy

Contemporary ontologies share many structural similarities, regardless of the language in which they are expressed. Most ontologies describe individuals (instances), classes (concepts), attributes, and relations.

Common components of ontologies include:

- Individuals: instances or objects (the basic or "ground level" objects), are the elements of the domain of discourse
 - Equivalent to constant in FOL
- Classes: collections, concepts, classes in programming, types of objects, or kinds of things. Sets of objects sharing certain characteristics.
 - Equivalent to unary predicates in FOL
- Relations: ways in which classes and individuals can be related to one another. Sets of pairs (tuples) of objects.
 - Equivalent to binary predicates in FOL
- Attributes: aspects, properties, features, characteristics, or parameters that objects (and classes) can have

Often there are also other elements like function terms, restrictions, rules, axioms and events.

In the following we're going to describe some different kinds of ontologies or data structures able to represent semantics. In particular *RDF* and its specialization in *OWL*, *WordNet* from Princeton, the *W4 model* for self-organization in pervasive computing system and finally the *sTuples model* as a semantic tuple space implementation.

2.1 RDF, RDFS & OWL

2.1.1 RDF

RDF(*Resource Description Framework*) is a standard model for data interchange on the Web. RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed.

RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a "triple"). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications

This linking structure forms a directed, labeled graph, where the edges represent the named link between two resources, represented by the graph nodes. This graph view is the easiest possible mental model for RDF and is often used in easy-to-understand visual explanations[7].

RDF identifies things using Web identifiers (URIs), and describes *resources* with *properties* and *property values*.

- A **Resource** is anything that can have a URI, such as "http://www.w3schools.com/rdf"
- A **Property** is a Resource that has a name, such as "author" or "homepage"
- A **Property value** is the value of a Property, such as "Jan Egil Refsnes" or "http://www.w3schools.com" (note that a property value can be another resource)

The following RDF document could describe the resource "http://www.w3schools.com/rdf":

```
<?xml version="1.0"?>

<RDF>
  <Description about="http://www.w3schools.com/rdf">
    <author>Jan Egil Refsnes</author>
    <homepage>http://www.w3schools.com</homepage>
  </Description>
</RDF>
```

The combination of a Resource, a Property, and a Property value forms a *Statement* (known as the subject, predicate and object of a Statement). An RDF Statement can be model as a three field tuple <Subject,Predicate,Object>.

RDF gives a formalism for meta data annotation, and a way to write it down in XML, but it does not give any special meaning to vocabulary such as `subClassOf` or `type`, they remain only properties subjected to interpretation with a non formal and defined meaning.

2.1.2 RDFS

RDFS(*RDF Schema*) is an extensible knowledge representation language, providing basic elements for the description of ontologies, otherwise called Resource Description Framework (RDF) vocabularies, intended to structure RDF resources[10].

RDF Schema allows to create Class and Property. Moreover specifying the concept of class and subclass, it lets you define hierarchies of classes. RDF Schema does not provide actual application-specific classes and properties. Instead RDF Schema provides the framework to describe the application-specific ones[7].

- **Class:** every resource described in RDF is instance of class *rdfs:Resource*
Subclass of *rdfs:Resource* are:

rdfs:Literal A literal or a string.
rdfs:Property The properties.
rdf:Class Is a class (object-oriented style).

- **Properties:**

rdf:type The type of a resource.
rdfs:subClassOf Class/subclass relationship.
 (Support multiple inheritance).
rdfs:subPropertyOf A property is a specialization of another one.
rdfs:seeAlso The resource is described somewhere else, too.
rdfs:isDefinedBy Indicate who make the assertion.

- **Constraint:**

rdfs:range Codomain
rdfs:domain Domain

In the example above there is the RDFS definition of the resource “horse” as a subclass of the class “animal”.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.animals.fake/animals#">

  <rdf:Description rdf:ID="animal">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
```

```

<rdf:Description rdf:ID="horse">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#animal"/>
</rdf:Description>
</rdf:RDF>

```

Problem with RDFS Even if RDFS is an efficient mode to describe the ontology we want to use in application specific domain, it results too weak to describe resources in sufficient detail because [8]:

- No localized range and domain constraints

E.g. Can't say that the range of hasChild is person when applied to persons and elephant when applied to elephants
- No existence/cardinality constraints

E.g. Can't say that all instances of person have a mother that is also a person, or that persons have exactly 2 parents
- No transitive, inverse or symmetrical properties

E.g. Can't say that isPartOf is a transitive property, that hasPart is the inverse of isPartOf or that touches is symmetrical
- Difficult to provide reasoning support

No "native" reasoners for non-standard semantics

2.1.3 OWL

OWL(*Ontology Web Language*) is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics[6].

OWL is a revision of the DAML+OIL web ontology language and has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full.

- **OWL Lite** supports those users primarily needing a classification hierarchy and simple constraints.
- **OWL DL** supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions. OWL DL is so named due to its correspondence with *description logics*, a field of research that has studied the logics that form the formal foundation of OWL.
- **OWL Full** is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary.

Each of these sublanguages is an extension of its simpler predecessor, both in what can be legally expressed and in what can be validly concluded. The following set of relations hold. Their inverses do not.

- Every legal OWL Lite ontology is a legal OWL DL ontology.
- Every legal OWL DL ontology is a legal OWL Full ontology.
- Every valid OWL Lite conclusion is a valid OWL DL conclusion.
- Every valid OWL DL conclusion is a valid OWL Full conclusion.

Important elements of OWL ontology are Class, Properties and Individuals.

In the next figures we can see the OWL Constructors (figure 2.1) and the OWL Axioms (figure 2.2).

Constructor	DL Syntax	Example	Modal Syntax
intersectionOf	$C_1 \sqcap \dots \sqcap C_n$	Human $\sqcap$ Male	$C_1 \wedge \dots \wedge C_n$
unionOf	$C_1 \sqcup \dots \sqcup C_n$	Doctor $\sqcup$ Lawyer	$C_1 \vee \dots \vee C_n$
complementOf	$\neg C$	$\neg$ Male	$\neg C$
oneOf	$\{x_1\} \sqcup \dots \sqcup \{x_n\}$	{john} $\sqcup$ {mary}	$x_1 \vee \dots \vee x_n$
allValuesFrom	$\forall P.C$	$\forall$ hasChild.Doctor	$[P]C$
someValuesFrom	$\exists P.C$	$\exists$ hasChild.Lawyer	$\langle P \rangle C$
maxCardinality	$\leq nP$	≤ 1 hasChild	$[P]_{n+1}$
minCardinality	$\geq nP$	≥ 2 hasChild	$\langle P \rangle_n$

Figure 2.1: OWL Constructors

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq \neg$ Female
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	{President_Bush} $\equiv$ {G.W.Bush}
differentFrom	$\{x_1\} \sqsubseteq \neg \{x_2\}$	{john} $\sqsubseteq \neg$ {peter}
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
equivalentProperty	$P_1 \equiv P_2$	cost $\equiv$ price
inverseOf	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent $^-$
transitiveProperty	$P^+ \sqsubseteq P$	ancestor $^+$ $\sqsubseteq$ ancestor
functionalProperty	$\top \sqsubseteq \leq 1P$	$\top \sqsubseteq \leq 1$ hasMother
inverseFunctionalProperty	$\top \sqsubseteq \leq 1P^-$	$\top \sqsubseteq \leq 1$ hasSSN $^-$

Figure 2.2: OWL Axioms

Example of OWL ontology: What does it mean?

```

Class(pp:old+lady complete
  intersectionOf(pp:elderly pp:female pp:person)
)
Class(pp:old+lady partial
  intersectionOf(
    restriction(pp:has_pet allValuesFrom(pp:cat))
    restriction(pp:has_pet someValuesFrom(pp:animal)))
)

```

Ans: Every old lady must have a pet cat.
(Because she must have some pet and all her pets must be cats.)

2.2 WordNet

WordNet® is a lexical English database developed under the direction of George Miller at Princeton University. In WordNet “nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations”[5]. WordNet defines every word according to the meaning(s) that it has in the English dictionary thus being an agreed-upon standard (the English language).

For a better explanation of what WordNet is, go to chapter 4.

2.3 W4 model

The *W4 model* concentrates its focus on pervasive computing systems. In these systems there is a continuous stream of data flowing from each part of the network to somewhere else. This large amount of data (received from sensors, RFIDs, agents and more) is often impossible to use because of its heterogeneity, its amount and the impossibility to automatically merge data in order to obtain knowledge. Indeed having data does not mean that we have informations.

As explained in [2] from Menezes, Zambonelli and Castelli, raw data generated by pervasive computing devices is often hard to be used. The standard approach is then to enriched it by relating and combining data items with each other so as to generate higherlevel, more expressive and easy to be managed, knowledge that, in turn, is also represented in some standard and processable way. Moreover in a context where data continue flowing through the system the need for a self-organizing approach to knowledge generation is clearly required by the inherent decentralized nature of pervasive computing, which asks for a data management process that is adaptive and based on local information. In order to obtain this self-organized behaviour the background idea is that the more some knowledge is queried, the more the generation of similar knowledge is reinforced.

2.3.1 The model

In the W4 model, every piece of contextual knowledge is represented as a simple *four-field tuple* which contains

- Who - represents who is doing some activity
- What - represents the activity
- Where - represents the place
- When - represents the time

In this model someone or something (*Who*) does/did some activity (*What*) in a certain place (*Where*) at a specific time (*When*). A W4 tuple represents the elementary information of a fact that occurs in the world (as a piece of data or knowledge atom)[2].

2.3.2 Data representation

Every field of the W4 tuple is denoted by a specific data representation

- Who - associates a subject to a fact. The field is represented by a *type:value* pair (e.g. *student:Mark*)
- What - describes the activity performed by the subject. The field is represented as a string containing a *predicate:complement* statement (e.g. *read:temperature=23*)
- Where - associates a location to the fact (both physical or logical). (e.g. *here* or *within:300m* or *latitude:longitude*)
- When - associates a time or time range to the fact (both exact or relative). (e.g. *2008-01-20* or *yesterday*)

The W4 model does not explicitly define the ontology to use, but lets the programmers decide what is the common a-priori ontology to be used. In the section 3.2 it is described how to compare and match w4 tuples.

Chapter 3

Comparing ontologies

3.1 Hmatch algorithm

H-Match[1] combines a measure of linguistic affinity and a measure of contextual affinity in order to evaluate a comprehensive measure of semantic affinity between ontology concepts using WordNet[5] ontology.

3.1.1 Ontology representation

The H-Match matching process uses a standard OWL ontology representation. For example, as described in figure 3.1, the Apple-q ontology specifies that the concept Apple is a fruit and a kind of food. Apples, in this ontology, origin from Italy. The Apple-o ontology describes two kinds of products, namely Edible fruits and Computer, where Mobile Computer is a kind of computers. These kind of ontologies are going to be used as a case study in the follow.

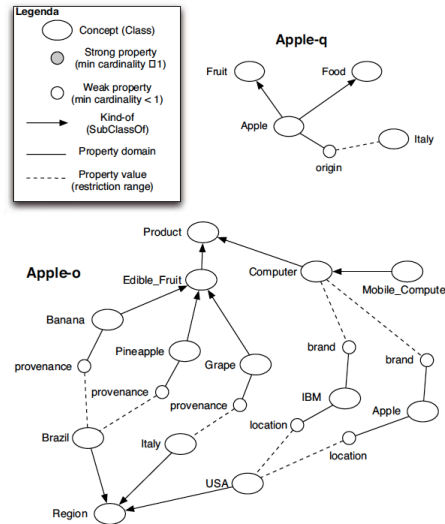


Figure 3.1: H-Model graphical representation of *Apple-q* and *Apple-o*

3.1.2 Linguistic features

In the H-Match algorithm terms can be basic or compound. Basic terms are all those terms that are included in WordNet, composed by one or multiple tokens. Compound terms are all those terms composed by more than one token that are not included in WordNet. Terminological relationships represented in the thesaurus are SYN, BT, NT, and RT.

- SYN (synonymy) denotes that two terms have the same meaning.
- BT (broader term) (resp., NT (narrower term)) denotes that a term has a more (resp., less) general meaning than another term.
- RT (related terms) denotes that two terms have a generic positive relationship.

These terminological relationships are derived from the relations defined in WordNet.

For the calculation of the matching affinity, a weight W_{tr} is associated with each terminological relationship. $W_{SYN} \geq W_{BT/NT} \geq W_{RT}$.

3.1.3 Contextual features

In H-Match they distinguish four semantic relations between two concepts c and c' , namely *same-as*, *kind-of*, *part-of*, and *associates*. The same-as relation denotes that c and c' are equivalent, while the kind-of and part-of relations denote that c and c' are related by a specialization relation and a composition relation, respectively. Finally, the associates relation denotes that c and c' are related by a generic positive semantic association.

For the calculation of the matching affinity, a weight W_{sr} is associated with each semantic relationship. $W_{same-as} \geq W_{kind-of} \geq W_{part-of} \geq W_{associates}$.

3.1.4 Matching process

The linguistic affinity provides a measure of similarity between the ontology concepts by considering their linguistic features, while the contextual affinity provides a measure of similarity by taking into account their contextual features. Four matching models (see figure 3.2), namely, *surface*, *shallow*, *deep*, and *intensive*, are defined for dynamically configuring H-Match for its adaptation to the semantic complexity of the ontologies to be compared [1].

3.1.5 H-match algorithm

According to the matching process chosen by the user and in order to evaluate the similarity/compatibility of terms, datatypes, properties and semantic relations, respectively, the H-match algorithm uses the basic matching functions:

- Term Affinity Function - the value of the highest-strength path of terminological relationships between two terms, if at least one path exists, and is zero otherwise. A path strength is computed by multiplying the weights associated with each terminological relationship involved in the path.

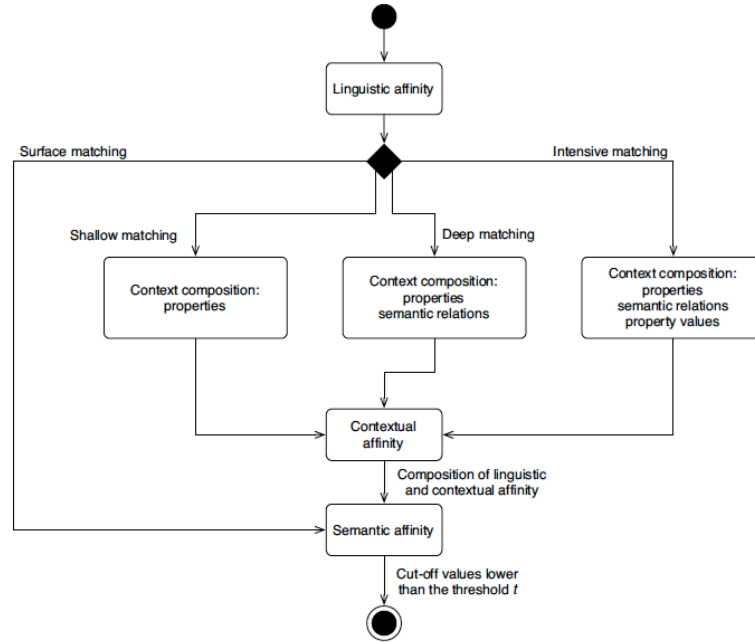


Figure 3.2: The matching process of H-Match

- Datatype Compatibility Function - one if data types of two concept are compatible, zero otherwise. (E.g. Compatible are `xsd:integer` ↔ `xsd:int`)
- Property and Semantic Relation Closeness Function - exploits the weights associated with context elements and returns a value in the range $[0,1]$ proportional to the absolute value of the complement of the difference between the weights associated with the elements.

3.2 W4 comparison

The W4 knowledge networks approach is based on the consideration that a relationship between knowledge atoms can be detected by a relationship (a pattern matching) between the information contained in the atoms fields[2]. In particular, for the W4 model, we can identify two types of pattern matching correlations between knowledge atoms:

- **Same value same field** : We can link together W4 tuples belonging to the same user, about the same place, activity or time (or, more in general, those W4 tuples in which the values in the same field match according to some pattern-matching function). Matching two or more *same value - same field* relationships, we can render complex concepts related to groups of W4 tuples.
- **Same value different field** : We can link atoms in which the same information appears in different fields. This kind of pattern matching can

be used for augmenting the expressive level of the information contained in the W4 tuples.

Exploiting this kind of matching is possible to correlate different and various W4 tuples in order to obtain a higher level knowledge. For example if in the system there are the tuples: (*student:Mark, - , room:computer lab, 2010-02-12 10am*) and (*class:MAS, - , room:computer lab, 2010-02-12 morning*) (where '-' means empty field) the system can create an higher level logical information stating that Mark is attending MAS class (*student:Mark, attending:MAS , room:computer lab, 2010-02-12 10am*).

Chapter 4

SOSTuS project

4.1 Semantic Tuple Matching

Tuples are ordered lists of elements. Matching of tuples in coordination systems has traditionally used a discrete (boolean actually) approach in which two tuples either perfectly match or they do not. Data organization, however, requires a soft matching approach in which the value of the matching varies continuously from 0 (completely different) to 1 (exactly the same). The reason is simple: data organization should consider the fact that tuples may not match but they are related semantically. Take for instance a simple example: if a process is consuming tuples related to feline data (cats, tigers), it is more likely to become interested in other animals than in tuples related to computing terms. The assumption is crucial in the definition of the organization because what we have just defined is an ontology for the relationship of terms: in our example, the concept “dog” is more related to “cat” than the concept “motherboard”.

Hence our first problem was not defining an algorithm able to compare two tuples but define an ontology to compare concepts. Indeed, in order to establish a correct semantic relationship between two concepts, the system bases itself on a *a priori* ontology which defines the meaning of a particular word. An ontology is the study of being or reality in general, as well as the basic categories of being and their relations. So an ontology defines what a concept is, how it could be described (because an ontology uses another ontology to describe itself, in general they are the same) and the relations that it has with other concept or the categories it belongs.

The problem of defining an ontology over the tuple and Linda model is very crucial. As said before, tuples in general are sets of data which could be heterogeneous both in type and in values. They can be represented by a tree in which at each level the root is the functor of the tuple and the children are the arguments (which could be other tuples). So, in general, is not so simple to map an ontology over this kind of model because the heterogeneity of data doesn't give us the global prospective of the relations that data have and how we can use them to describe a concept. For example having this tuple *person(John Smith, 40, teacher)* is could be clear for a human that this is the representation of the person John Smith who is 40 years old and is a teacher, but the mechanism we use to relate the terms is not explicit neither in the tuple nor in the ontology.

We could think that 40 are the age of working as a teacher, not the age of the person. To avoid this kind of problem we decided to use *first order tuples* in which every argument is *ground* (term with arity zero) and introducing a sort of default relation rule, every argument simply describes the main concept expressed by the functor.

4.2 Adoption of WordNet

The adoption of WordNet as the default ontology is due to the fact that the network created by the structure of the database links explicitly words and concepts which are related semantically. According to the notion of tuple described before, if every element of a tuple follows this ontology, it is possible to establish which semantical relation exists between two terms giving us the possibility to compare tuples.

There are several APIs available to interface the access to WordNet. We opted to use MIT's Java WordNet Interface (JWI) developed by Mark A. Finlayson [4] because it is extensively documented and connects the WordNet database to Java, the language we used for the implementation.

4.3 Organization Based on Soft Matching

The approach proposed here entails a matching mechanism used solely for the purpose of tuple organization and movement. The approach does not affect how tuples are matched from the point of view of the Linda-like primitives which continues to be based on an exact basis (a tuple either fully matches another or not). Yet, as we have pointed out, we would like the system to be self-organized so that once an agent requires the tuple the tuple is at relatively close proximity. In order to achieve this we propose an organization tuples based on a WordNet ontology as defined in Section 4.2.

Our proposal on the relatedness of two tuples is to use what we call *semantic matching*. The semantic matching of two tuples, t_1 and t_2 , denoting by $\delta(t_1, t_2)$ is the measure of how related these tuples are according to the adopted ontology. Here we assume that $0 \leq \delta(t_1, t_2) \leq 1$ meaning that the more two tuples are semantically related, the more the semantic match approximates 1 and the less related they are the closer the value is to 0.

The mechanism used to compare two tuples is relatively simple. It starts from the premise that the value of the semantic relationship between tuples depends on the semantic relationship between the words (terms/concepts) that exist as fields of one tuple and the words of the other tuple. We use this same approach in our everyday life; indeed, when people try to explain a concept, they normally base their explanation on an aggregation of related words that give more information about the concept requiring explanation. For example, when defining the animal *cat*, a person could add the information *animal*, *race*, *mammal*, and so on. The mechanism we use in this paper attempts to extract this idea in a more precise way.

Given a tuple $a(a_1, \dots, a_n)$ which defines the concept A and a tuple $b(b_1, \dots, b_m)$ which defines a concept B , whose arities are respectively n and m , the algorithm

makes the word-by-word comparison of all the possible combinations between every pieces of information in the tuple a with all the other ones in the tuple b , including both the tuple name and the tuple arguments. For example given a tuple $cat(animal)$ and $dog(animal)$, there are 3 possible combinations that are: $cat-dog$, $cat-animal$, and $animal-animal$.

In tuple-based systems, the name of the tuple is more important as a definition of the tuple than its arguments, therefore, when comparing tuples, one should try to avoid giving the same importance to every combination of elements in the tuple because what we want is to compare “main concepts” explained by the tuple. Given the algorithm described before there are three different kinds of relationships: *name-to-name*, *name-to-argument* and *argument-to-argument*. Assuming that the concept that a tuple wants to explain is defined by the name of the tuple while the arguments are adding informations that help the explanation of a concept, is obvious that the *name-to-name* relationship is the most important. Then to evaluate the semantic relation that exists between two tuples the *argument-to-argument* is lower important because if two tuples has the same name but different arguments it means that tuples are explaining the same concept but in a different way so the weight to assign to the comparison between arguments must be lower than the one between the names. The last kind of relationship is the *name-to-argument*. This kind of comparison is less important than the other ones because in this case we’re trying to compare words that have different roles: one is the concept explained by the tuple, while the other one is an further information which is something like a description.

Based on the textual argument above and after after some empirical experiments we decided to give a weight of 1.0 for the *name-to-name* (later called N_N) relationship, weight 0.8 for the *argument-to-argument* (later called A_A) relationship and weight 0.5 for the *name-to-argument* (later called N_A) relationship. After defining these weights and denoting by $\Delta(w_1, w_2)$ the semantical relationship between two words, we have that:

$$\delta(a(a_1, \dots, a_n), b(b_1, \dots, b_m)) = N_N\Delta(a, b) + N_A\left(\sum_{i=1}^n \Delta(a_i, b) + \sum_{j=1}^m \Delta(a, b_j)\right) + A_A \sum_{i=1}^n \sum_{j=1}^m \Delta(a_i, b_j), \quad (4.1)$$

as the formula used to calculate the semantic matching between two tuples.

In order to obtain a value of the semantic matching value included by 0 and 1 which has to be independent by the lenght of the tuples we normalized the formula (4.1) dividing by the max semantic matching value γ which can be reached when the relationship that exists between every word is the synonym one and can be calculated by the equation:

$$\gamma(a(a_1, \dots, a_n), b(b_1, \dots, b_m)) = N_N + N_An + N_Am + A_Anm, \quad (4.2)$$

where n and m represent the arities of the two tuples.

In this way we obtain the λ function

$$\lambda(a(a_1, \dots, a_n), b(b_1, \dots, b_m)) = \frac{\delta(a(a_1, \dots, a_n), b(b_1, \dots, b_m))}{\gamma(a(a_1, \dots, a_n), b(b_1, \dots, b_m))}. \quad (4.3)$$

The explanation above describes the entire process but we still have not defined how the Δ function works. To correctly understand the mechanism we proposed we need to further discuss the the semantic relations that a words could have between themselves.

4.4 Semantic Relationships

WordNet defines several relationships among words. More precisely, WordNet looks are relationships between *synsets*, which are groups of words that are synonyms. However, in our ontology we want to define the relationship among every word and not synsets so we included “synonym” as a possible semantic relationship. The relationships we use in our matching calculation are:

Synonym: A synonym is a word that has the same meaning of another one. This relation could be explained by the “*same as*” relationship. Examples of synonym pairs are (*picture, photo*) and (*table, mesa*).

Hyponym: A hyponym is a word or phrase whose semantic range is included within that of another word, its hypernym. Often in computer science this term can be explained using the “*is a*” relationship. For example, *red is a hyponym of color*.

Hypernym: A hypernym of a word is another word that explains a more general concept than the word which it’s referred to does. It’s the opposite of Hyponym. Hence *color is a hypernym of red*.

Meronym: A meronym denotes a constituent part of, or a member of something. In knowledge representation languages, meronym is often expressed as “*part of*” relationship. For instance, *eye is a meronym of face*.

Holonym: A holonym denotes the whole part. It’s the opposite of Meronym. It could be explained by the “*contains*” relationship. Similarly to the example before, *face is a holonym of eye*.

Figure 4.1 shows a simple example in WordNet and the relation between the terms. We show only the relations above since these are the only ones used in the algorithm we use in our experiments.

So based on the relationships above, we use WordNet as mechanism to define an ontology that describes the semantic relationship of two given terms in a tuple. The mechanism we devised is loosely based on the work of Castano et al.[1]. We start systematically by searching whether two given terms/words are synonyms, then hypernyms or hyponyms, and later if they are meronyms or holonyms. The motivation is that the “*same of*” relation between terms is a stronger relation than the “*is a*” which in turn is a stronger than the “*part of*” or “*contains*”. All these relationships are mutually exclusive, so when the algorithm find out the existence of a relationship, it stops without proving the existence of other ones. Again similar to the work of Castano et al. we attributed a weight to the type of relationships varying in the range [0.0, 1.0]

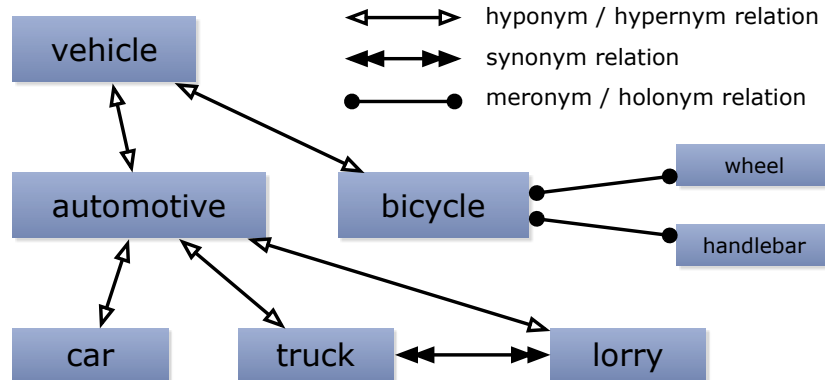


Figure 4.1: Something

where 1.0 denotes the strongest relationship. We used 1.0 for the synonyms, 0.8 for hypernyms and hyponyms, 0.5 for meronyms and holonyms.

Differently from the work of Castano et al. our algorithm does not create a thesaurus able to contain the terms to compare, but the algorithm tries to find the relationships at run time. We don't want to compare ontologies represented by graphs but concept represented by tuples. In our algorithm doesn't exist a sort of information typing, but every piece of information has a different weight according to the role it has in the tuple (name or argument) and the kind of relationship (N_N, A_A or N_A) it is involved. Our algorithm aims to find a number which represent how a concept is related to another one more than establish if two ontologies represent the same concept.

If two words are not directly related, the algorithm tries to find the relations that exists recursively, so the result given by the Δ function is the weight of the relationship found multiplied how many the recursion has done, that is, how distant in the network the words are apart. Given two words the algorithm keeps one fixed and tries to search if it is present in the synonyms (or meronyms or hypernyms respectively) of the other one. Obviously this mechanism is repeated twice exchanging the role of the words. In case of different values (due to the not exact symmetry of the WordNet) we take the largest one since it represents the closest relationship between the two words.

Given the vast size of WordNet and its obvious incompleteness (it is nearly impossible to represent all relationships between every pair of words), we found that only in few instances we will get two words that are semantically related in a direct way as described before. Take for instance a simple example. If one thinks about *dog* and *cat* concepts, it is obvious that they are related in some way even if they do not represent the same concept or do not to fall in any of the categories listed earlier in this section. Indeed they are both hyponyms of another concept, *animal*. Our approach then needs to try to take this into consideration so that these two terms are considered to be more related between themselves than to terms such as *computer* and *banana*. Following this reasoning we decide to find the *Common Lowest Hypernym (CLH)* of two words. In this case, after trying to find the straight relationships between two word, if there's

no relation, we follow both two paths from every word with his hypernyms to find the common word that is the a hypernym of both words according to the WordNet. At least the algorithm stops at the word *entity* which is the root of the WordNet tree.

For example using WordNet 3.0 [5] and searching for the first hypernyms of the word *dog* we find

```
dog, domestic dog, Canis familiaris
=> canine, canid
    => carnivore
        => ...
```

Searching for the first hypernyms of cat we find

```
cat, true cat
=> feline, felid
    => carnivore
        => ...
```

In the example above, we see that *carnivore* is the first hypernym that they have in common. It is not a direct hypernym but one that is distant 2 steps from the words *dog* and word *cat*. So the total path between the given words is made of 4 steps. In this case $\Delta(dog, cat) = 0.8^4 = 0.4096$. Note that we use the sum of the distances between the words and the hypernym because we want the value of $\Delta(dog, carnivore)$ —the direct comparison of a word and a hypernym—to give us a better similarity/matching.

In general given two words w_1 and w_2 and defining l_1 and l_2 the distances that respectively there are between the words and their LCH in term of steps, the Δ function is

$$\Delta(w_1, w_2) = 0.8^{l_1+l_2} \quad (4.4)$$

4.5 Semantic and Lexical Matching

Given Equation (4.3), it is possible to see that two tuples must be completely made by all synonyms to obtain a 100% of semantic match. This is correct, because the more two tuples tries to explain the same concept in the same way, the more the semantic match must be closer to *1.0*. If on one hand the semantic match offers the possibility to reach to the maximum with tuples made all by synonyms, on the other hand we want to give some importance also to the lexicographic match which takes in account the word-by-word equality of every tuple, in order to obtain a higher value even if tuples are not composed by synonyms, but are lexically equal. To make this, we introduced the σ function, which is a weighted average of a word-by-word lexicographic match.

$$\sigma(a(a_1, \dots, a_n), b(b_1, \dots, b_m)) = \begin{cases} 0, & \text{if } n \neq m, \\ \frac{\sum_{i=0}^n eq(a_i, b_i)}{n+1}, & \text{if } n = m \end{cases} \quad (4.5)$$

where a_0 and b_0 denote the tuple names and

$$eq(a, b) = \begin{cases} 1, & \text{if } a = b \\ 0, & \text{if } a \neq b \end{cases} \quad (4.6)$$

is the equality function. In this function the equal (=) symbol has the lexicographical meaning. To be more precise, it means that two words have to be written in exactly the same way to be considered equal.

In conclusion, using (4.3) and (4.5) we create the weighted average to calculate the semantic match using α as the weight.

$$\Lambda(a(a_1, \dots, a_n), b(b_1, \dots, b_m)) = \alpha \lambda(a(a_1, \dots, a_n), b(b_1, \dots, b_m)) + (1 - \alpha) \sigma(a(a_1, \dots, a_n), b(b_1, \dots, b_m)) \quad (4.7)$$

After some experiments we decided to assign to α the value of 0.75.

The following example show the console output comparing the tuple *cad(dog)* to itself.

```
Comparing cat(dog) - cat(dog)
N_N cat-cat val: 1.0
N_A cat-dog val: 0.20480000000000004
N_A dog-cat val: 0.20480000000000004
A_A dog-dog val: 0.8
SEMANTIC matching is 0.7891428571428571
LEXICAL matching is 1.0
Calculating: 0.75 * 0.7891428571428571 + 0.25 * 1.0
Tot: 0.8418571428571429
```

Bibliography

- [1] S. Castano, A. Ferrara, and S. Montanelli. Matching ontologies in open networked systems: Techniques and applications. *Journal on Data Semantics (JoDS)*, V, 2005.
- [2] Gabriella Castelli, Ronaldo Menezes, and Franco Zambonelli. Self-organized control of knowledge generation in pervasive computing systems. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1202–1208, New York, NY, USA, 2009. ACM.
- [3] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, June 1993.
- [4] Java WordNet Interface. Home page: <http://projects.csail.mit.edu/jwi/>.
- [5] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller. Wordnet: An on-line lexical database. *International Journal of Lexicography*, 3:235–244, 1990.
- [6] W3 OWL Recommendation. Home page: <http://www.w3.org/tr/owl-features/>.
- [7] W3 RDF Recommendation. Home page: <http://www.w3.org/rdf/>.
- [8] Peter F. Patel-Schneider Sean Bechhofer, Ian Horrocks. Tutorial on owl. *ISWC, Sanibel Island, Florida, USA*, oct 2003.
- [9] Wikipedia KBS Page. Home page:
http://en.wikipedia.org/wiki/knowledge-based_systems.
- [10] Wikipedia RDFS Page. Home page: <http://en.wikipedia.org/wiki/rdfs>.