OntoKBEval:

A Support Tool for OWL Ontology Evaluation

Qing Lu

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

September 2006

Your file Votre référence
ISBN: 978-0-494-20778-9
Our file Notre référence
ISBN: 978-0-494-20778-9

Abstract

OntoKBEval: A Support Tool for OWL Ontology Evaluation

Qing Lu

The Support Tool for OWL Ontology Evaluation (OntoKBEval) has been developed to apply Description Logics reasoning to ontology evaluation by deriving information from knowledge bases. The principal objective is to evaluate ontologies and to present results using a user-friendly visualized interface to users.

OntoKBEval offers hierarchical diagrams describing the structure of OWL-DL ontologies divided into the description logics view of TBoxes and ABoxes. Furthermore, corresponding detailed information is offered for these structures to guide further evaluation directions. The three main methods for ontology evaluation are: (i) quick-view ontology evaluation (providing a keyword search for named concepts) (ii) general ontology evaluation (performing a more comprehensive TBox- and ABox-based evaluation) (iii) multi-file ontology evaluation (facilitating multiple OWL ontology evaluation by offering basic TBox and ABox information). The implementation relies on the OWL-DL reasoner RacerPro to support reasoning functionalities.

Acknowledgements

I would like to express my deepest appreciation to my supervisor, Dr. Volker Haarslev for all his guidance, support and encouragement. All the great help leads me to meet the challenge throughout my research work.

I would like to give great thank to Dr. Christopher J. O. Baker for all his advice and help in my thesis work.

My special thanks go to my parents, all my colleagues and all my friends for all their kindly support, encouragement and understanding. And thanks to Concordia Community for offering a great environment for my study and research.

Table of Contents

List of Figures

List of Tables

1. Introduction

In this chapter we introduce background knowledge related to our research work. It includes the topics description logics, ontologies, OWL, and our motivation to build OntoKBEval.

1.1 Description logics

The main research purpose in knowledge representation is to provide theories and systems for expressing structured knowledge and for accessing and reasoning with it in a principled way. Description logics (DLs) are considered the most important knowledge representation formalism unifying and giving a logical basis to the well-known traditions of Frame-based systems, Semantic Networks and KL-ONE-like languages, Object-Oriented representations, Semantic data models, and Type systems [10].

A DL knowledge base consists of two components: TBox and ABox. Figure 1-1 describes the architecture of a knowledge representation system based on Description Logics.

In order to describe DL more clearly, we use a simple ontology '*family*'. It is in a format of a Racer file whose details are listed in Appendix A1.

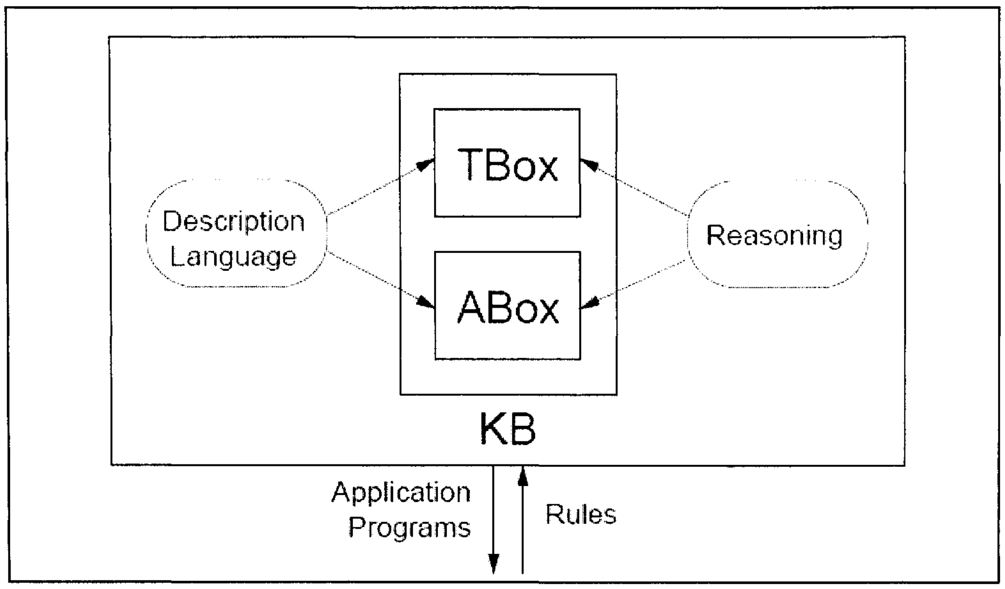
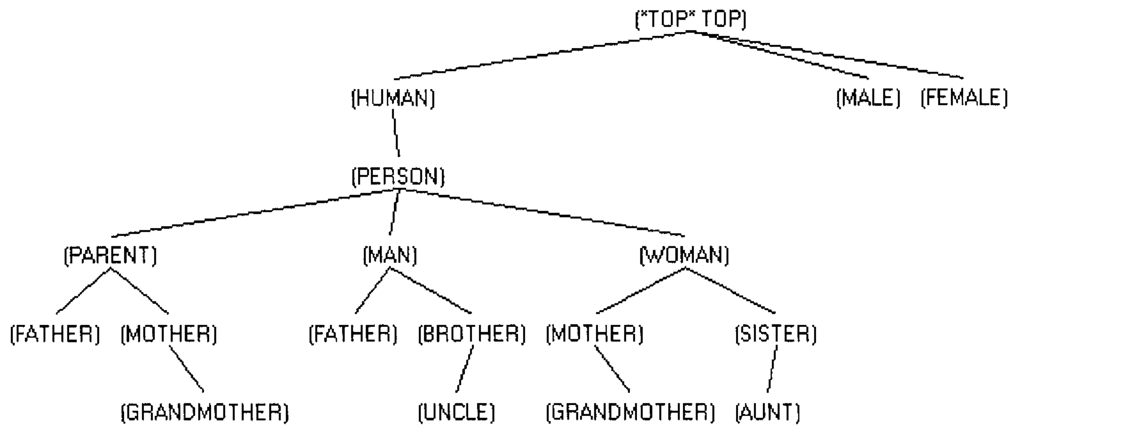


Figure 1-1 Architecture of a knowledge representation system based on Description Logics copied from [30]

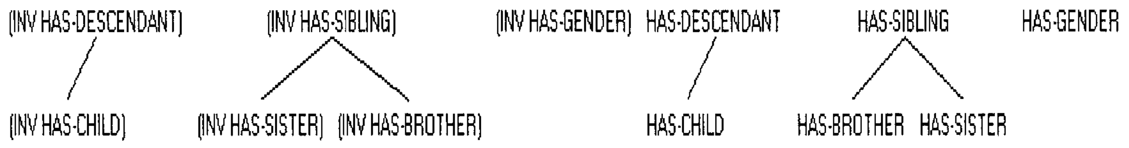
1.1.1 TBox

A TBox (Terminology Box) consists of a set of concept axioms, where a concept denotes a set of individuals, roles, which denote binary relationships between individuals [30].

In a TBox, the concept definition of a new concept is in terms of other defined concepts, for example, $woman \equiv person \sqcap female$. In particular, the classification of concepts enables one to place a new concept in a proper place in a taxonomic concept hierarchy. Concepts axioms help describe the classification. In general, TBox axioms have the following form: $C \sqsubseteq D$ $(R \sqsubseteq S)$ or $C = D$ $(R = S)$, where C, D are concepts and R, S are roles. The former represents the subsumption relationship between C and D (or R and S); the latter is the relationship of equivalence. In this case, we can have hierarchies for concepts and roles, which provide the basic structures of the TBox. Figure1-2 shows the hierarchies of the small '*family*' ontology.



(a) Concept hierarchy



(b) Role hierarchy

Figure 1-2 '*Family*' TBox hierarchies

For small ontologies, the hierarchies similar to those in Figure 1-2 are readable and comprehensible. However, as we have ontologies with hundreds or thousands of concepts/roles in TBoxes, this kind of diagrams cannot easily help and might become unreadable. We will describe one possible solution in this thesis.

1.1.2 ABox

An ABox (Assertion Box) contains assertions about named individuals in terms of concepts and roles [30].

It contains two main assertional axiom types: (1) concept assertions for individuals, *a:C*, where *a* is an individual and *C* is a concept; (2) role assertions for individual pairs, *(a,b):R*, where *a* and *b* are individuals and *R* is a role. In this case, we need to consider the individuals w. r. t. concept and role assertions. For example, in the '*family*' ontology, we have five individuals: *Alice*, *Betty*, *Charles*, *Doris*, and *Eve*. We can characterize the individual '*Betty*' as an instance used in two ways: (1) as a mother *Betty: Mother*; (2) and as Doris mother *(Betty, Doris): has-child.* Figure 1-3 describes the ABox hierarchy of '*family*' ontology based on role assertions.

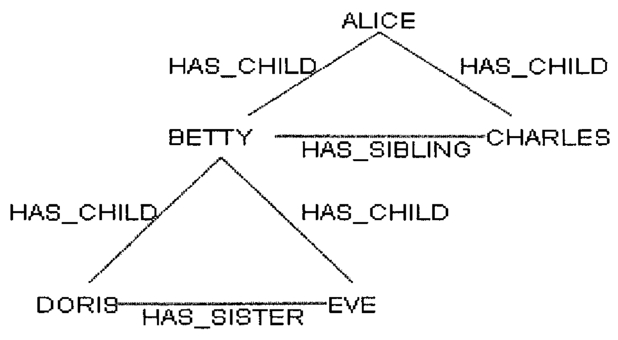


Figure 1-3 '*Family*' ABox hierarchy

Other ways of ABox presentation are discussed in Chapter 3.

1.2 Ontology

The term ontology originates from philosophy and is about the subject of existence. In our context, ontologies are explicit formal specifications of the terms in a domain and relations among them [16]. They are used to share a common understanding of the information structure among people or software agents and enable reuse of domain knowledge. In recent years, work on ontologies has migrated from Artificial Intelligence to other kinds of domains in research or industries.

Ontologies contain a formal explicit description to define a knowledge base consisting of concepts (or classes) in a domain, roles (or properties, or slots) between instances of concepts, restrictions (or facets) on roles and together with a set of individuals (or instances) to define a knowledge base. Concepts are the focus for most ontologies and roles describe properties of concepts and individuals. Ontologies can help to share semantic understanding of information, make assumptions explicit, separate domain knowledge form the operational knowledge [1], clarify the term definitions, help reasoning knowledge on indicated rules, etc.

In order to support ontology building, various ontology languages have been developed: Resource Description Format (RDF) [25], RDF schema (RDFS) [8], and DAML+OIL [40], to the most recent popular Web Ontology Language (OWL) [43]. In our research, we solely focus on OWL ontologies.

1.3 Web Ontology Language (OWL)

In late 1990s, work on ontologies has become a research area targeting the Web. After the release of OWL (Ontology Web Language) as a semantic web standard, it has been considered a useful standard 'for formally specifying knowledge in the web and recently renewed the focus on ontologies' [3]. OWL facilitates a greater capability of machine interpretation of Web contents than XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics [33]. OWL has three sublanguages: OWL Lite, OWL DL, and OWL Full.

In order to make our work better understandable, we will describe the axioms about concepts, roles and individuals supported by OWL.

1.3.1 Concept axioms in OWL

Concept axioms are used to define concepts. We can use *owl:Class* with a concept identifier, ex. *<owl:Class rdf:ID="Human"/>*. However, this is only a simple declaration and does not give much information about the concept. Hence, concept axioms should contain additional components to state their characteristics.

Together with concept declarations, OWL contains three helpful language constructs to form concept axioms: *rdfs:subClassOf* (to state that a concept is described as a subset of another concept), *owl:equivalentClass* (to state that a concept is an equivalent of another concept), *owl:disjointWith* (to state that a concept has no common members with another concept).

1.3.2 Role axioms in OWL

A role axiom defines characteristics of a role, for example, *<owl:ObjectProperty rdf:ID="hasParent"/>*. Four kinds of constructs for role axioms are supported in OWL as follows:

- RDF Schema constructs: *rdfs:subPropertyOf*, *rdfs:domain* and *rdfs:range*
- relations to other properties: *owl:equivalentProperty* and *owl:inverseOf*
- global cardinality constraints: *owl:FunctionalProperty* and *owl:InverseFunctionalProperty*
- logical property characteristics: *owl:SymmetricProperty* and *owl:TransitiveProperty*

These constructs allow one to define roles in more detail (for example, *has_pet* is an inverse role of *is_pet_of*).

1.3.3 Individual axioms in OWL

Two types of individual axioms are discussed here.

- Individual axioms about concept membership and role values

  For example,

  ```
  < Mother rdf:ID= "Alice">
      <has_child rdf:resource= "# Betty">
  < /Mother>
  ```

  The first line indicates that '*Alice*' is an instance of concept '*Mother*' and the second line says '*Alice*' has a role assertion *(Alice, Betty):has_child.*

- Individual axioms about identity

For example:

```
<Mother rdf:ID="Alice">
    <owl:differentFrom rdf:resource="#Betty"/>
</Mother>
```

It indicates that '*Alice*' and '*Betty*' are two different individuals.

1.4 Ontology evaluation methodology and motivation

1.4.1 Survey of ontology evaluation methodology

Many ontologies seem to support a variety of Semantic Web services. We face ontologies in all kinds of domains on the web, such as gene, bioinformatics, plant, languages, etc. Ontologies on the Web range from large taxonomies categorizing Web sites (such as on Yahoo!) to categorizations of products for sale and their features (such as on Amazon.com) [31].

Even for one subject in a certain domain such as bioinformatics, the Gene ontology [39], and so on, we can find quite a few ontologies, which contain from tens of concepts to millions of concepts and potential users have to evaluate and select ontologies that best match their current interests. Knowledge engineers are regularly searching for ontologies in the web in order to incorporate such ontologies into their systems and choose the ontologies by relying on their experience and intuition [22]. We have an idea that good ontologies are the ones that serve their purpose [7].

Further, the creation of ontologies depends on their designers' understanding and view of the information in a certain domain. Potential users of ontologies need to assess the quality and possible benefits of available ontologies. Thus, ontology evaluation becomes an important issue.

Until now, various approaches for ontology evaluation have been considered depending on the specific kind of ontologies and purposes. Broadly speaking, most evaluation approaches fall into the following categories [5]:

- those based on comparing an ontology to a "golden standard" (which may itself be an ontology; e.g. [28]);
- those based on using an ontology in an application and evaluating the results (e.g. [35]);
- those involving comparisons with a source of data (e.g., a collection of documents) about the domain to be covered by the ontology (e.g. [7]);
- those where evaluation is done by humans who try to assess how well an ontology meets a set of predefined criteria, standards, requirements, etc. (e.g. [26]).

In addition, ontology evaluation approaches can be grouped based on the level of evaluation [5]:

- lexical, vocabulary, or data layer—to focus on concepts, roles, and individuals in an ontology
- hierarchy or taxonomy—to build b a '*is-a*' relationship between concepts

- other semantic relations—to contain other relationships other than '*is-a*' relationship
- context or application level—some definitions in an ontology related with other ontologies
- syntactic level—to match the syntactic requirements of the building language
- structure, architecture, design—to meet particular design requirements.

Table 1-1 gives a summary of the approaches mentioned above.

	Approach to evaluation			
Level	Golden standard	Application-based	Data-driven	Assessment by humans
Lexical, vocabulary, concept, data	x	x	x	x
Hierarchy, taxonomy	x	x	x	x
Other semantic relations	x	x	x	x
Context, application		x		x
Syntactic	x[1]			x
Structure, architecture, design				x

Table 1-1 An overview of approaches to ontology evaluation

"Golden standard" in the sense of comparing the syntax in the ontology definition with the syntax specification of the formal language in which the ontology is written (e.g. RDF, OWL, etc.). [5]

1.4.2 Motivation of our work

As we mentioned in the previous section, with the emergence of more and more ontologies, users have to select those, which can meet their needs. This leads to the idea of ontology evaluation. Furthermore, as many ontology evaluation methods are proposed, it is necessary and important to have tools available to use these methods in order to facilitate evaluation. These reasons encouraged us to look for better methodologies and tools for ontology evaluation.

Currently, there are a number of ontology evaluation methods emerging. Some methods of evaluation are intended for the ontology pre-modeling stage to test for inconsistencies. Some are concerned with checking for consistency and logical errors. The others are helpful for the evaluation of ontologies after their release.

We note that the term ontology refers to a DL KB, i.e., a set of ABox and/or TBox axioms expressing knowledge regarding a domain of interest and the terms ontology and DL KB will be used interchangeably [13]. Furthermore, the term ontology is commonly used to refer to a structure capturing knowledge about a certain area via providing relevant concepts and relations between them [6]. Thus, in our ontology evaluation we focus on TBoxes and ABoxes, which are the components of Description Logics knowledge bases. For the TBoxes, we consider hierarchies of concepts and roles; for the ABoxes, we group the individuals by concept assertions and by role assertions.

In our research, we try to evaluate ontologies in both qualitative and quantitative ways with the help of DL technology. For TBoxes and ABoxes, the distributions of their elements are represented in structured hierarchical figures. In this case, we focus on the structure while ignoring detailed concept, role and individual names. We also provide corresponding functions to browse detailed information on TBoxes and ABoxes, so, users can decide where to focus their further evaluations. The OntoKBEval system provides a graphical interface to help users evaluate their ontologies.

In this thesis, we address our methods and the ontology evaluation tool by using the OWL reasoner RacerPro version 1.9.

1.5 Thesis organization

In this thesis, we try to cover all the related methodologies, and the implementation of our ontology evaluation solution. In Chapter 2 the reasoner RacerPro is described as well as RacerPorter. In Chapter 3 we give potential users' requirements for ontology evaluation. In Chapter 4 we introduce some recent ontology evaluation support works about ontology evaluation. In Chapter 5 our OntoKBEval system is described in detail including its design, functions, interfaces, etc. In Chapter 6 results testing the OntoKBEval system are analyzed. Chapter 7 includes our conclusion and future work. The appendix provides a Racer file and the Racer commands used to implement the system.

2. Reasoner tools

This chapter describes the necessary tools to help with our OntoKBEval design, implementation, and testing.

2.1 RacerPro

RacerPro (Renamed ABox and Concept Expression Reasoner Professional) is one of the fastest OWL DL reasoners. As the name suggests, the origins of RacerPro are within the area of description logics [36]. It can be used as a reasoning engine for ontology editors such as Protégé [32].

RacerPro offers reasoning services for multiple TBoxes and ABoxes encoded as OWL DL knowledge bases [18] and supports the specification of general terminological axioms. A TBox may contain general concept inclusions (GCIs) to present the subsumption relationship between two concepts. RacerPro can also deal with multiple definitions or even cyclic definitions of concepts.

RacerPro implements the HTTP-based quasi-standard DIG for interconnecting DL systems with interfaces and applications using an XML-based protocol [4]. Most of the functions specified in the earlier Knowledge Representation System Specification (KRSS) can be used with RacerPro.

Given a TBox, various kinds of queries can be answered. Based on the logical semantics of the representation language, different kinds of queries are defined as inference problems. We list only the most important ones for TBox as follows:

- Concept consistency, for example, if *brother*$\equiv$*person*$\cap$*female*, an inconsistency exists
- Concept subsumption, e.g., a *mother* is always a *parent*
- All inconsistent concept names mentioned in a TBox
- Get the parents and children of a concept, e.g., a *person* is a parent of a *man* and a *man* is a child of *person*
- Get the parents and children of a role, e.g., *has-sibling* is a parent of *has-brother* and *has-brother* is a child of *has-sibling*

If an ABox is given, the following types of queries are possible:

- Check the consistency of an ABox, for example, the knowledge specified for the individual *Betty* is consistent with the concept *mother* as in *Betty:mother*
- Instance testing to make sure if an individual is an instance of a certain concept
- Instance retrieval to find all individuals proven to be the instance of a certain concept
- Retrieval of individuals that satisfy certain conditions
- Computation of the direct types of an individual, for example *mother* is the most specific concept of which *Betty* is an instance
- Computation of the fillers of a role with reference to an individual. After computing these fillers, the function 'all-role-assertions' returns also the implicit role fillers
- Check if certain concrete domain constraints are entailed by an ABox and a TBox

RacerPro provides another semantically well-defined query language (nRQL [19], new Racer Query Language), which also supports negation as failure, numeric constraints w.r.t. attribute values of different individuals, substring properties between string attributes, etc [36].

As to the special OWL features such as annotation and datatype properties, special OWL querying facilities have been incorporated into nRQL. The query language OWL-QL [4] is the W3C recommendation for querying OWL documents. nRQL has been used as a basic engine for implementing a very large subset of the OWL-QL query language [36].

We use many Racer commands in our ontology evaluation tool and the details of each command are shown in Appendix A2.

2.2 RacerPorter

RacerPorter is the native, graphical user interface for RacerPro [34]. RacerPorter uses the TCP/IP network interface to connect to RacerPro servers and helps to load ontologies, view TBoxes and ABoxes, switch among taxonomies, and send queries or commands to RacerPro to get reasoning results. It facilitates RacerPro users to operate on ontology files.

During the process of implementing our ontology evaluation tool, we compared results from RacerPorter with our results to make sure our results are correct.

3. Requirements and scenario

In this chapter, we analyze users' requirements about OWL ontology evaluation and in which way they can be implemented.

- Why to use a reasoner?

As ontologies can provide lots of information, more and more researchers and engineers use ontologies for searching related information in their interest domains. However, only few of these people have background knowledge about the OWL language and it is hard to get useful data directly from reading OWL code. So a reasoner should be used to help humans better understand OWL ontologies. For the OWL DL reasoner RacerPro, it is convenient to use by sending meaningful commands, for example, 'all-roles' to get all the roles in an ontology. Furthermore, it can load a large ontology relatively fast.

- How to know domains that ontologies belong to?

When ontology users get an ontology, first, it is important to know which domain this ontology is in. An efficient way is to look through all concepts, which are fundamental components of an ontology. For example, an ontology contains many concepts about '*gene*', '*DNA*', '*cell*', '*enzyme*' etc. Then this ontology is very likely about biology.

- How to know if ontologies contain information related with a certain subject?

For example, users' most interest subject is '*enzyme*'. After users select an ontology about biology, they may get a lot of results about '*enzyme*'. Users can browse all the related concepts with the '*enzyme*' concept with parent-children relationships. Figure 3-1 is an example (ontology: '*FungalOntology.owl*' [37]).

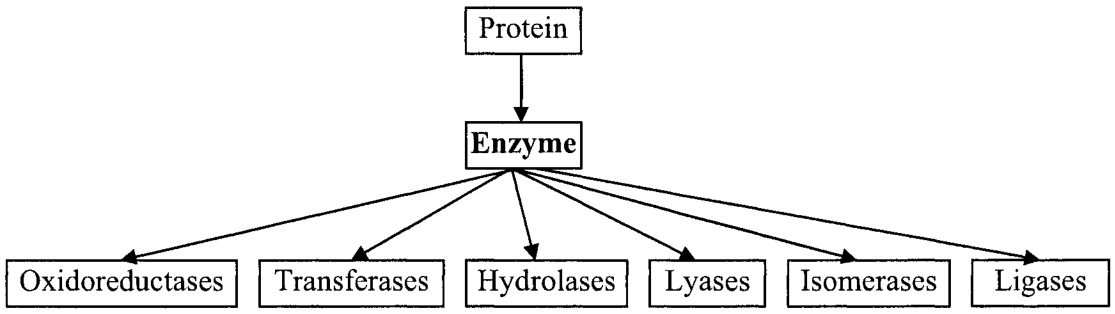


Figure 3-1 Concepts related with '*enzyme*'

- Why do we need ontology structures to help in ontology evaluation work?

Usually users get detailed information about ontologies indicated by concepts, roles, and individuals. However, ontologies are built with descriptions of relationships among these elements. So their structures are also very important parts, which cannot be ignored when we evaluate ontologies.

➢ TBox structures

Hierarchies of TBoxes (concepts and roles) are based on the '*is-a*' relationship. As shown in Figure 3-1, we get related concepts of '*enzyme*'. This is only a small part of the whole hierarchy.

From ontology hierarchies, which provide overviews of ontology structures, users know how big ontologies are and how concepts and roles are distributed. Every OWL ontology has a '*top*' at the first level and a '*bottom*' at the last level. For example, the more direct children of '*top*' exist, the less subsumption relationships are found among concepts besides those with the '*top*' concept, in which case, users may find less information about indicated concepts. So among ontologies with similar numbers of concepts, it is more likely to obtain more knowledge from the ontology having the least number of concepts as direct children of '*top*'. This also gives hints for ontology creators to avoid defining most concepts as direct children of '*top*'. It also helps further ontology optimization.

➢ ABox structures

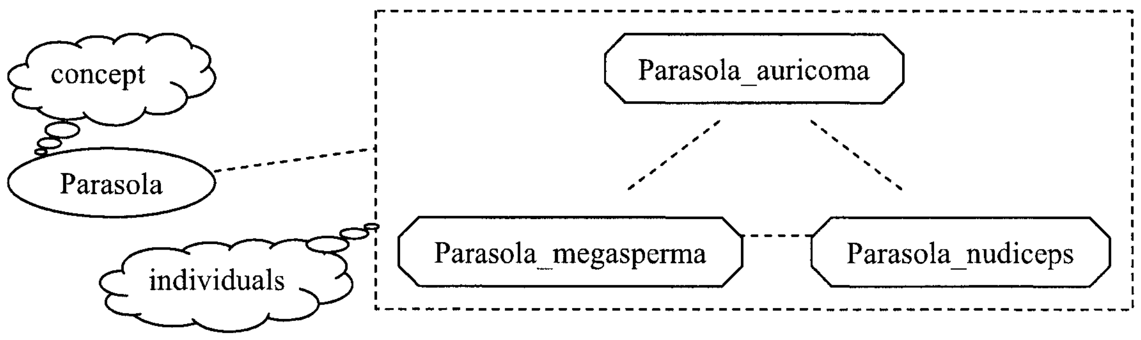
For ABoxes, it is not sufficient to only describe concept assertions and role assertions in words. For example, consider the following two role assertions taken from the '*FungalOntology*':

(Rhizopus_stolonifer,Pectinase):Has_been_reported_to_have_enzyme

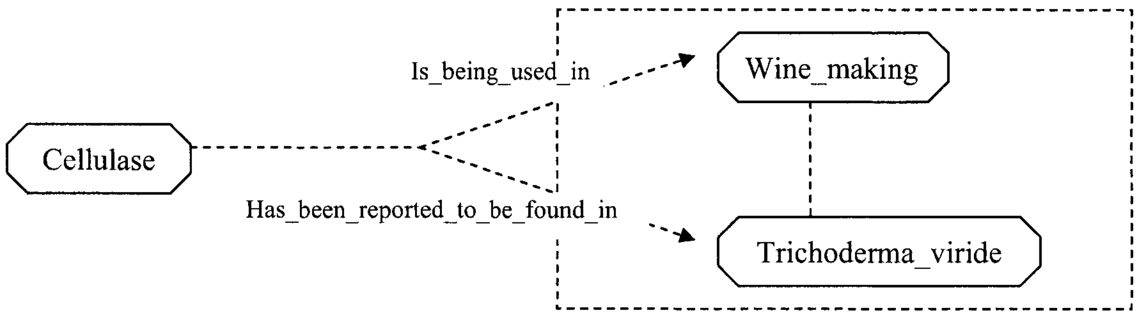
(Alternaria_citri,Pectinase):Has_been_reported_to_have_enzyme

These two assertions both contain the individual '*Pectinase*', so the individuals '*Rhizopus_stolonifer*' and '*Alternaria_citri*' have certain relationships because they are both related with '*Pectinase*'. As this kind of relationship is very common in ABoxes, users always want to know how many individuals have this kind of relationship in ontologies and how the relationship is built. If the individuals met users' interests have more related individuals, the more knowledge users may get from this ABox. So it is also important to have an overview structures for ABoxes.

Therefore ABox structures should not only describe how big ABoxes are, but how individuals are related with others as well. Compared with TBox hierarchies, ABox structures are much more difficult to present. Relationships are based on different conditions. Some are deduced from concepts assertions, some from role assertions, and others from both concept assertions and role assertions. Figure 3-2 (a) shows the individual relationships based on concept assertions and Figure 3-2 (b) illustrates the individual relationship based on role assertions.



(a) Individual relationship based on concept assertions



(b) Individual relationship based on role assertions

Figure 3-2 Individual relationships based on ABoxes

4. Other work on ontology evaluation support

In this chapter, we describe related work and tools in the domain of ontology evaluation. These tools or methodologies focus on different facets of ontologies. We compare them with OntoKBEval in Section 6.4.

4.1 ODEval

ODEval [9] is a tool to evaluate concept taxonomies of RDF(S), DAML+OIL, and OWL from a knowledge representation point of view. It is a complement for ontology parsers and ontology platforms.

ODEval can automatically detect possible problems in ontology concept taxonomies (inconsistency: circularity issues and partition errors; redundancy grammatical problems). This tool is used when the development of ontologies has finished. Figure 4-1 presents possible problems that can exist when taxonomic knowledge is modeled.

As work on ontologies migrates from academic institutions into commercial environments, ontologies have to conform to stronger requirements (correctness, consistency, completeness, conciseness, etc.). For this reason, ontology tools (like ODEval) are needed to prevent possible anomalies in ontologies, both in the research area and in the industrial area, in order to provide reliable ontology-based systems [23].

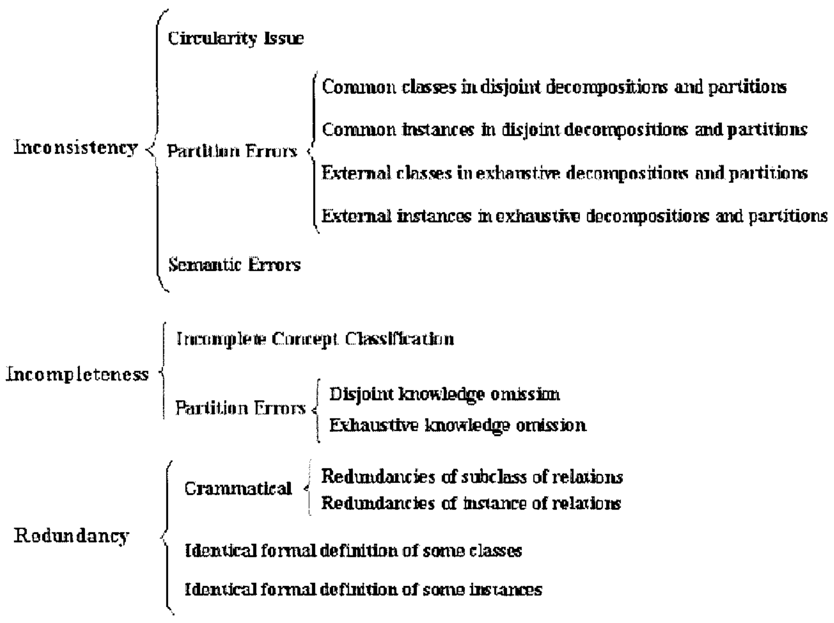

Inconsistency
Circularity Issue
Partition Errors
Common classes in disjoint decompositions and partitions
Common instances in disjoint decompositions and partitions
External classes in exhaustive decompositions and partitions
External instances in exhaustive decompositions and partitions
Semantic Errors
Incompleteness
Incomplete Concept Classification
Partition Errors
Disjoint knowledge omission
Exhaustive knowledge omission
Redundancy
Grammatical
Redundancies of subclass of relations
Redundancies of instance of relations
Identical formal definition of some classes
Identical formal definition of some instances

Figure 4-1 Potential problems that might appear in taxonomies

copied from [14]

ODEval uses a set of algorithms based on graph theory [15]. An ontology concept taxonomy is considered as a directed graph *G (V, A)*, where *V* is a set of vertex and *A* is a set of directed arcs. For each language and each type of problem, the elements in the sets *V* and *A* are different. Table 4-1 shows potential problems in RDF(S), DAML+OIL, and OWL.

ODEval is used to detect possible anomalies from a knowledge representation point of view. Hence, this tool helps ontology developers in designing ontologies and helps ontology engineers to reuse ontologies.

	RDF(S)	DAML+OIL	OWL
Circularity problems	Looking for cycles in the graph *G (V, A)*	Looking for cycles (Mixed cycles and Equivalence cycles) in the graph *G (V, A)*	Looking for cycles in the graph *G (V, A)*
Partition errors	No existence (because no definition allowed of disjoint or exhaustive knowledge with any of the primitives of the language)	Disjoint groups: an error occurs in a disjoint decomposition or a partition, formed by the classes $\{class\_P_1, class\_P_2, \ldots, class\_P_n\}$, if there are common elements in two or more branches of the partition	Detecting errors in disjoint groups: an error occurs in a disjoint decomposition or a partition, formed by the classes $\{class\_P_1, class\_P_2, \ldots, class\_P_n\}$, if there are common elements in two or more branches of the partition
		Exhaustive groups: if an element is only reachable from the base class (or its equivalents) and it is not reachable from the classes of the decomposition, then an error occurs	
Redundancy problems	For each class *class_A* in *V* and each arc r_i in *A* whose origin is *class_A*, taking r_i out of *A* and check if this change affects the set of elements reachable from the *class_A*. If no change, this means at least one of the r_i is dispensable. In this way, at least one problem can be found		

Table 4-1 Means of problem detection in RDF(S), DAML+OIL, and OWL

4.2 OntoManager

OntoManager [21] helps determine the truthfulness of an ontology with respect to its problem domain [23].

In an ontology-based information portal, ontologies often support semantic annotation and conceptual navigation. However, for each business environment, changes should be applied to the ontology to better meet users' needs. If the underlying ontology is not up-to-date or the annotation of knowledge resources is inconsistent, redundant or incomplete, then the reliability, accuracy and effectiveness of the system decrease significantly [27]. In order to avoid these problems, ontology-based applications have to be supported by a mechanism for discovering these changes, analyzing and resolving them in a consistent way [41].

To achieve these goals, OntoManager is designed to detect and resolve mismatches in a consistent and verifiable manner. The state-of-the-art is evaluated in ontology management tools and the best-in-class techniques and methods are selected. In order to facilitate the management of ontologies, a workbench is proposed to integrate these tools and enable interoperability between them (as Figure 4-2). This tool is used to evaluate ontologies after their release.

In this way, OntoManager can find the "weak places" in the ontology according to users' requirements, and try to modify it. It relies on the analysis of usage data by tracking users' interactions with the application in a log file and it is possible to collect useful information that can be used to assess users' main interests. However, it has the usage limitation for evaluating ontologies in general as it is best applied in domains where it occurs in ontology-based application. Therefore, OntoManager might be used as an additional ontology analysis in help with an ontology evaluation.

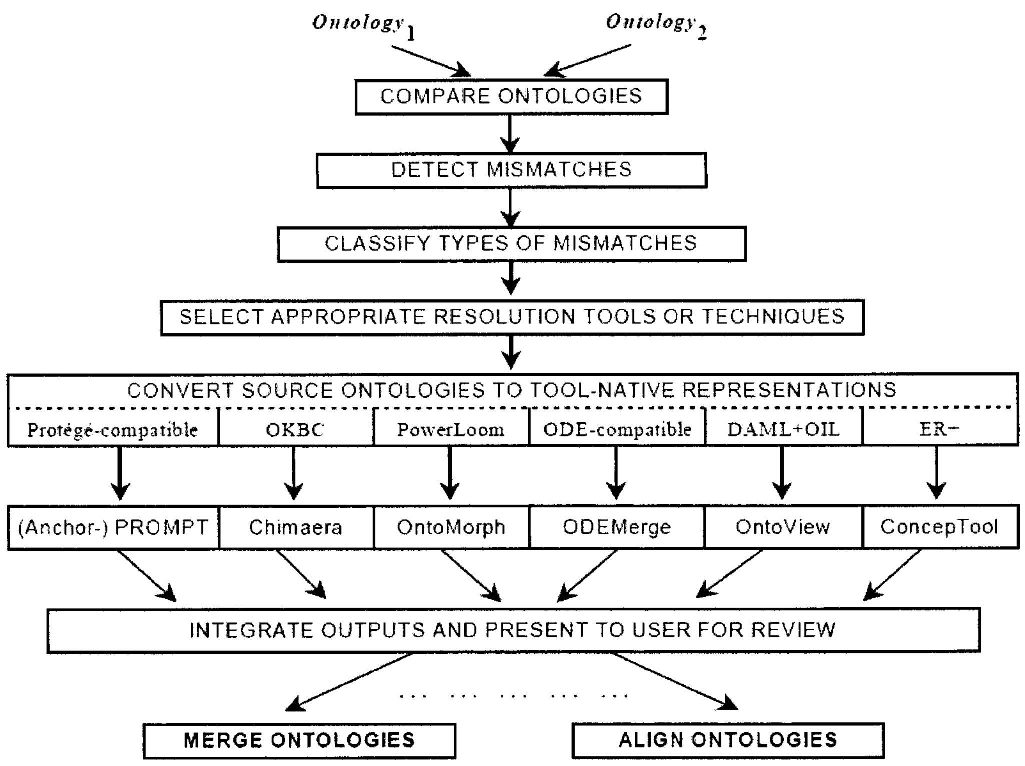


Figure 4-2 A schematic diagram of the OntoManager

copied from [21]

4.3 CleanONTO

For the Semantic Web, having consistent ontologies are very important. The system of CleanONTO [38] describes concepts with definitions, which are paths from the concept to the root of the ontology. In the current study, these definitions have been extracted from WordNet 2.1 [29, 11] rather than a domain specific corpus. To explain the system more clearly, we first describe the OntoClean [17] methodology and the different implementations available to evaluate taxonomic relationships in ontologies.

4.3.1 OntoClean Methodology

The OntoClean approach focuses on the concepts and roles (properties) involved in taxonomic relationships. It is an '*IS-A*' relation, for example if P and Q are unary predicate symbols, and I is an interpretation function, then P *IS-A* Q iff $I(P) \subseteq I(Q)$.

As part of the methodology, a set of meta-properties are introduced drawn from philosophy—unity, identity, essence and dependence, to characterize relevant aspects of the intended interpretation of the roles, concepts, and relations that make up ontologies. Moreover, these meta-properties dictate several constraints on the ontology taxonomy structure, used to evaluate ontologies.

- Unity: to recognize all the parts forming an instance and to determine the intended meaning of properties in ontologies. +U—carry unity; -U—carry no unity; ~U—carry anti-unity;
- Identity: based on intuition abound how users interact with individual entities and distinguish an individual from others in a certain concept by analyzing essential characteristic properties. +I—carry an identity criterion; -I—do not carry; +O—supply 'own' but not inherited from subsuming properties.
- Essence: the properties with the rigid definition in order to let the individual remain the same when displaying different properties at different times. +R—rigid property; -R—non-rigid property; ~R—anti-rigid property
- Dependence: involving every instance dependent on a concept

Figure 4-3 (a) and (b) display the 'unclean ontology' and 'cleaned taxonomy' by OntoClean respectively as follows.

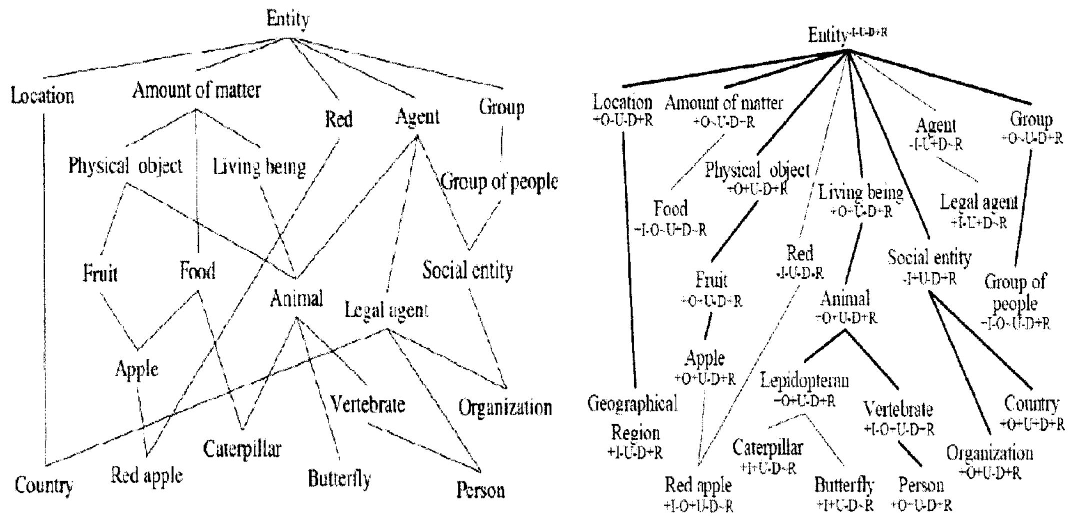


Figure 4-3 (a) unclean ontology (b) cleaned taxonomy by OntoClean

copied from [38]

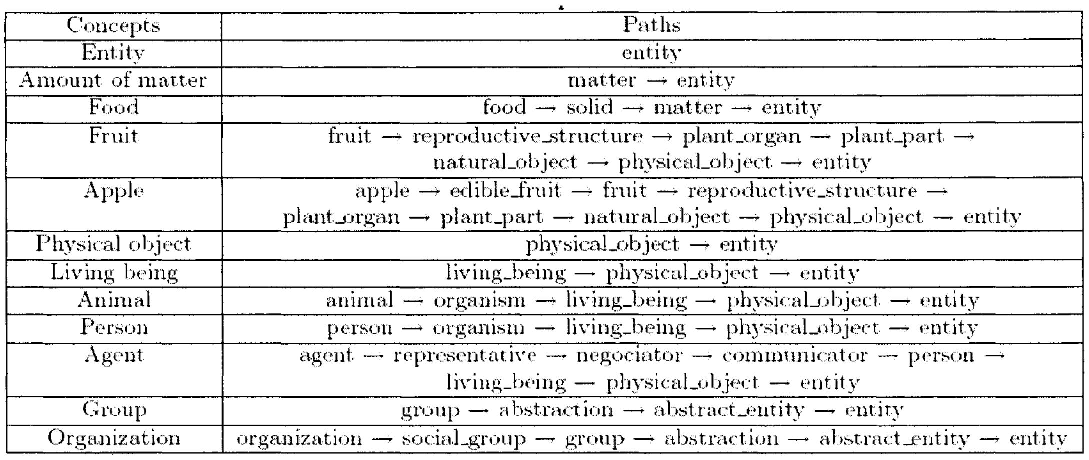
However, different people may describe the same concept with different sets of meta-properties. In this case, it seems that the meta-properties schema is not a good utility for domain experts and engineers.

4.3.2 CleanONTO Methodology

The approach has mainly three distinct phases as follows.

- Acquire descriptions for each concept

  Each of the concepts in the 'unclean' ontology is looked up in the online version of WordNet [42] by an investigator reporting each of the paths.



Concepts	Paths
Entity	entity
Amount of matter	matter → entity
Food	food → solid → matter → entity
Fruit	fruit → reproductive_structure → plant_organ → plant_part → natural_object → physical_object → entity
Apple	apple → edible_fruit → fruit → reproductive_structure → plant_organ → plant_part → natural_object → physical_object → entity
Physical object	physical_object → entity
Living being	living_being → physical_object → entity
Animal	animal → organism → living_being → physical_object → entity
Person	person → organism → living_being → physical_object → entity
Agent	agent → representative → negociator → communicator → person → living_being → physical_object → entity
Group	group → abstraction → abstract_entity → entity
Organization	organization → social_group → group → abstraction → abstract_entity → entity

Table 4-2 Paths for concepts found in WordNet

copied from [38]

- Break inappropriate links

In this system, the parent-child relationship is defined as: *A* concept *B* subsumes a concept *A*, iff the whole path of *B* is found in the path of *A*, where concept *A* is called the child, and *B* is called the parent [38].

When links break, the circumstances occur as Figure 4-2 (e) and (f) above. There are some kinds of 'circles' in the taxonomy, which means a parent can be inherited by a child and child can be inherited by a parent at the same time. In the CleanONTO system, when the path of a parent is not in the path of child, the link is inconsistent. These inconsistent 'subtrees' will be removed from the ontology by the algorithm set in CleanONTO and the basic ontology (Tree-0) forms as Figure 4-4 (the original unclean ontology is shown in Figure 4-3 (a)).

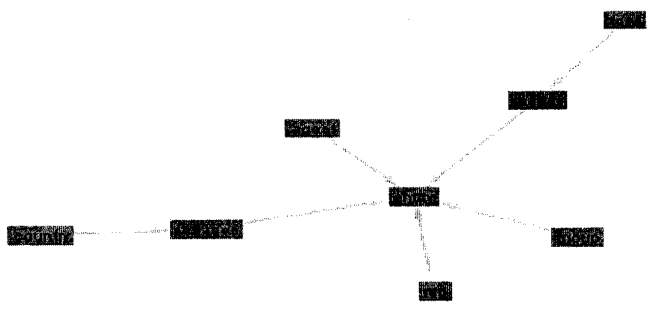


Figure 4-4 Resulting Tree-0 in CleanONTO

copied from [38]

- Create a consistent tree

  After removal all inconsistent 'subtrees', the next step is to try to place these temporarily removed inconsistent elements back to Tree-0 in a consistent way.

For each ready-to-add concept, it may have several direct parents according to Table 4-2. It is added to the Tree-0 as the child of the 'least-root' parent. For example, we try to place the 'subtree' of the concept '*entity*' back to Tree-0. In Tree-0 in Figure 4-4 we already have '*entity*→*group*'. i) the concept '*person*' has the path of '*person*→*organism* →*living being* →*physical object* →*entity*' (in Table 4-2); only the '*entity*' is included in the Tree-0; so we add '*person*' as the direct child to '*entity*' shown in Figure 4-5 (1); ii) add '*organization*', with a path of '*organization*→*social_group*→*group*→*abstraction*→*abstract entity*→*entity*' (in Table 4-2), in this case, it has two parents—'*group*' and '*entity*', because '*group*' is the child of '*entity*', we add '*organization*' as the direct child of '*group*' shown in Figure 4-5 (2); iii) add '*living being*' with the path of '*living* *being→physical object→entity*', its parent—'*entity*' appears after the step ii), but the '*person*' includes the its path as well, which means '*person*' is a child of '*living being*', so the Tree will be arranged as Figure 4-5 (3).

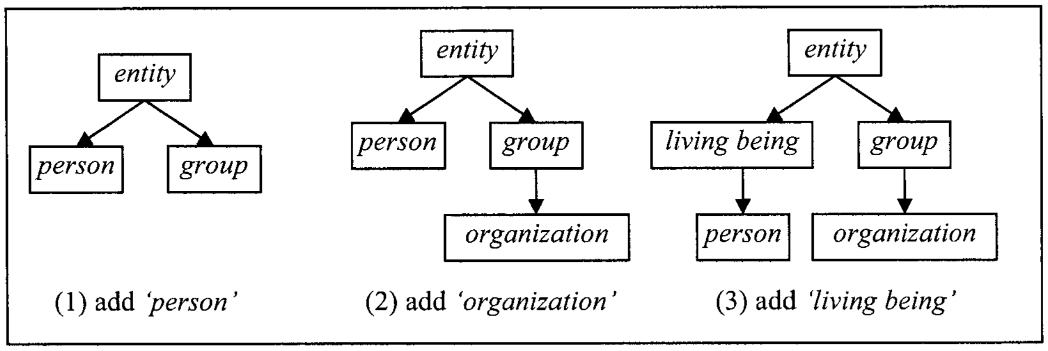


Figure 4-5 Example steps to recover consistent tree

For the above example, the taxonomy tree is recovered and arranged and it is illustrated as a cleaned ontology in Figure 4-6. The GUI is shown in Figure 4-7.

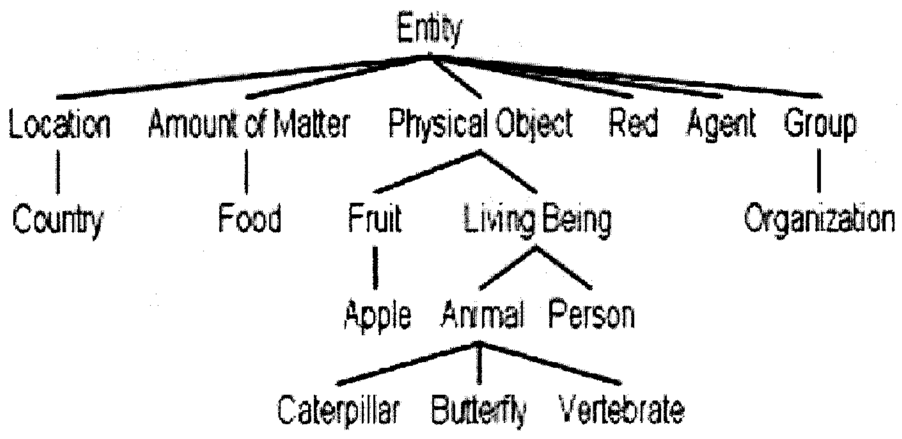
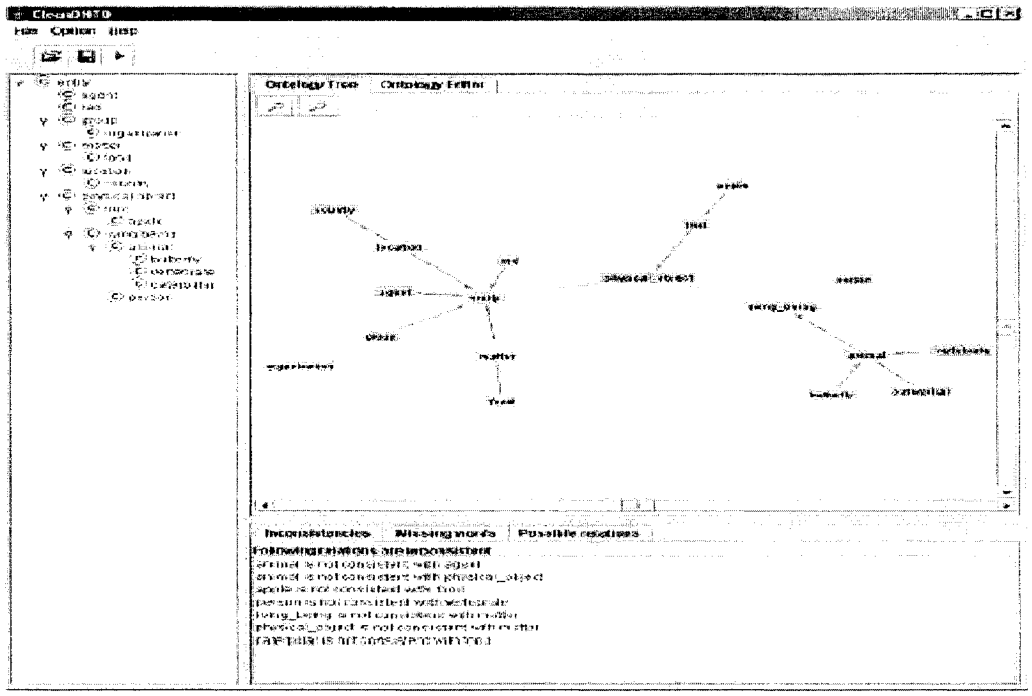


Figure 4-6 Cleaned ontology as produced by CleanONTO

copied from [38]



Following relations are inconsistent
animal is not consistent with agent
animal is not consistent with physical_object
apple is not consistent with food
person is not consistent with vertebrate
living_being is not consistent with matter
physical_object is not consistent with matter
caterpillar is not consistent with food

Figure 4-7 GUI of CleanONTO

copied from [38]

4.3.3 Comparison of OntoClean and CleanONTO

Although OntoClean and CleanONTO provide methodologies to change an inconsistent ontology into a consistent one, they are different in some ways. Compared with OntoClean, CleanONTO is likely easier to use for ontology engineers with a GUI to create a consistent taxonomy. Table 4-3 describes other distinctions between these two methodologies.

OntoClean	CleanONTO
Based on meta-properties; Sometimes add a new concept to the taxonomy to show the different levels.	The concepts in the taxonomy not in WordNet are not in the revised tree.

Table 4-3 Main difference between OntoClean and CleanONTO

4.4 CORE

CORE (a tool for Collaborative Ontology Reuse and Evaluation) provides automatic similarity measures for comparing a certain problem or Golden Standard to a set of available ontologies, retrieving not only those most similar to the domain described by the Golden Standard, but the ones rated best by prior ontology users, according to selected criteria [12]. The tool retrieves a ranked list of ontologies for each criterion, using rank fusion techniques [2]. The architecture is shown in Figure 4-8. Figure 4-9 shows the GUI of CORE.

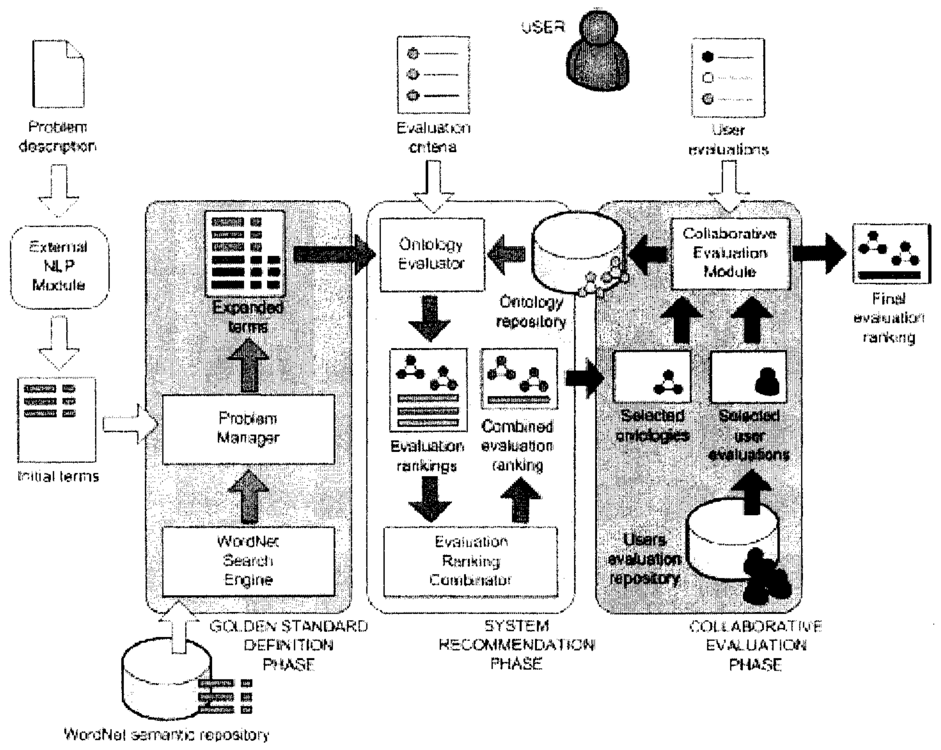
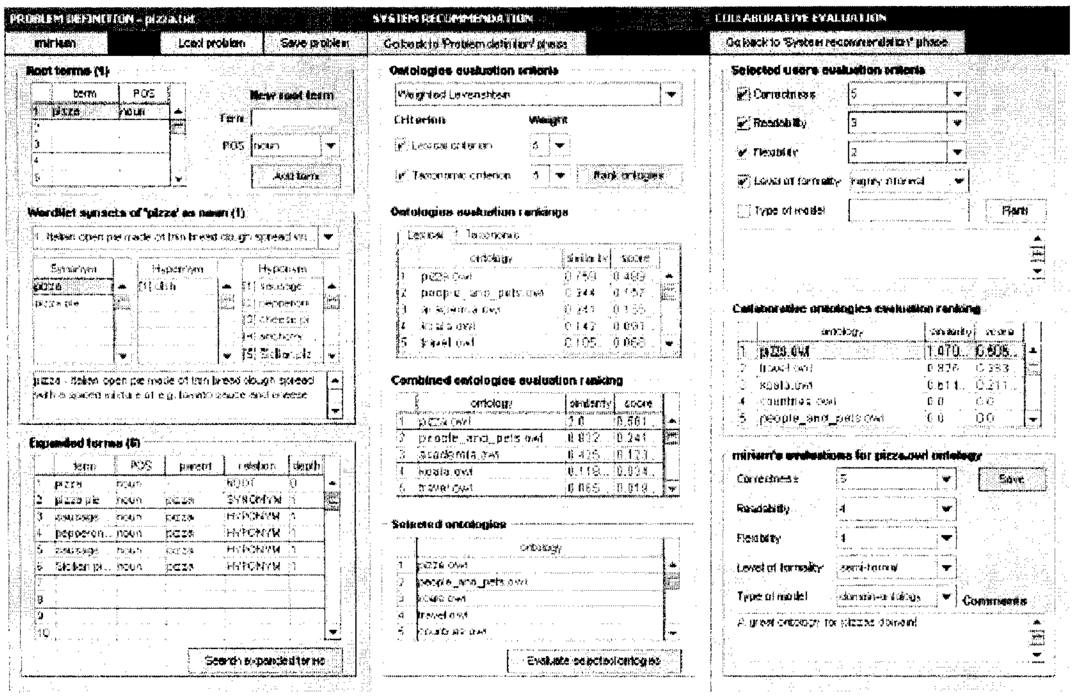

USER
Problem description
External NLP Module
Initial terms
Expanded terms
Problem Manager
WordNet Search Engine
WordNet semantic repository
GOLDEN STANDARD DEFINITION PHASE
Evaluation criteria
Ontology Evaluator
Ontology repository
Evaluation rankings
Combined evaluation ranking
Evaluation Ranking Combinator
SYSTEM RECOMMENDATION PHASE
User evaluations
Collaborative Evaluation Module
Selected ontologies
Selected user evaluations
Users evaluation repository
COLLABORATIVE EVALUATION PHASE
Final evaluation ranking

Figure 4-8 CORE architecture

copied from [12]


PROBLEM DEFINITION - pizza.txt

Root terms (1)

	term	POS
1	pizza	noun
2		
3		
4		
5		

New root term: Term: POS: noun

WordNet synsets of 'pizza' as noun (1)

1. Italian open pie made of thin bread dough spread wi...

Synonym	Hypernym	Hyponym
pizza	[1] dish	[1] sausage
pizza pie		[2] pepperoni
		[3] cheese pi
		[4] anchovy
		[5] Sicilian piz

pizza - Italian open pie made of thin bread dough spread with a spiced mixture of e.g. tomato sauce and cheese

Expanded terms (6)

	term	POS	parent	relation	depth
1	pizza	noun		ROOT	0
2	pizza pie	noun	pizza	SYNONYM	1
3	sausage	noun	pizza	HYPONYM	1
4	pepperon...	noun	pizza	HYPONYM	1
5	sausage	noun	pizza	HYPONYM	1
6	Sicilian pi...	noun	pizza	HYPONYM	1
7					
8					
9					
10					

SYSTEM RECOMMENDATION

Ontologies evaluation criteria: Weighted Levenshtein

Criterion	Weight
✓ Lexical criterion	5
✓ Taxonomic criterion	5

Ontologies evaluation rankings (Lexical | Taxonomic)

	ontology	similarity	score
1	pizza.owl	0.750	0.469
2	people_and_pets.owl	0.344	0.157
3	academia.owl	0.241	0.135
4	koala.owl	0.142	0.091
5	travel.owl	0.105	0.068

Combined ontologies evaluation ranking

	ontology	similarity	score
1	pizza.owl	2.0	0.561
2	people_and_pets.owl	0.832	0.241
3	academia.owl	0.425	0.123
4	koala.owl	0.118...	0.034...
5	travel.owl	0.065	0.019

Selected ontologies

	ontology
1	pizza.owl
2	people_and_pets.owl
3	koala.owl
4	travel.owl
5	countries.owl

COLLABORATIVE EVALUATION

Selected users evaluation criteria

✓ Correctness	5
✓ Readability	3
✓ Flexibility	2
✓ Level of formality	highly informal
Type of model	

Collaborative ontologies evaluation ranking

	ontology	similarity	score
1	pizza.owl	1.070...	0.605...
2	travel.owl	0.826	0.353
3	koala.owl	0.611...	0.211...
4	countries.owl	0.0	0.0
5	people_and_pets.owl	0.0	0.0

miriam's evaluations for pizza.owl ontology

Correctness	5
Readability	4
Flexibility	4
Level of formality	semi-formal
Type of model	domain-ontology

Comments: A great ontology for pizzas domain!

miriam
Load problem
Save problem
Add term
Search expanded terms
Go back to 'Problem definition' phase
Rank ontologies
Evaluate selected ontologies
Go back to 'System recommendation' phase
Rank
Save

Figure 4-9 GUI of CORE

copied from [12]

Three different modules are distinguished as follows:

- Golden standard definition phase: it allows users to expand root terms and some of the relations.
- System recommendation phase: it allows users to select a set of ontology evaluation techniques used by the system to recover the ontologies closest to the given Golden Standard.
- Collaborative evaluation phase: it re-ranks the list of recovered ontologies by considering feedback and users' evaluation.

CORE considers two lexical and taxonomic content ontology evaluation levels, for each several measures have been developed and tested.

- Lexical evaluation: compared the lexicon entries or words of the Golden Standard and a certain ontology are compared, the similarity between the two ontologies is described. A new lexical evaluation measure is based on [28].
- The taxonomic evaluation: it assesses the degree of overlapping between the hierarchical structure of the ontology, defined by the '*IS-A*' relation and the Golden Standard structure, defined by the derivations of terms to complete the domain representation.

4.5 Ontology evaluation methodology for ontology evolution

This methodology considers the core questions as follows: how to define what a good ontology for a particular context is, and how to perform ontology evolution to actually obtain better ontologies in an automated manner. A central role in this approach is played by the ontology evaluation function, which guides the discovery of changes that lead to an improved ontology [20].

Two important forms of context: the usage-context and the domain context are selected, which are relevant in many ontology-based applications. Furthermore, the method provides the flexibility to essentially define arbitrary ontology evaluation functions [20] for a variety of contexts and is open to embed new methods for change discovery leading to improved ontologies.

The ontology evaluation is a basis to define what a good ontology is. It takes evaluation considerations into account '*during*' and '*before*' evolution. It fulfills two conditions: i) to be able to tell which is 'better' for two given ontologies; ii) to offer a way to come up with sensible ontology change operations in order to evolve the ontology automatically.

In this case, first of all, it lays the foundations to capture an ontology together with its context, where it uses a formal ontology model based on OWL [24]; second, it formalizes how to capture the contextual information; third, it formalizes the notion of an ontology evaluation function; fourth, it presents the discover-change problem for ontology evolution according to the optimization of the ontology evaluation function.

Figure 4-10 illustrates the logic architecture of the methodology. The key advantage is that users are allowed to define for themselves what a 'good' ontology is and provide support for making 'better' ontologies automatically.

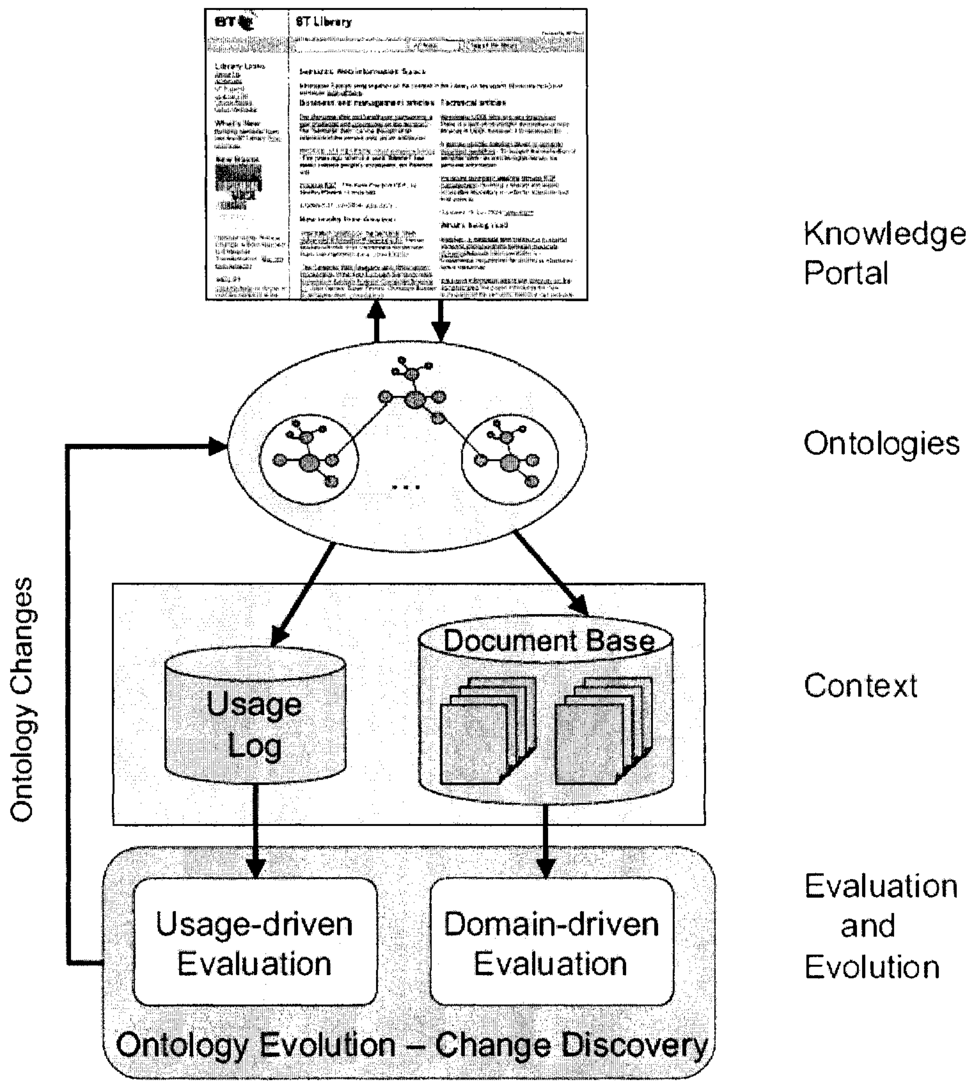


Figure 4-10 Logic architecture

copied from [20]

5. OntoKBEval (A Support Tool for OWL Ontology evaluation)

This chapter focuses on the OntoKBEval system design and implementation details. We describe our design ideas, the system architecture, and the main methods and functions.

5.1 Design ideas

As mentioned in Chapter 1, one can get thousands of ontologies through the Internet; for example, one can find hundreds of ontologies about bio-informatics, which may have large TBoxes or ABoxes with hundreds of thousands of elements. Users should have an idea of finding the 'best' one to meet their need from their point of view. Ontology evaluation can provide possible solution. We use RacerPro to translate OWL ontologies into TBox and ABox information.

We present the evaluation results in the following three ways:

- Overview diagram: instead of showing all details in a hierarchical structure, we consider the overview structure. We think it is a better way to present structures for large ontologies by a Christmas tree, coordinate and clustered circle figures. For example, if one wants to have the hierarchy of the concepts in traditional ways, a standard diagram of a structured concept-tree could be generated. This may make sense when the ontology does not have a large number of concepts. However, to imagine, when there are thousands of concepts, the tree figure will be very complex and hard to understand, and the graph layout could be difficult. Thus, we decided to only count the number of concepts at each level and represent them with a line whose length is proportional to the number of concepts and to the scale of the 'longest' line. So users can easily get an overall impression of the concepts' distribution.

- Details browser: besides an overview structure, it is also important to give detailed information about the DL KB, for example, listing all the concept names in the ontology, furthermore, users can search for all concept names containing certain keywords.
- Statistics: we compute the number of classified elements in a TBox or an ABox to have a first general impression of the ontology. For example, we provide the total number of concepts, individuals, and roles in an ontology in order to give users an idea about how big the ontology is; the number of levels for the concept hierarchy shows the depth of the concept tree. We use possible classification solutions to present concepts, roles, and individuals and their interrelationships in certain efficient ways.

5.2 System architecture

The OntoKBEval tool consists of two main components:

- Reasoner
  - RacerPro 1.9 (access by using JRacer)
- User interface
  - Users choose ontology files and send queries to RacerPro

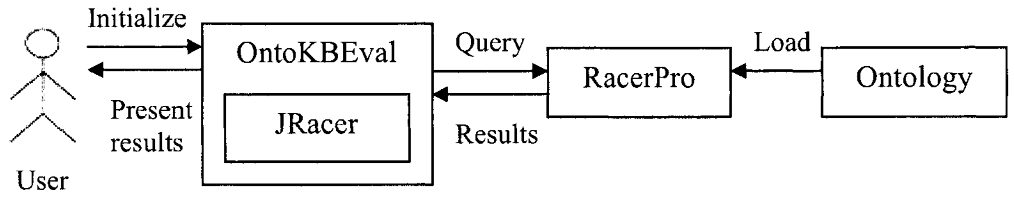


Figure 5-1 System architecture

5.3 System overview

The OntoKBEval system helps users to determine whether a loaded ontology is the right one to meet their needs. In most circumstances, concept names are of primary interest to users. Users want to know if the ontology file contains certain concepts they are interested in.

For these purposes, we offer to evaluate ontology files mainly in three ways: quick-view ontology evaluation, general ontology evaluation, and multi-file ontology evaluation. We describe them in later sections respectively. We use the ontology file '*people-pets.owl*' as an example in the following. We implemented our tool in Java and express ontology information using Racer commands.

5.4 System main window

The main window of our tool is shown in Figure 5-2. After loading an ontology file, users can choose one of the three options mentioned above. We consider the first step of a general evaluation the most important one, which mainly presents basic TBox information to give users a first impression about the ontology; so the first results of the general ontology evaluation are shown in the main window, such as the number of concepts, the number of levels of the concept hierarchy, and so on.

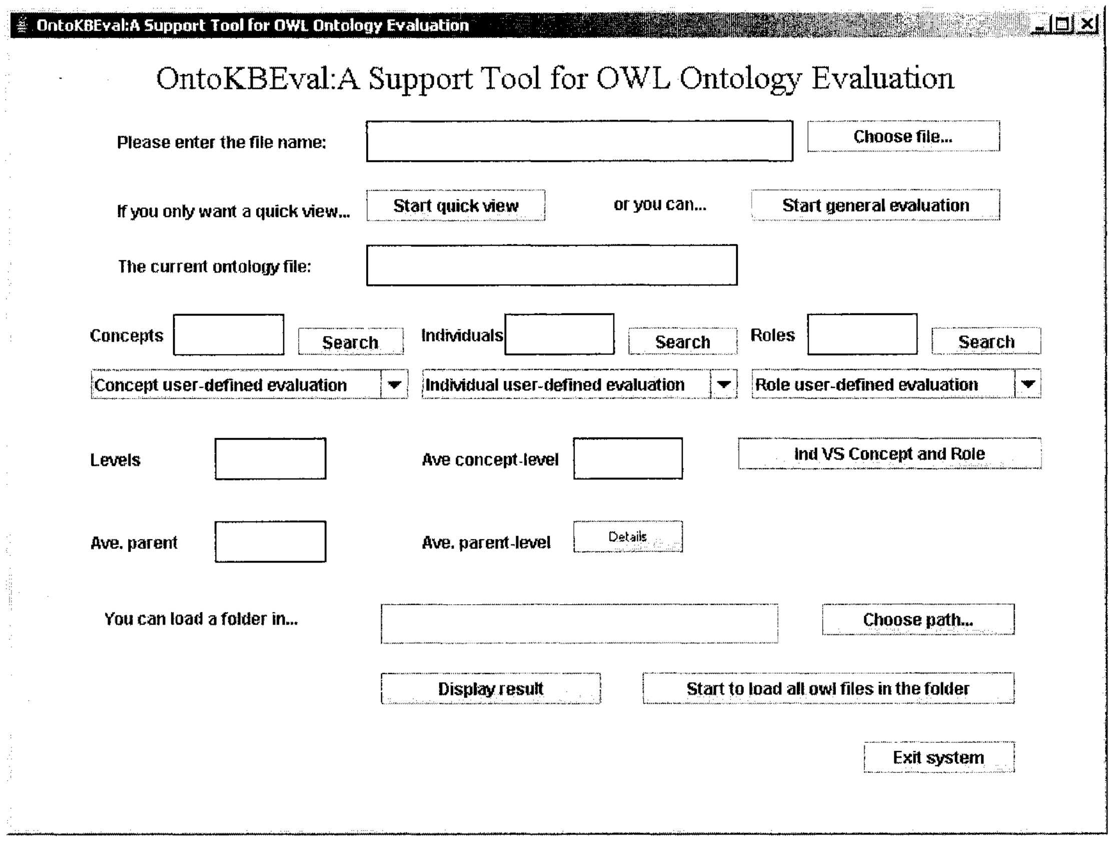

OntoKBEval:A Support Tool for OWL Ontology Evaluation
OntoKBEval:A Support Tool for OWL Ontology Evaluation
Please enter the file name:
Choose file...
If you only want a quick view...
Start quick view
or you can...
Start general evaluation
The current ontology file:
Concepts
Search
Individuals
Search
Roles
Search
Concept user-defined evaluation
Individual user-defined evaluation
Role user-defined evaluation
Levels
Ave concept-level
Ind VS Concept and Role
Ave. parent
Ave. parent-level
Details
You can load a folder in...
Choose path...
Display result
Start to load all owl files in the folder
Exit system

Figure 5-2 OntoKBEval main window

In order to start any kind of evaluation, it is required for users to first choose and load an ontology file. The choose-file page is shown in Figure 5-3.

After clicking '*Open*', we finish the ontology file selection; but the ontology file will not be immediately loaded using RacerPro. The loading action can only be triggered by starting any kind of evaluation. The separation between the 'choose-file' and 'load-file' functions can help avoid loading a 'wrong' ontology file by mistake.

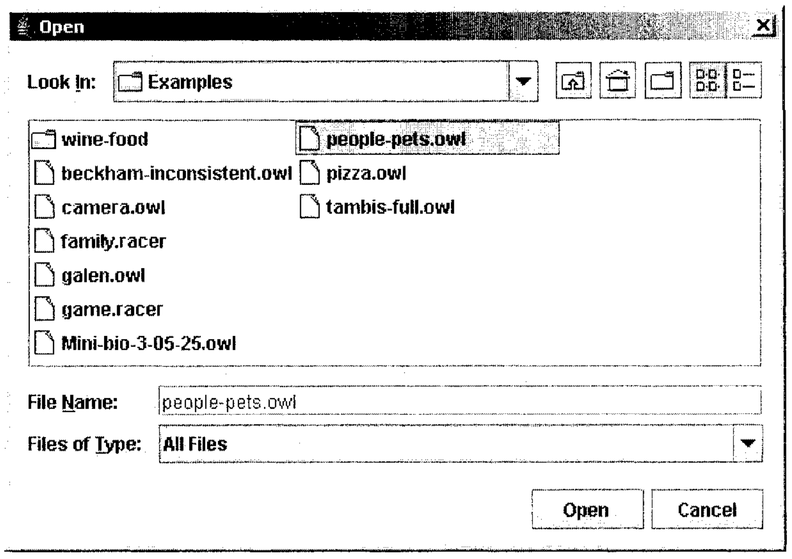


Figure 5-3 Ontology file loading window

5.5 Assumptions

Before we start to explain the functions, we mention some assumptions that we may apply when dealing with query results from RacerPro.

- Concept parent-children relationships: let us consider the conditions as shown in Figure 5-4(a), (b), (c), (d). In the case of (d), for example, concepts, '*C1*' and '*C2*' are in the 1st level, '*C3*' is in the 2nd level, and '*C4*' is in the 3rd level, though '*C4*' is a direct child of '*C2*'. As to the conditions of (e) and (f), we do not have to consider them because in the case of (e), we get *C1*$\rightarrow$*C2*, *C1*$\rightarrow$*C3*, and *C2*$\rightarrow$*C3*, and integrate them into *C1*$\rightarrow$*C2*$\rightarrow$*C3* as the relationship in case (a); in the case of (f), we get *C1*$\rightarrow$*C2*, *C2*$\rightarrow$*C3*, and *C3*$\rightarrow$*C1*, then we can conclude *C1*, *C2*, and *C3* are equivalent and RacerPro would have discovered this already. We assume this kind of error does not exist in our hierarchy.

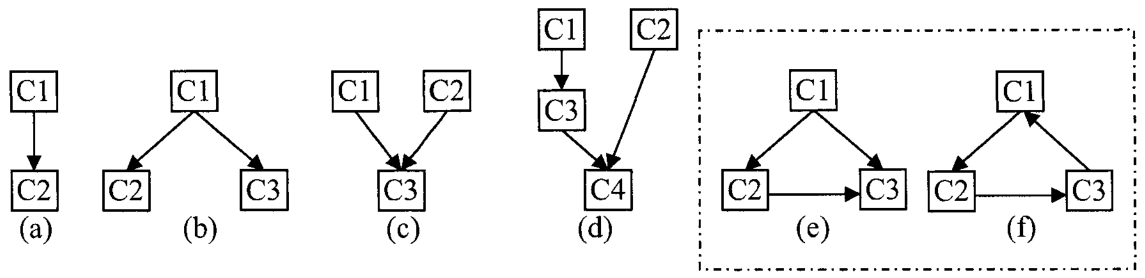


Figure 5-4 Concept parent-children relationship

- To save the computation time in the 'general ontology evaluation' part, we only compute the concept part in advance; for roles, individuals and assertions, we only compute them on demand.
- We assume that every ontology file has both a TBox and an ABox.
- The concept, individual, and role names are case sensitive.
- When the tool is started, the RacerPro should be started as well.

5.6 Quick-view ontology evaluation

This option is motivated by the idea of being able to look through ontologies faster. It gives users a quick overview about the use of concept names in ontology files and tries to find a 'qualified' ontology file. Usually, concept names reflect to which domain and branch of the domain the ontology belongs to. Users can search concepts by entering keywords. This function does not show concept hierarchies and can operate much faster especially for large ontologies. Users can search ontology files one by one very quickly until they find a desired ontology.

If there are matching concepts, they will be shown in a list-box. In this case, the user can look through the results to see if they meet the requirements for our tasks. If the ontology file is appropriate, users can choose to do the general ontology evaluation for retrieving more detailed information. For example, if '*cat*' was entered as a search phrase, we get the results as shown in Figure 5-5. The number of results is shown next to the bottom-left corner of the list-box on the left. The results include all concept names, with the keyword '*cat*'.

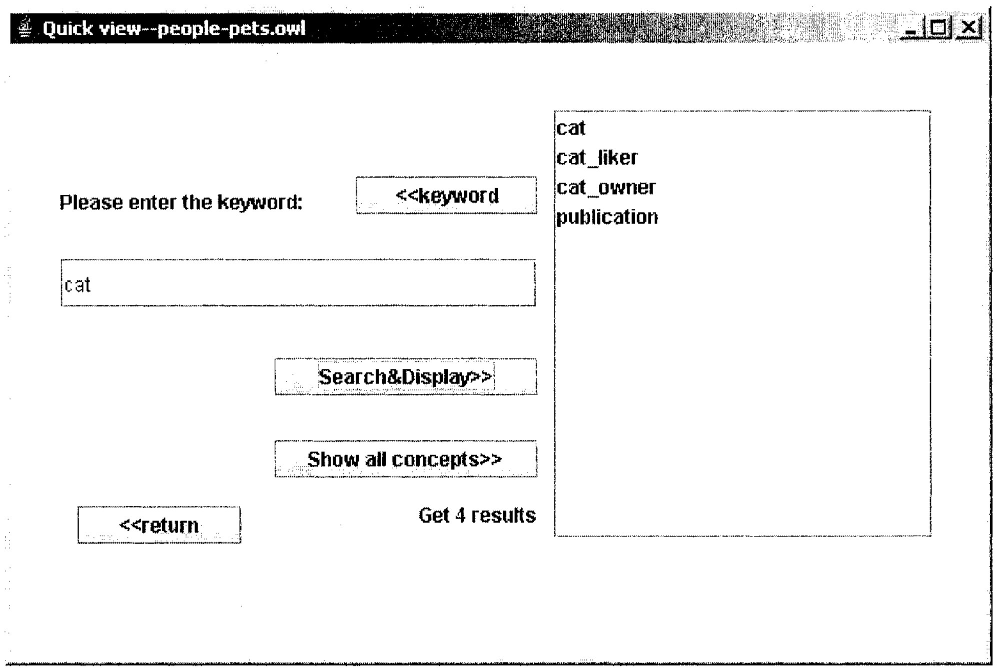


Figure 5-5 Quick-view—main window

To facilitate the search operations, users can list all concept names in the list-box by clicking '*show all concepts*' button, in which way users can directly browse the list and find all concepts in alphabetic order (shown in Figure 5-6). For small ontologies with a few hundreds of concepts, this is a quick way to get an overview; however, for larger ontologies, we suggest to use keyword search to find related concepts. Users can only choose one of the concept names as keyword, which will be copied automatically in the keyword text field by clicking on the '*keyword*' button.

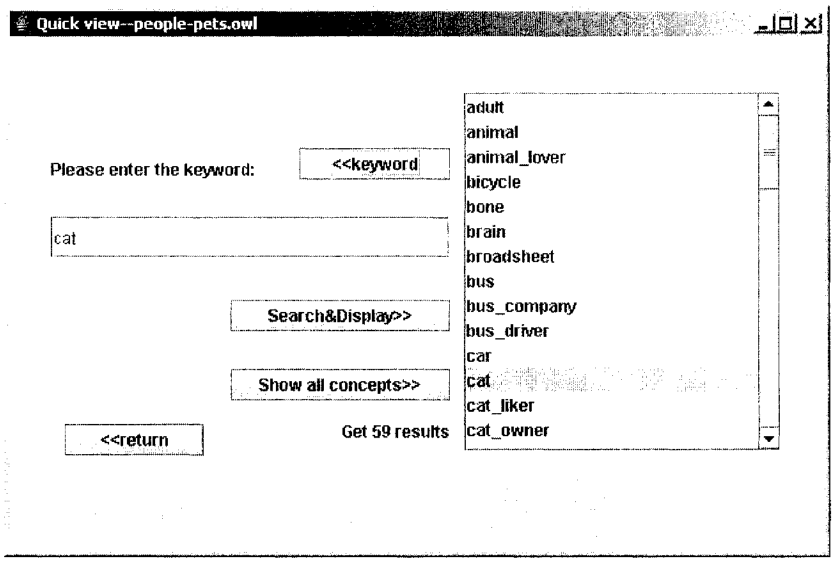
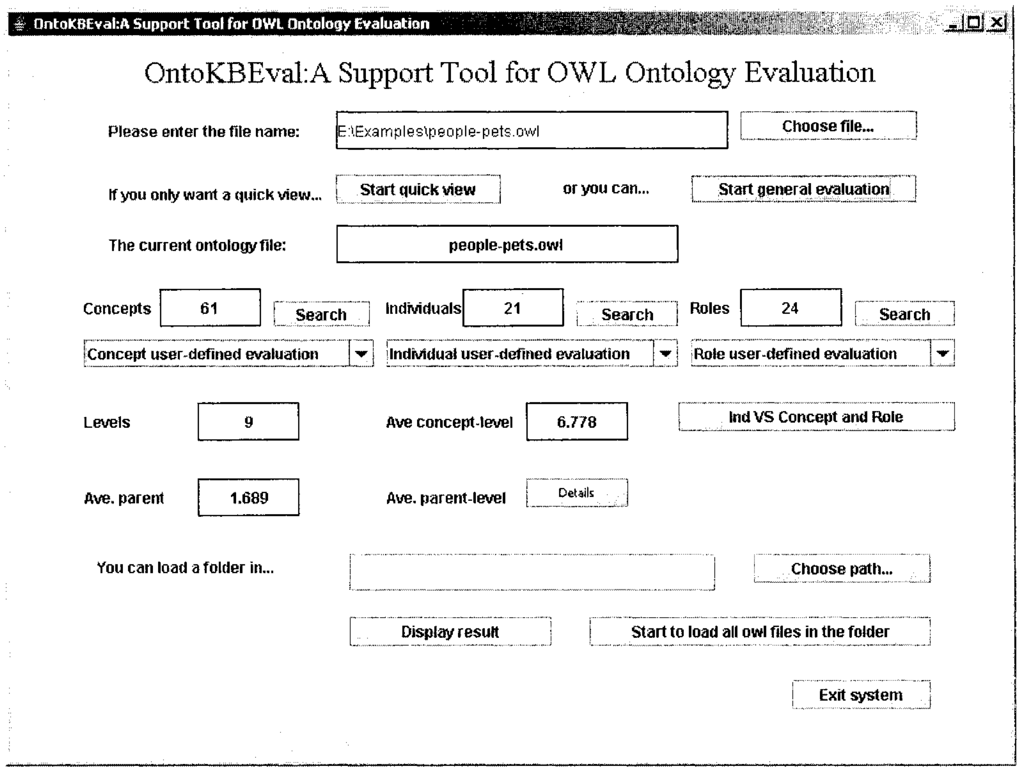


Figure 5-6 Quick-view—list all concepts

5.7 General ontology evaluation

This part is the most vital part in our ontology evaluation system. It provides the whole 'description' of indicated ontologies. Both the TBox and ABox are considered.

Usually in the first step, we evaluate the TBox and then process the ABox, including concept and role assertions. For the TBox, we provide mainly the number of concepts and roles, and hierarchical diagrams For the ABox, we classify assertions as tuples and clusters according to relationships between related elements. The clusters are presented by circles. In both parts, browse operations can be used to retrieve more detailed information about concepts, roles, individuals, and assertions. The results of a general evaluation are shown in Figure 5-7.


OntoKBEval:A Support Tool for OWL Ontology Evaluation

OntoKBEval:A Support Tool for OWL Ontology Evaluation

Please enter the file name: E:\Examples\people-pets.owl

The current ontology file: people-pets.owl

Concepts 61 | Individuals 21 | Roles 24

Concept user-defined evaluation | Individual user-defined evaluation | Role user-defined evaluation

Levels 9 | Ave concept-level 6.778

Ave. parent 1.689 | Ave. parent-level

You can load a folder in...

Figure 5-7 General—results of the first step (mainly about TBox)

5.7.1 TBox evaluation

In this section, we present the functions about concept and role evaluation. Concepts are fundamental parts, which describe individuals with common characteristics. Roles define relationships among individuals.

We describe relationships between two concepts as follows:

- The parent-child relationship: when one concept is subsumed by another one.

- The ancestor-descendant relationship: when one concept is subsumed directly or indirectly by another one.
- Concept '*top*' is the ancestor of all other concepts; it only has children and descendants but no parents.
- Concept '*bottom*' is the descendant of all other concepts; it only has parents and ancestors but no children.

In our system, we count the number of levels from the '*top*', which is at level '0'; the children of '*top*' are in the 1st level, and so on; until the concept is '*bottom*', which is in the last level of the hierarchical structure.

Roles are used to create connections between individuals, the instances of concepts. A role may be transitive, symmetric, or have inverse, or equivalent roles.

Roles are also organized in a hierarchical structure similar to that of concepts. A role can inherit from other roles and can be inherited by other roles. However, unlike in a concept hierarchy, the role hierarchy does not have the '*top*' and '*bottom*' elements as in a concept hierarchy. The roles, which do not have any parent, are in the 1st level; the roles, which do not have any child, are in the last level.

Except for the overview evaluation, we also provide functions to provide a more detailed exploration of TBox information. We offer the overall figures for the results from the detailed search. With the integration of overall and detailed information, users can have an idea not only about what the TBox looks like, but what are components of this structure.

5.7.1.1 Concept Xmas-tree figure

Figure 5-8 displays the concept hierarchy of '*people-pets.owl*'. For each level, we mark the number of concepts at that level. Users can get an idea about the concepts distribution in the whole hierarchy of the ontology. We know that in the '0' level, we only have '*top*'; in the 8th level, we only have '*bottom*'.

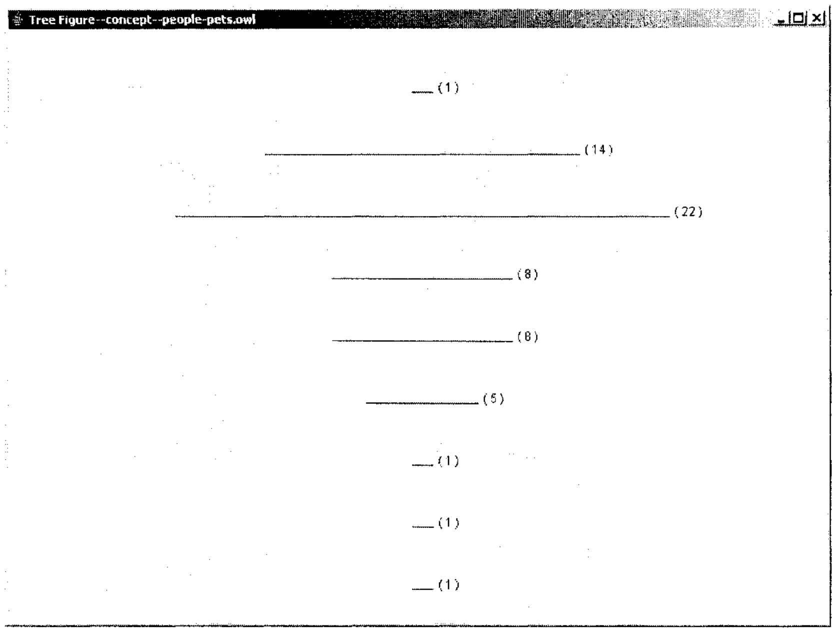


Figure 5-8 General—concept Xmas-tree figure

In contrast to other ontology browsers, we do not provide detailed information in this figure, for example, the concept names at each level and their parents. However, we can check the concept names using the function '*concept level details*'. We discuss it in Section 5.7.1.9.

To create the Xmas-tree figure, we use the following algorithm:

i) We retrieve parents and children of each concept in an ontology using the Racer commands '(concept-parents *concept_term*)' and '(concept-children *concept_term*)' respectively.

ii) We mark each concept in the format of a level-mark string *(parent_name, current_concept, have_children_or_not, level_no)*, where if a '*current_concept*' has children, we note '*have_children_or_not*' as '#', otherwise '0':

- For '*top*', its level-mark string is *(0, top, #, 0)*, where the 1st '0' means '*top*' has no parents and the 2nd '0' indicates that the level number of '*top*' is '0'.
- For '*bottom*', its level-mark string is *(#, bottom, 0,0)*, where '#' means '*bottom*' always has parents; the 1st '0' means '*bottom*' has no children and the 2nd '0' is just a temporary note and the level number of '*bottom*' will be computed after all level numbers of concepts are settled.
- For concepts other than '*top*' and '*bottom*':
  - ➢ If the parent of a concept is '*top*' and its only child is '*bottom*', its level-mark string is *(top, current_concept, bottom, 1)*, which means this concept is at the 1st level.
  - ➢ If the parent of a concept is '*top*' and it has one or more children other than '*bottom*', its level-mark string is *(top, current_concept, #, 1)*, which means this concept is at the 1st level;

- If the parents of a concept do not include '*top*' and its only child is '*bottom*', for each of these parents, its level-mark strings are *(parent_name, current_concept, bottom, 0)*, where '0' is just a temporary marker and the level number will be computed in the next step.
- If the parents of a concept do not include '*top*' and it has one or more children other than '*bottom*', for each of these parents, its level-mark strings are *(parent_name, current_concept, #, 0)*, where '0' is just a temporary marker and the level number will be computed in the next step

iii) To compute the level number for each concept. As we have marked all the children concepts of '*top*', we mark the level numbers of the children of these concepts on the 1st level, and the children of concepts on the 2nd level, and so on, until all the concepts other than '*bottom*' are marked.

iv) To arrange level numbers for concepts other than '*top*', '*bottom*' and the children of '*top*'. Because case (d) in Figure 5-4 may occur, for each concept, which has more than one parent, we arrange the level number of each concept as the maximum value among its level-mark strings.

v) To mark the level number of '*bottom*'. The level number of '*bottom*' is marked as (the maximum level number of other marked concepts+1).

vi) To count the number of concepts on each level and draw the Xmas-tree figure.

This kind of presentation method may encounter problems when the ontology has too many levels or the number of concepts at a certain level is much larger than that at the others. For example,

i) If there are 100 levels, the lines will be very dense and may be overlapped. The diagram will display a solid black hierarchy drawing and does not properly show how the concepts are distributed.

ii) If we have five levels and the numbers of concepts at each level from '*top*' to '*bottom*' are respectively 1, 1000, 6, 5, 1, we draw the longest line to present the level with 1000 concepts; but compared to 1000, the lines for 1, 5, 6 are almost identical.

iii) If the cases of i and ii occur at the same time, the presentation will become even worse.

To improve the quality of presentation in such cases, we introduce another diagram—the coordinate figure discussed in the next section.

5.7.1.2 Concept coordinate figure

Another hierarchical diagram is the coordinate figure shown in Figure 5-9. The idea comes from mathematical method about diagrams of functions.

This figure is different from that of the Xmas-tree figure in the x-y coordinates; it has the scales for x- (consider the maximum number of concepts among the levels) and y- (the number of levels) coordinates. After testing with more than 30 ontologies with sizes from several KB to several MB, we decided to mark the coordinates (both x and y) using the following parameters:

Domain (x or y)	Scale starts from 1 and marks every interval
[1, 20]	1
[21, 70]	5
[71, 150]	10
[151, 350]	20
[351, 700]	50
[700, 999]	100
>=1000	Computing $\log_{10}(X)$ or $\log_{10}(Y)$ and using the results to mark the scales in a of the linear domain

Table 5-1 General—coordinate figure—scale selection

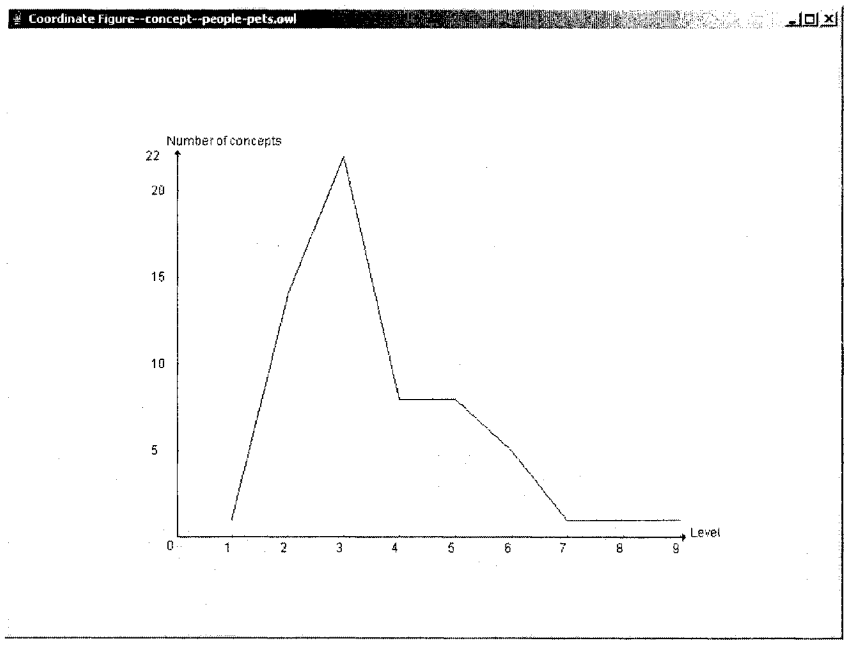
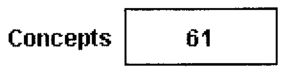


Figure 5-9 General—concept coordinate figure

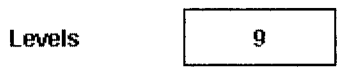
As shown in Table 5-1, if either x or y is less than 1000, we take different scales to set marks in the coordinate axes in order to present results in proper scales. If either x or y is greater than or equal to 1000, we use the logarithmic value of x or y in order to reduce the distortion resulting from having too many concepts at a certain level.

5.7.1.3 The total number of concepts


Concepts 61

The number of concepts is computed as one of the statistics values to measure how big the TBox is. The example ontology is relatively small with 61 concepts. We send the Racer command to get all concepts and arrange the results to count the number of concepts.

5.7.1.4 The number of levels


Levels 9

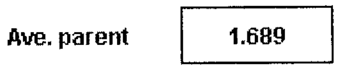
This value is often used together with the concept hierarchy. It presents the depth of the hierarchy. It is computed after the hierarchy has been received from RacerPro.

5.7.1.5 The average number of concepts at levels

Ave concept-level 6.778

The average number of concepts in each level presents the average concept distribution among levels.

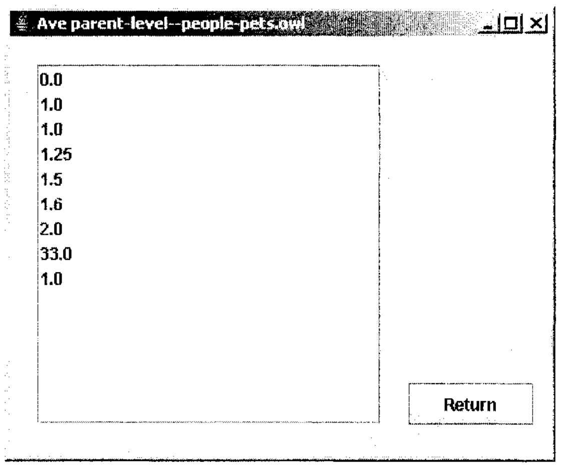
5.7.1.6 The average number of parents for each concept



The parent-child relationship is important in the hierarchy. The number of parents for a concept indicates how many parents a child inherits from.

5.7.1.7 The average number of parents for each concept at each level

This function is like the one described in the last section but indicates of how many parents a concept inherits on average at certain levels. It indicates the trends of the weight of the concept hierarchy. Figure 5-10 displays the average parent number of concepts at each level.



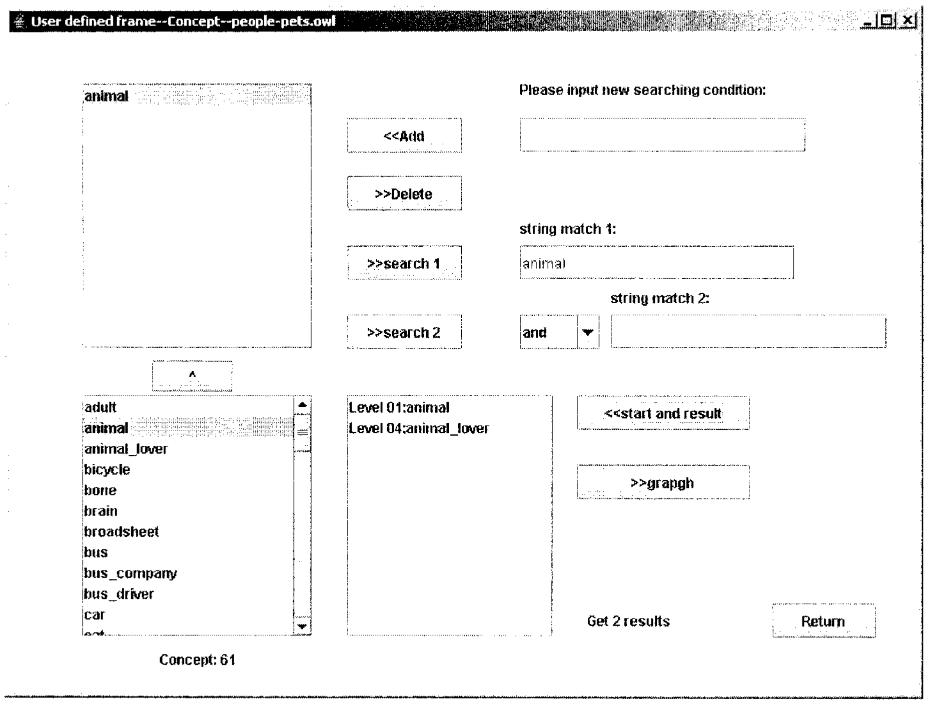
```
0.0
1.0
1.0
1.25
1.5
1.6
2.0
33.0
1.0
```

Figure 5-10 General—average number of parents in each level

5.7.1.8 Concept user-defined evaluation

Figure 5-11 shows the main window for a user-defined concept evaluation. All the concepts and the number of concepts are listed in the left '*all-concept*' list-box on the bottom of the frame.

To add a keyword in the '*search-condition*' list-box, users can directly enter a keyword in the text-field and click the '*add*' button to put it in the '*search-condition*' list-box. To facilitate users' operations, users can easily add a keyword in the '*search-condition*' list-box by selecting a concept and clicking the button above the '*all-concept*' list-box.



adult
animal
animal_lover
bicycle
bone
brain
broadsheet
bus
bus_company
bus_driver
car

Concept: 61

Level 01:animal
Level 04:animal_lover

Get 2 results

Figure 5-11 General—concept user-defined evaluation

For the search function, we provide simple and compound methods. The combo-box contains 'and', 'or', 'not' options. i) Assume a user only enters a keyword in the '*string match 1*' text-field. For example, we entered '*animal*' as shown in Figure 5-11; we click the '*start and result*' button; then we get two results in the '*result*' list-box: '*animal*' (in the 1st level) and '*animal_lover*' (in 4th level). ii) Assume a user only enters a keyword in the 'string match 2' text-field. In this case, only the 'not' option can be used while the '*and*' and '*or*' options do not work. For example, we enter '*animal*' and choose 'not'; then we get all concepts except '*animal*' and '*animal-lover*'. If the option is '*and*' or '*or*', the result is the same as using only '*string match 1*'; iii) Assume a user enters keywords both in '*string match 1*' and '*string match 2*' with one of the '*and*', '*or*', '*not*' options selected, which starts a compound search. This compound search helps meet multi-purpose keyword searches.

After getting search results, users can create a Xmas-tree figure for all the results in order to see their selected distribution, which might be more efficient if we have many results. For example, if we specify the following condition: one keyword '*a*' (in '*string match 1*'), the search option of '*not*', and the other keyword '*ain*' (in '*string match 2*'); then we get the results and the sub-hierarchy figure as shown in Figure 5-12 (a) and (b).

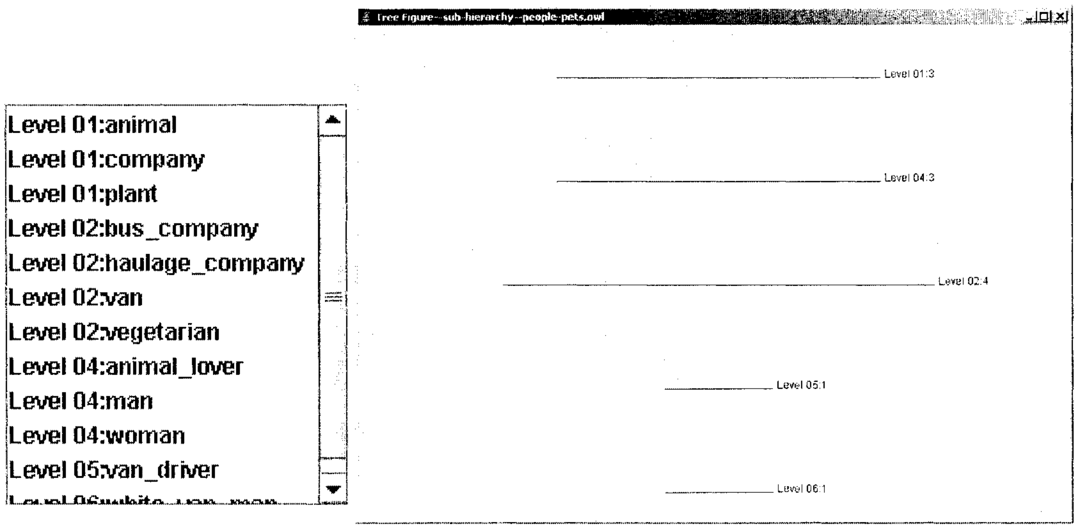
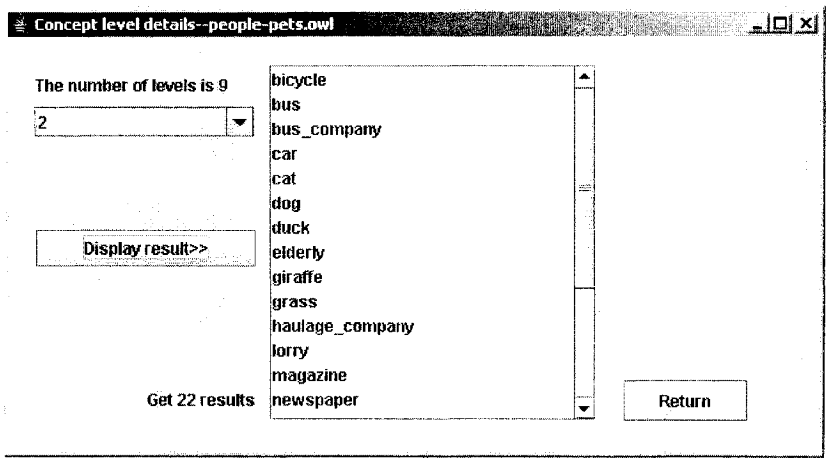

Level 01:animal
Level 01:company
Level 01:plant
Level 02:bus_company
Level 02:haulage_company
Level 02:van
Level 02:vegetarian
Level 04:animal_lover
Level 04:man
Level 04:woman
Level 05:van_driver

Figure 5-12 General—examples of (a) compound search results and (b) the sub-hierarchy

5.7.1.9 Concept level details

This function is a complement for the Xmas-tree and coordinate figures. We list all the concept names from the selected level. Figure 5-13 displays the concepts from the 2nd level as an example.



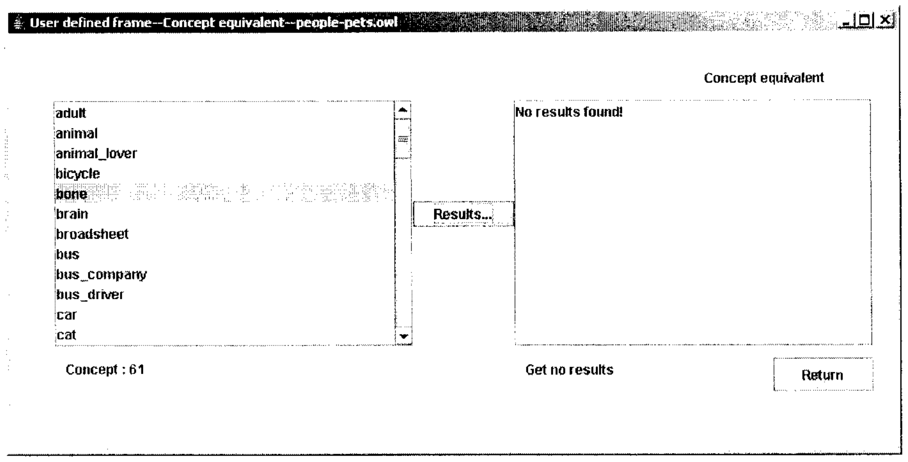
Concept level details--people-pets.owl
The number of levels is 9
2
Display result>>
Get 22 results
Return

```
bicycle
bus
bus_company
car
cat
dog
duck
elderly
giraffe
grass
haulage_company
lorry
magazine
newspaper
```

Figure 5-13 General—concept level details

5.7.1.10 Concept equivalent user-defined evaluation

Concepts can have synonyms, in which case these concepts are called equivalent. Users can check if there are equivalent concepts for a selected one, as shown in Figure 5-14.


User defined frame—Concept equivalent—people-pets.owl

Concept equivalent

adult
animal
animal_lover
bicycle
bone
brain
broadsheet
bus
bus_company
bus_driver
car
cat

Results...

No results found!

Concept : 61

Get no results

Return

Figure 5-14 General—concept equivalent user-defined evaluation

5.7.1.11 Role Xmas-tree figure

This evaluation method is similar as that of the concept Xmas-tree figure. Figure 5-15 displays the role '*people-pets.owl*' in Xmas-tree figure. It only has two levels, which is not a deep hierarchy.

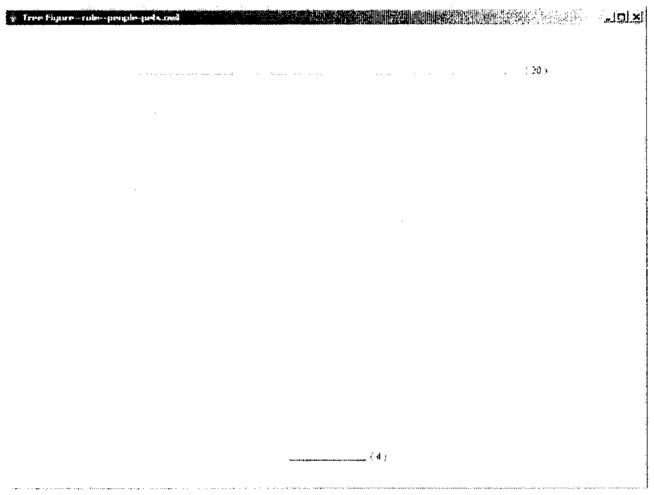


Figure 5-15 General—role Xmas-tree figure

5.7.1.12 Role coordinate figure

This function is similar to the concept coordinate figure. Figure 5-16 shows the role coordinate figure accordingly.

Similar to a detailed concept evaluation, we also provide a detailed evaluation for roles.

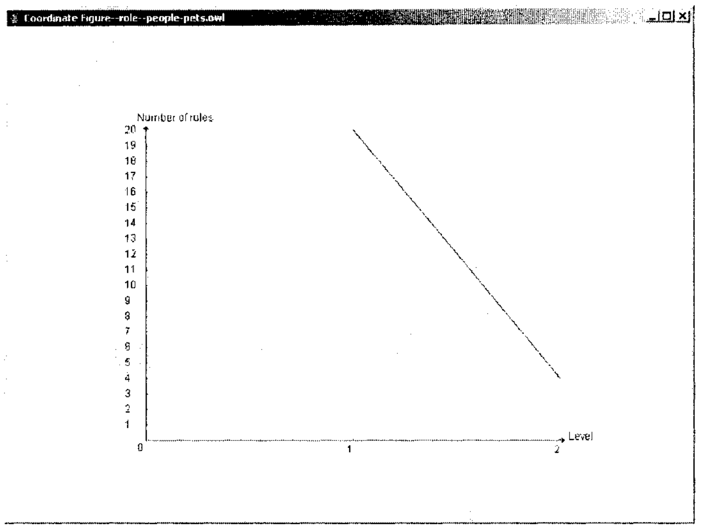


Figure 5-16 General—role coordinate figure

5.7.1.13 Role user-defined evaluation

The main functions are similar to those used for concepts. Users can search for roles, which satisfy the indicated conditions and can have the Xmas-tree figures for the results. The GUI is also the same as the one for concepts.

5.7.1.14 Role level details

This function offers details of roles at each level. Compared to concepts, we count the level number from '1'. Users can also get the number of levels in a role hierarchy.

5.7.1.15 Role transitive user-defined evaluation

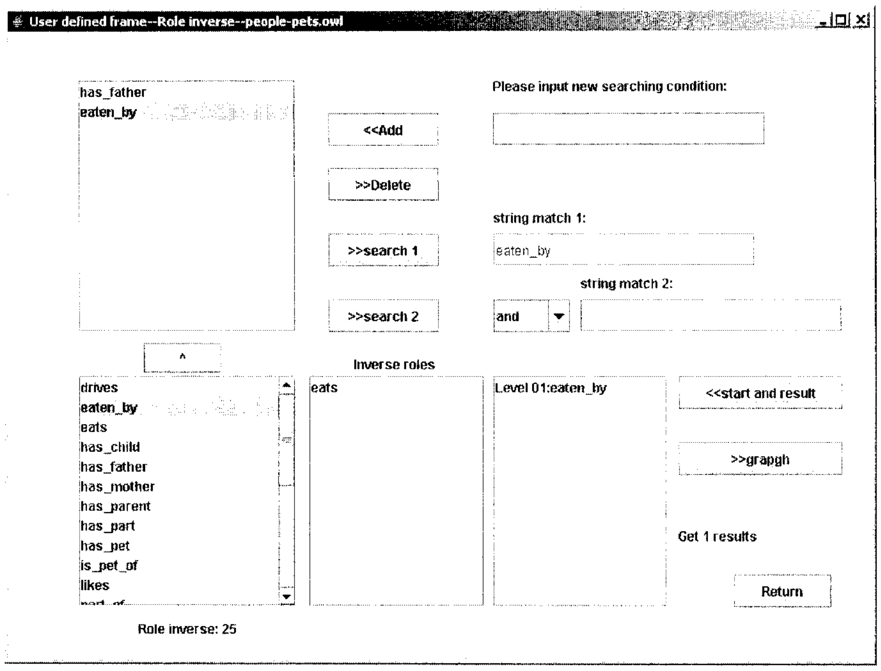
Roles may be transitive. We compute all transitive roles automatically. If no transitive role exists, a message-box will pop up as a reminder. The GUI is similar to that for equivalent concepts.

5.7.1.16 Role symmetric user-defined evaluation

Roles may be symmetric. We compute all symmetric roles automatically. If no symmetric role exists, a message-box will pop up as a reminder. The GUI is similar to that of equivalent concepts.

5.7.1.17 Role inverse user-defined evaluation

Roles may have inverse roles. When users click on one of the roles in the 'all-role' list-box, its inverse role will be displayed on the right automatically if it exists. For example, as shown in Figure 5-17, the role '*eaten-by*' has the inverse role '*eat*'. Other functions in this evaluation are similar to that of the concept user-defined evaluation, such as the simple and compound search, the Xmas-tree figure for all the results, etc.



has_father
eaten_by

Please input new searching condition:

string match 1: eaten_by

string match 2: and

drives
eaten_by
eats
has_child
has_father
has_mother
has_parent
has_part
has_pet
is_pet_of
likes

eats

Level 01:eaten_by

Get 1 results

Role inverse: 25

Figure 5-17 General—role inverse user-defined evaluation

5.7.1.18 Role equivalent user-defined evaluation

Roles may have equivalent roles. This function is similar to the concept equivalent user-defined evaluation. The GUI is similar to that for equivalent concepts.

5.7.2 ABox evaluation

In this section, the functions for individuals, concept assertions, and role assertions are described. The relationship among concepts, roles, and individuals are also presented.

We describe evaluation facilities about relationships between concepts and individuals caused by concept assertions.

We provide individual evaluations based on both concept and role assertions. To explain this more clearly, we introduce the notion of 'tuple' and 'cluster'. Individuals are classified in 'tuples' according to their concept assertions, which indicate all the instances of one 'class'. For the role assertions, we combine the original role assertions retrieved from RacerPro; for example, if *I1*, *I2*, *I3* are individuals, which have the role assertions $I1 \rightarrow I2$, $I2 \rightarrow I3$; then we combine them into $I1 \rightarrow I2 \rightarrow I3$. In this section, we use combined role assertions instead of original role assertions. Even if all assertions from RacerPro are already combined, we may find that several role assertions refer to the same individual; in this case, we allow users to define a similarity degree in order to further combine certain role assertions. We integrate concept assertions and combined role assertions to provide a better ABox evaluation. In the ontology of '*people-pets.owl*', we have the combined role assertions as follows:

$$Minnie \xrightarrow{has\_pet} Tom \qquad Walt \xrightarrow{has\_pet} Louie \qquad Walt \xrightarrow{has\_pet} Dewey$$

$$Walt \xrightarrow{has\_pet} Huey \qquad Fred \xrightarrow{has\_pet} Tibbs \qquad Joe \xrightarrow{has\_pet} Fido$$

$$Rex \xrightarrow{is\_pet\_of} Mick \xrightarrow{read} Dairy\_mirror$$

$$Rex \xrightarrow{is\_pet\_of} Mick \xrightarrow{drives} Q123\_ABC$$

We will discuss them in more detail in later sections of this Chapter.

5.7.2.1 Individual Xmas-tree figure based on concept assertions

We provide the Xmas-tree figure for individuals based on the concept assertions. Every individual has a corresponding concept while not all individuals are necessarily part of role assertions. All individuals are instances of the concept '*top*', so '*top*' has the most number of individuals among all the concepts, which means that in the Xmas-tree figure, the first line is the longest ones. Every individual that belongs to a concept also belongs to the parents of that concept. So the concept-assertion-based individual Xmas-tree is the tree with the widest top and a small bottom. Figure 5-18 illustrates the individual hierarchy.

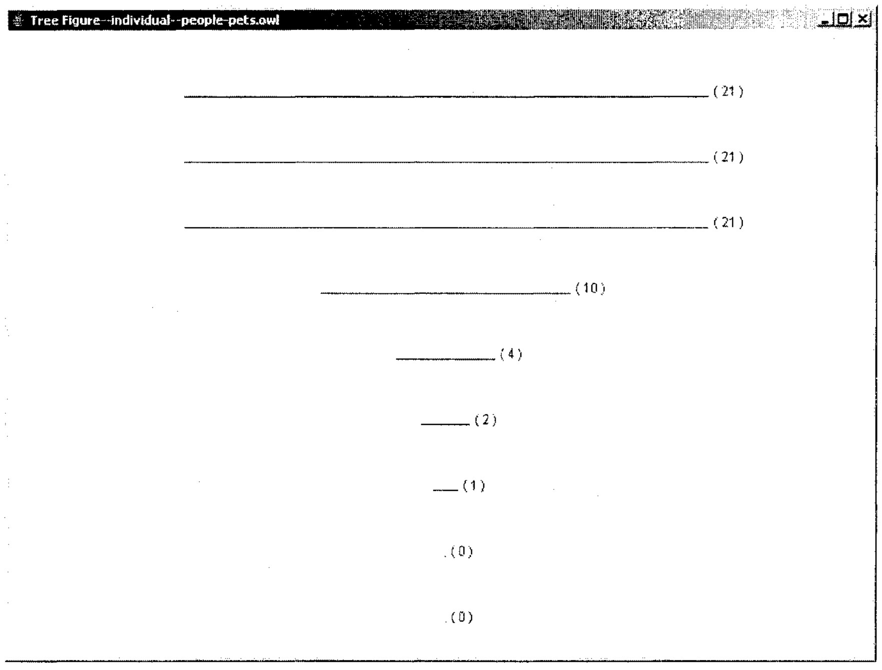


Figure 5-18 General—individual Xmas-tree figure

5.7.2.2 Individual coordinate figure based on concept assertions

The individual coordinate figure is similar to those for concepts and roles.

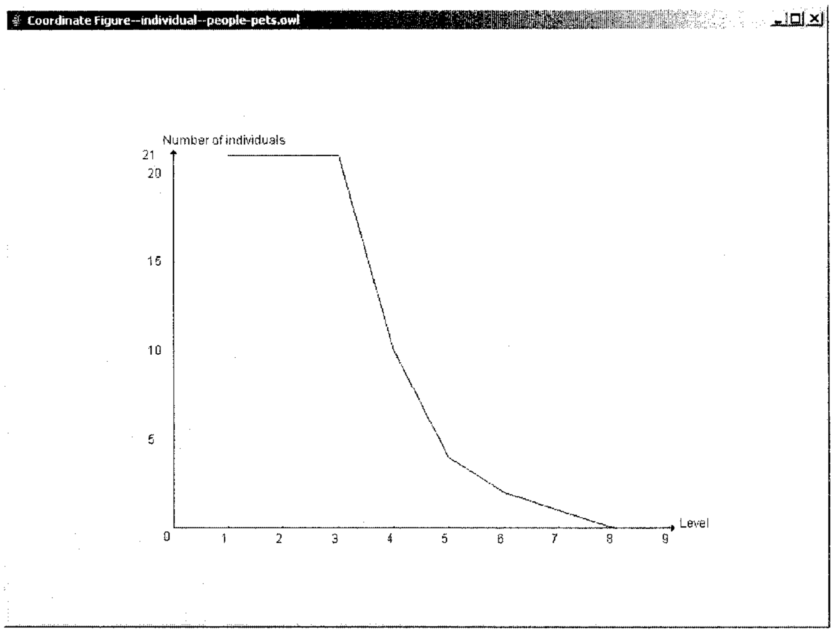
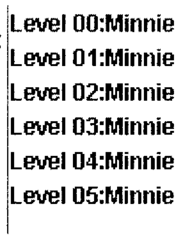


Figure 5-19 General—individual coordinate figure

5.7.2.3 Individual user-defined evaluation based on concept assertions

The main functions are similar to those defined for concepts and roles. Users can search for particular individuals. A search includes a single or compound condition and its results can be represented with Xmas-tree figures. The GUI is the same as that for concepts. The difference from the GUI for the concepts is that a certain individual name can appear many times belonging to several levels while a concept can only belong to one level. For example, if we look for the individuals containing '*Minnie*', we get the following result:



```
Level 00:Minnie
Level 01:Minnie
Level 02:Minnie
Level 03:Minnie
Level 04:Minnie
Level 05:Minnie
```

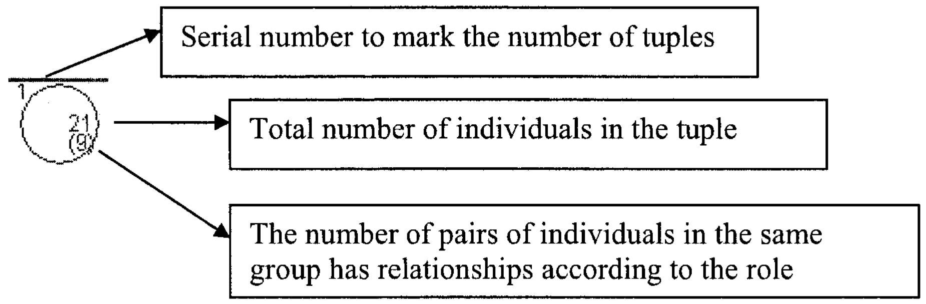
5.7.2.4 Individual level details based on concept assertions

This function offers details of individuals at each level. We count the level number from '0', which is the level where we can get the individuals of '*top*'. The GUI is similar to the GUI for concepts.

5.7.2.5 Individual graph based on tuples

We group all individuals belonging to a concept in one tuple. An example is shown in Figure 5-20. We define the elements in this kind of graph as follows:

- The circles stand for tuples of individuals, which have the same size no matter how many individuals are in a tuple. These circles are distributed evenly from the center of the frame.
- All the individuals in one circle are the instances of a certain concept.
- The numbers around circles have different meanings.



1
21
(9)
Serial number to mark the number of tuples
Total number of individuals in the tuple
The number of pairs of individuals in the same group has relationships according to the role

Figure 5-20 General—representation of numbers around tuple circles

- Relationships between individuals belonging to different circles will be represented by drawing lines between circles, and the number of pairs will be marked on the line if the relationships exist.

Figure 5-21 illustrates the tuple graph of '*people-pets.owl*'. The individuals are classified into 34 tuples and no relationships exist between the tuples.

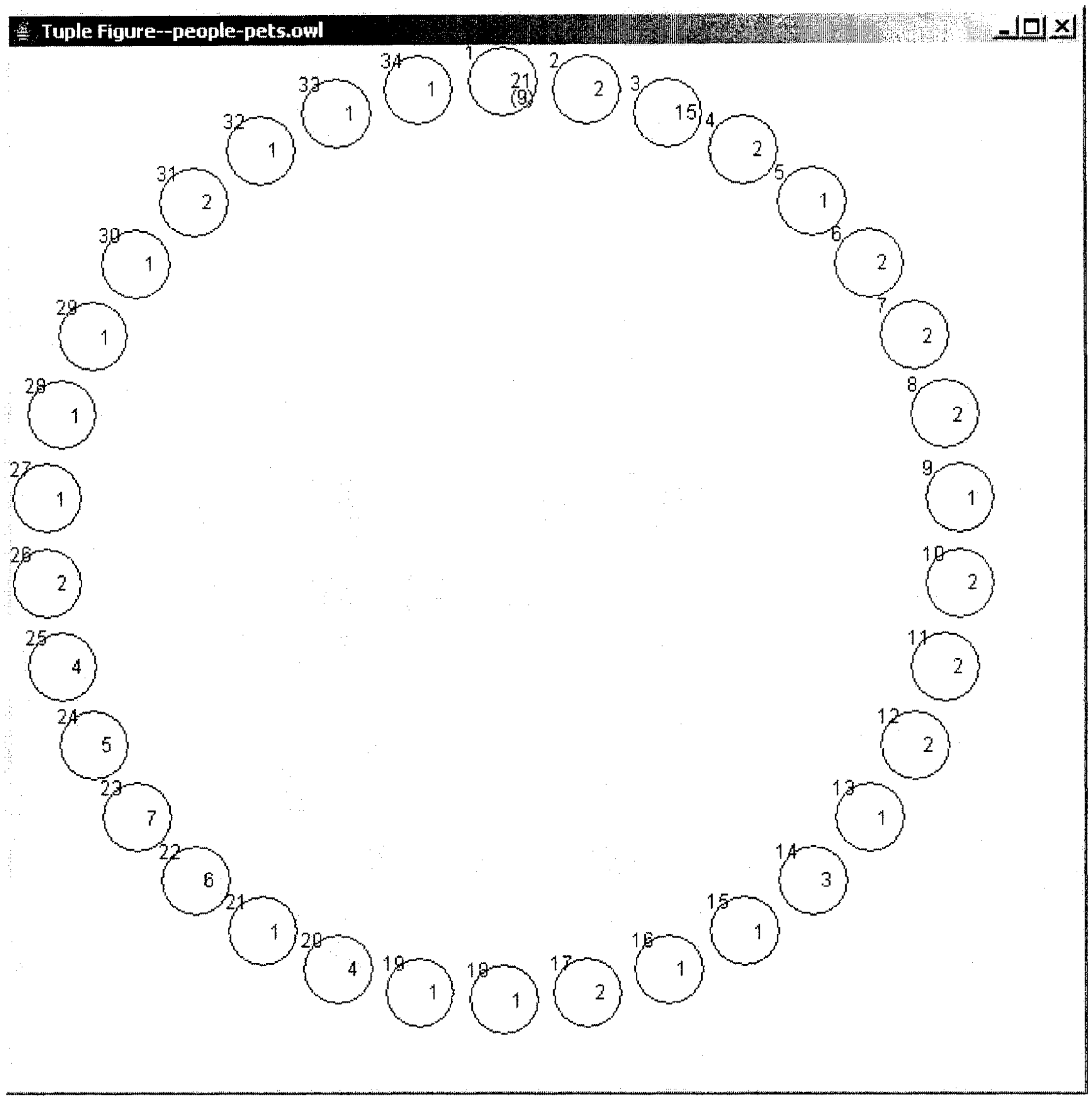


Figure 5-21 General—individual graph based on tuples

5.7.2.6 Individual graph based on combined role assertions

We know that possibly not all the individuals occur in role assertions (or combined role assertions). These individuals not included in any role assertion are grouped separately.

We use circles to represent these classified groups as well. The size of each circle is proportional to the number of individuals it includes. That is to say, the group with the largest number of individuals has the largest circle. We draw the circles from the center of the frame in a descending order. This means the groups with the largest number of individuals are drawn from the center. Figure 5-22 shows an example.

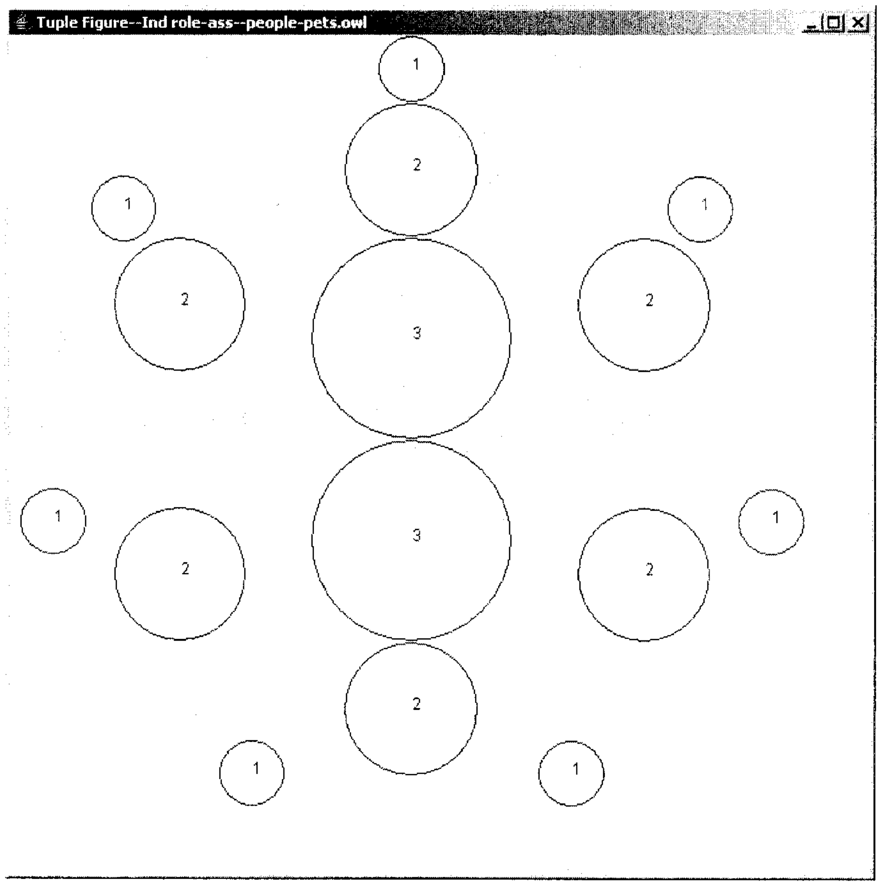
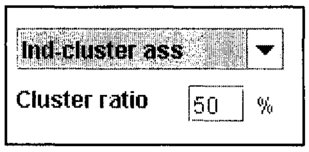


Figure 5-22 General—individual graph based on combined role assertions

Among these groups, relationships may exist as they may include & share certain individuals. In this case, we mainly focus on how individuals are distributed and ignore the relationship between two groups. The 'cluster' shows these relationships in more detail in the next section.

5.7.2.7 Individual cluster graph based on combined role assertions

To measure the degree of similarity between two combined role assertions, we introduce the notion of cluster ratio, which is a value from 1% to 99%, entered by users. Its default value is set to be 50%.



Cluster ratio 50 %

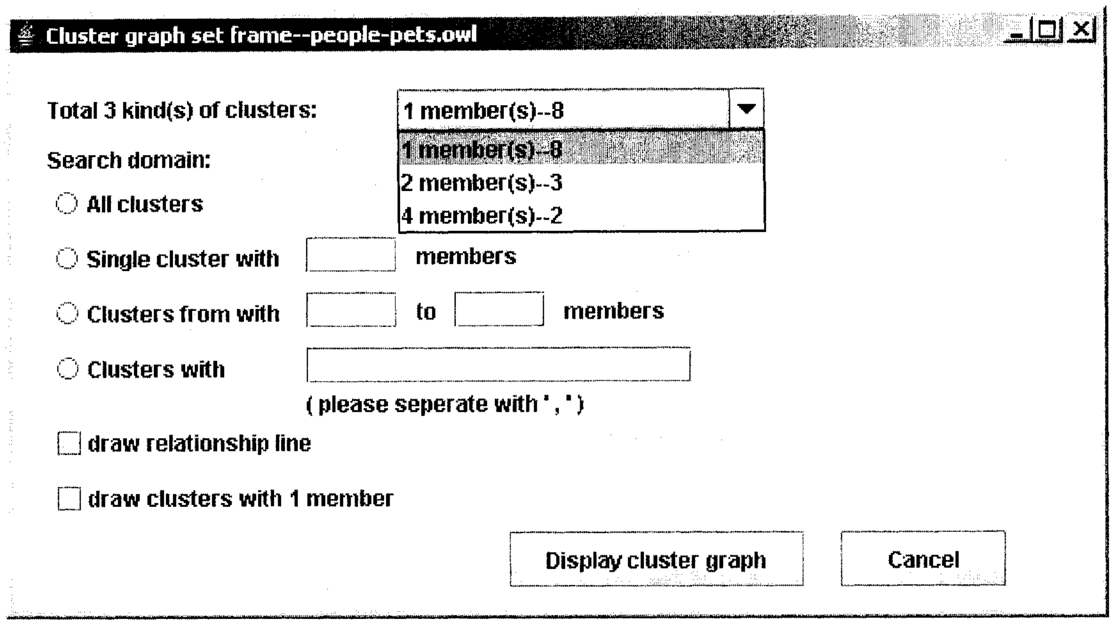
For example, if we consider two combined role assertions: *A: a→b→c→d* and *B: a→b→e*, the same individuals are '*a*' and '*b*', the ratio for A is *ratio_A*= the number of same individuals / the number of individuals = 2/4=50%; the ratio for B is *ratio_B*=2/3=67%

- when ratio=50%, *ratio_A*>=ratio and *ratio_B*>=ratio, we cluster the two combined role assertions into one cluster with *a*, *b*, *c*, *d*, *e*
- when ratio=90%, *ratio_A*<ratio and *ratio_B*<ratio, we do not cluster the two assertions
- when ratio=60%, *ratio_A*<ratio and *ratio_B*>ratio, we do not cluster the two assertions either.

We only combine two combined role assertions in a cluster if their ratios are both more than or equal to the given cluster ratio.

We use ratio=50% as an example to describe the whole process.

- Given a cluster ratio=50%, for '*Rex→Mick→Dairy_Mirror*' and '*Rex→Mick→Q123_ABC*', their ratios are both 2/3=67%>50%, so we cluster these two into one cluster with four elements: '*Rex*', '*Mick*', '*Dairy_Mirror*', and '*Q123_ABC*'; As to '*Walt→Louie*', '*Walt→Dewey*' and '*Walt→Huey*', for any two of them we can get the ratio of 1/2=50%, so we cluster the three into one cluster containing four elements '*Walt*', '*Louie*', '*Dewey*', and '*Huey*'; for the other combined role assertions, they have no common elements, so we do not cluster them.
- The 'cluster graph set frame' is shown in Figure 5-23. Users should choose one in the '*search domain*' to show certain clusters. Furthermore, users have two optional selections: i) draw relationship lines (if any two clusters share certain individuals, then a line is drawn) ii) draw clusters with 1 member (these clusters are less interesting for users, because they only have concept assertions); by default, the number of clusters with 1 member will be computed and the result is printed as a string in the lower-left corner of the graph. This option works only when one chooses to draw all clusters.



Cluster graph set frame--people-pets.owl

Total 3 kind(s) of clusters: 1 member(s)--8

1 member(s)--8
2 member(s)--3
4 member(s)--2

Search domain:

○ All clusters

○ Single cluster with [ ] members

○ Clusters from with [ ] to [ ] members

○ Clusters with [ ]

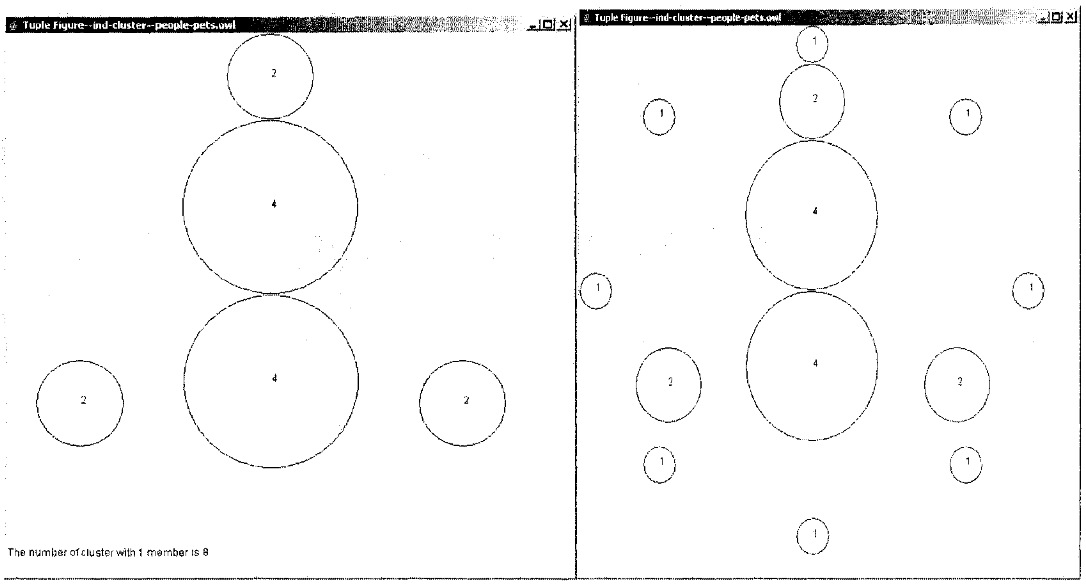
(please seperate with ' , ')

☐ draw relationship line

☐ draw clusters with 1 member

Figure 5-23 General—cluster graph set frame

- Display graph

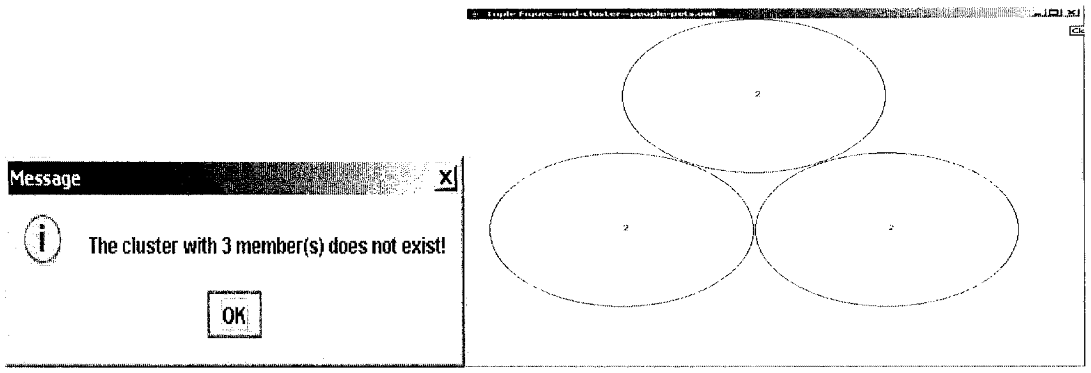


Tuple Figure--ind-cluster--people-pets.owl

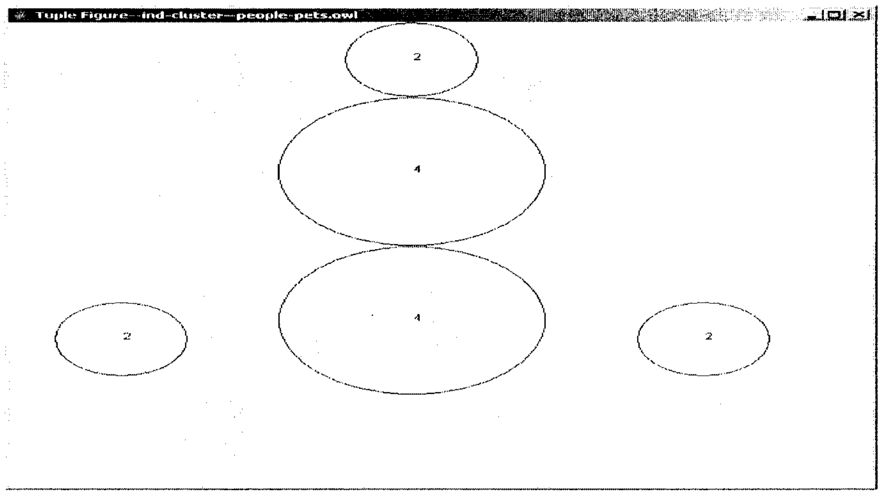
The number of cluster with 1 member is 8

Tuple Figure--ind-cluster--people-pets.owl

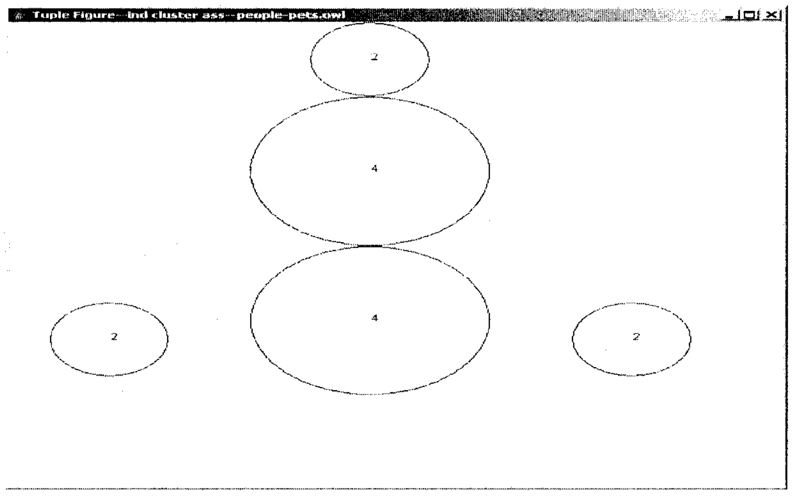
(a) all clusters without and with clusters with 1 member


The cluster with 3 member(s) does not exist!

(b) single cluster with 3 and 2 members



(c) clusters from with 2 to 4 members



(d) clusters with certain numbers of members (2, 3 ,4—2 and 4 exist but 3 not)

Figure 5-24 General—individual cluster graph

In the case of ratio=50%, there are no relationships among clusters. When we change the cluster ratio to 90%, we get clusters with relationship lines (the number in the middle of the line marks the number of same elements) as illustrated in Figure 5-25.

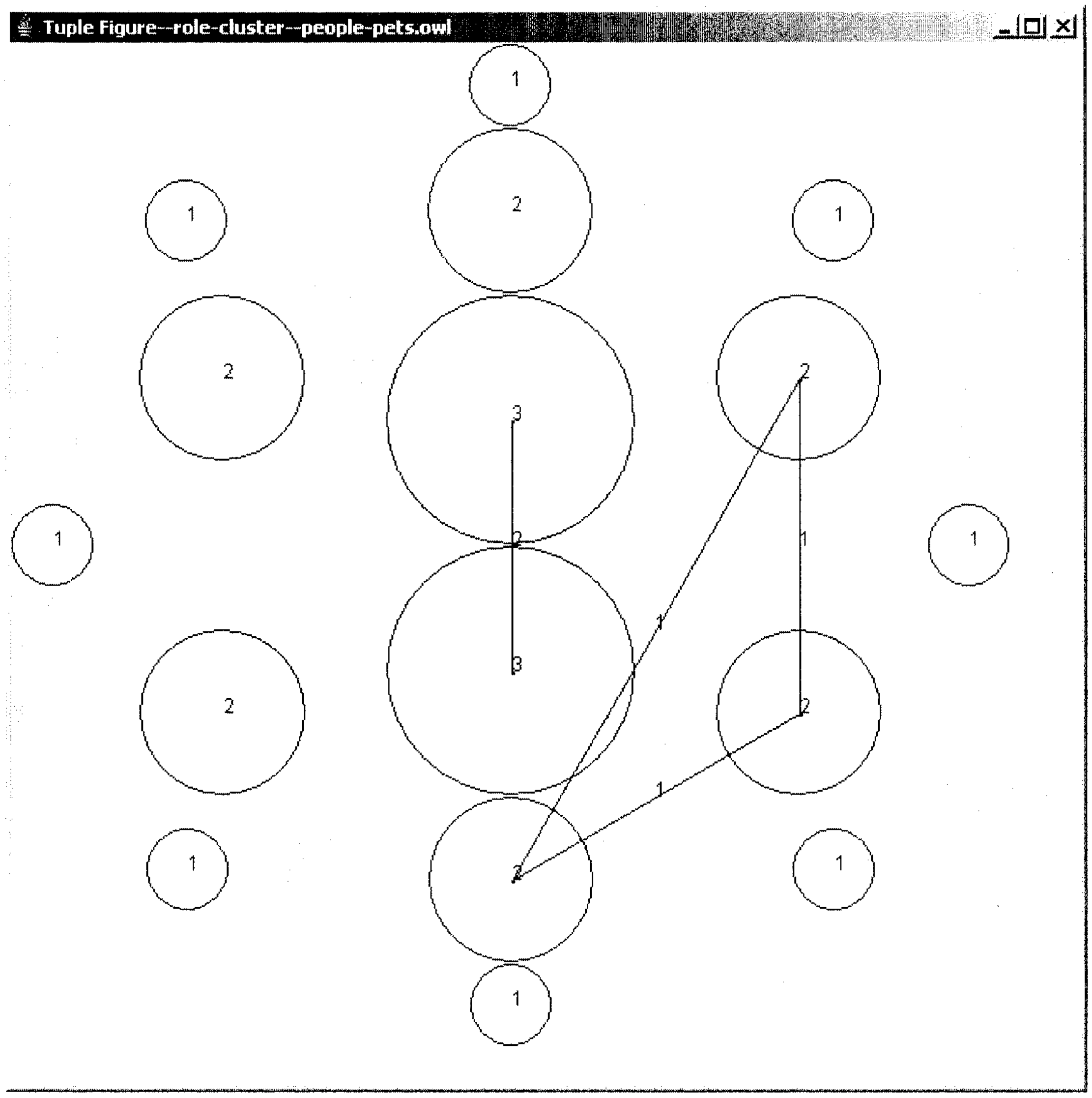
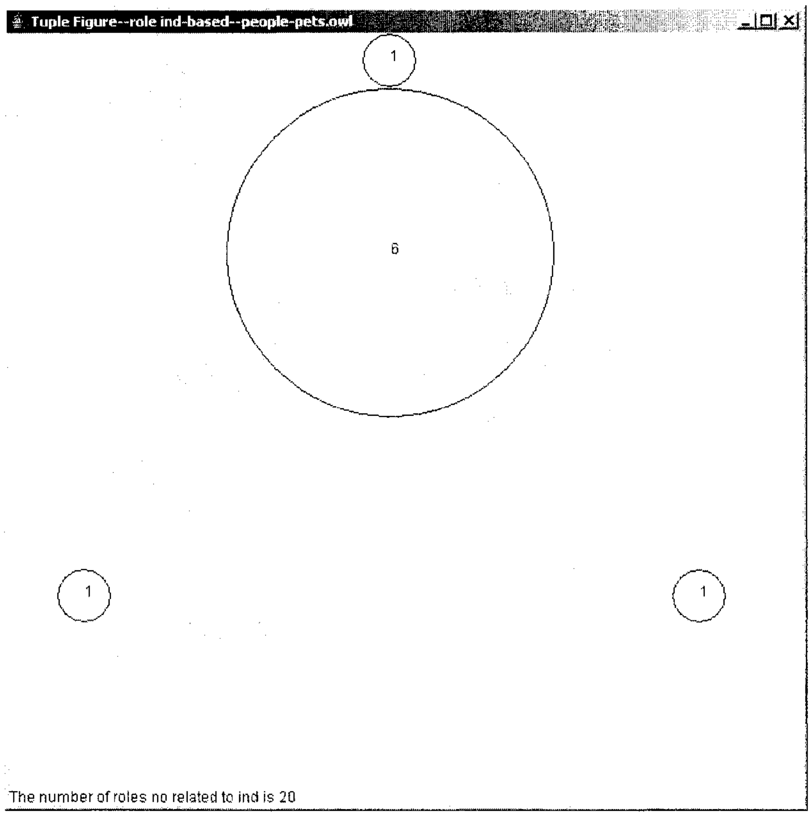


Figure 5-25 General—individual cluster graph (ratio=90%)

5.7.2.8 Role graph based on individuals

This function is used for roles. It gives the number of individual pairs used by roles in role assertions (Figure 5-26).



The number of roles no related to ind is 20

Figure 5-26 General—role graph based on individuals

5.7.2.9 Individual Xmas-tree figure based on assertions

First of all, we will detect if there exist any cycles in our combined role assertions. A cycle exists if an individual can reach itself through a certain number of role assertions. If no such individual is found, no cycle exists then.

If there exist cycles in role assertions, for example: $a \rightarrow b \rightarrow c$ and $c \rightarrow a$, we do not combine them as $a \rightarrow b \rightarrow c \rightarrow a$, because if in this way, the role assertion combination will be very complex. By now we do not have a good solution to deal with this case.

The individuals having only concept assertions are at the first level. In the combined role assertions, the left-most individuals appear at the first level, then the individuals directly on their right on the second level and so on, until all the individuals are marked. The Xmas-tree figure obtained is shown in Figure 5-27.

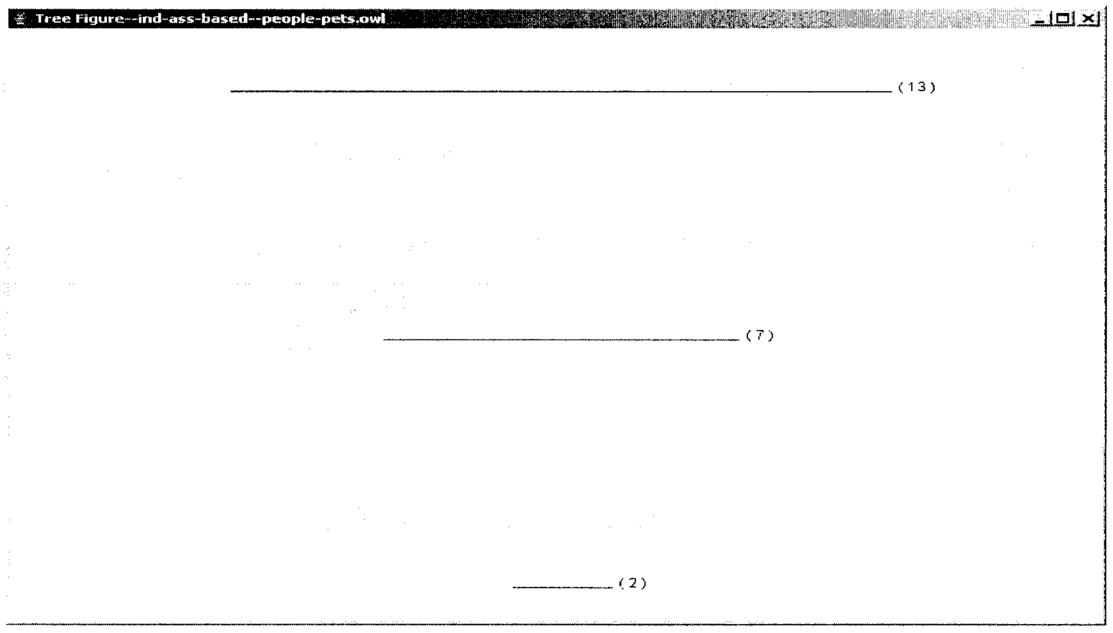


Figure 5-27 General—individual Xmas-tree figure based on assertions

5.7.2.10 Tuple detail retrieval

This function provides detailed information about each tuple. One can get all the individuals in selected tuples, and for each individual we can get the role assertions to which they are related to them. For example in Figure 5-28, we searched the '*top*' tuple and got 21 individuals; for the individual '*Fido*', we get one role assertion (*Joe, Fido): has_pet*. The result is displayed as '*Joe [has_pet]*'. As for the individual '*Fred*', the result will be presented as '*[has_pet] Tibbs*', due to its role assertion *(Fred, Tibbs): has_pet*.

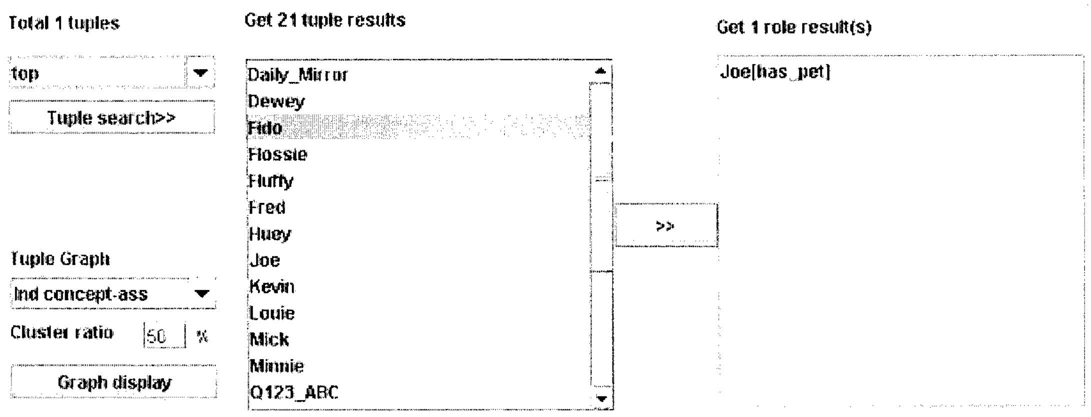


Figure 5-28 General—tuple detail retrieval

5.7.2.11 Combined role assertions retrieval

Before the combined role assertion retrieval starts, the system searches them automatically checks to find out if cycles exist. If any exists, it indicates in a combo-box is displayed, which lists all the cyclic assertions; otherwise, it indicates no cycles exist. As shown in Figure 5-29, we can get all the combined role assertions related to the selected individual.

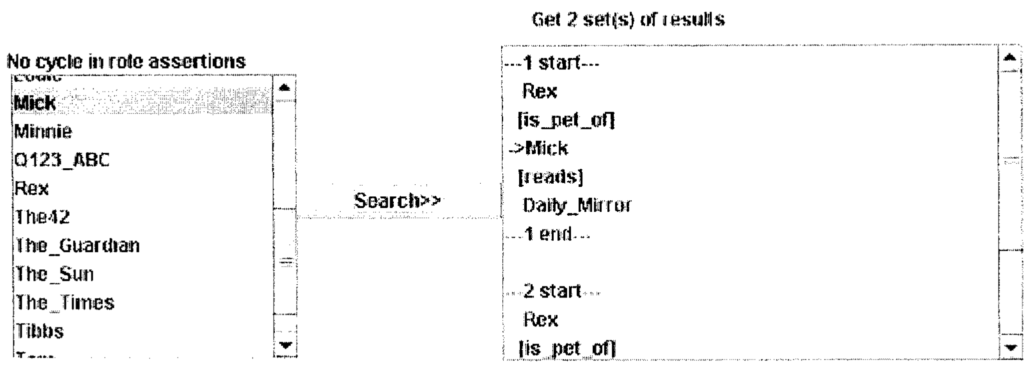


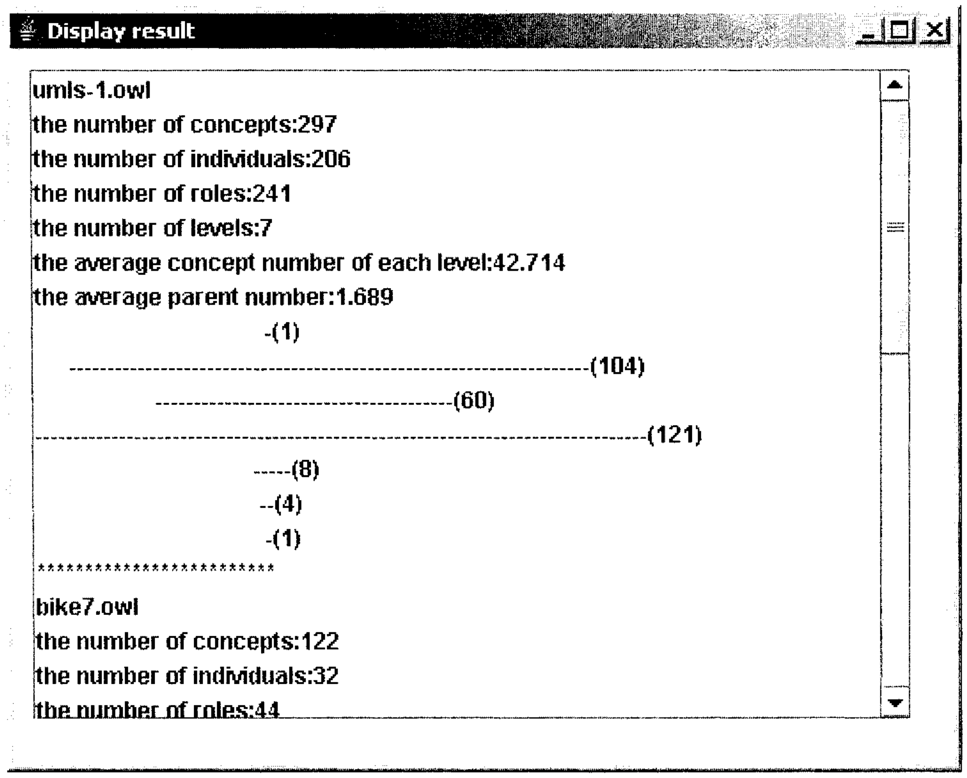
Figure 5-29 General—combined role assertion retrieval

5.8 Multi-file ontology evaluation

Sometimes users have numerous ontologies at hand and they have no idea which ones are interesting. For each of the ontologies, users would have to load a single ontology and evaluate it, and so on. However, this task is time-consuming and users have to take notes about the results.

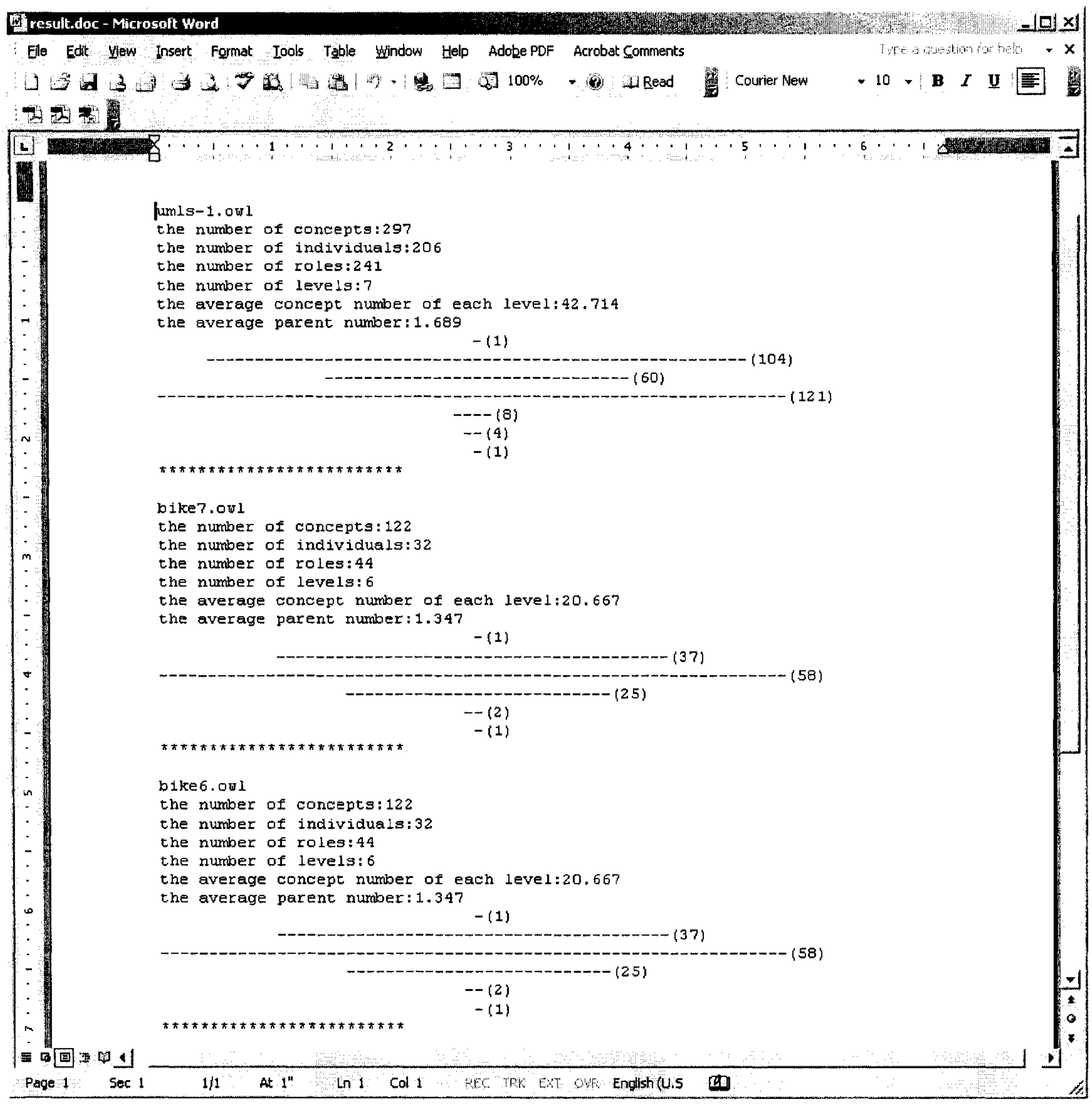
To facilitate this search, we introduce a multi-file ontology evaluation function. One can collect all ontologies in one folder and load these ontologies one after one automatically. This function gives users an overview of these ontologies, in which case the results contain the total number of concepts, individuals, and roles, the number of levels, the average number of concepts at levels, the average number of parents for each concepts, and the concept Xmas-tree figure. If users are interested in certain ontologies, they can perform general ontology evaluation for further information.

The results can be displayed in the user interface and written into a '.doc' file which is created or overwritten in a selected folder.



```
umls-1.owl
the number of concepts:297
the number of individuals:206
the number of roles:241
the number of levels:7
the average concept number of each level:42.714
the average parent number:1.689
                      -(1)
   -----------------------------------------------------------------(104)
            --------------------------------------(60)
-----------------------------------------------------------------------------(121)
                     -----(8)
                      --(4)
                       -(1)
**************************
bike7.owl
the number of concepts:122
the number of individuals:32
the number of roles:44
```

(a) Results shown in the user interface


result.doc - Microsoft Word
File Edit View Insert Format Tools Table Window Help Adobe PDF Acrobat Comments
100%
Read
Courier New
10
Page 1 Sec 1 1/1 At 1" Ln 1 Col 1 REC TRK EXT OVR English (U.S

```
umls-1.owl
the number of concepts:297
the number of individuals:206
the number of roles:241
the number of levels:7
the average concept number of each level:42.714
the average parent number:1.689
                                -(1)
     -------------------------------------------------------(104)
                 -------------------------------(60)
-----------------------------------------------------------------(121)
                              ----(8)
                               --(4)
                                -(1)
*************************

bike7.owl
the number of concepts:122
the number of individuals:32
the number of roles:44
the number of levels:6
the average concept number of each level:20.667
the average parent number:1.347
                                -(1)
            ----------------------------------------(37)
-----------------------------------------------------------------(58)
                   ---------------------------(25)
                               --(2)
                                -(1)
*************************

bike6.owl
the number of concepts:122
the number of individuals:32
the number of roles:44
the number of levels:6
the average concept number of each level:20.667
the average parent number:1.347
                                -(1)
            ----------------------------------------(37)
-----------------------------------------------------------------(58)
                   ---------------------------(25)
                               --(2)
                                -(1)
*************************
```

(b) Results shown in a '.doc' file

Figure 5-30 Multi-file—results

6. System testing

In this chapter, we report on the evaluation results using several ontologies with sizes from 40KB to about 8MB. These results are analyzed and discussed.

6.1 Ontology selection

For system testing, we used about 60 ontology files from several KB to about 8 MB, with large TBoxes and/or ABoxes. We list some typical ontology files in Table 6-1.

Ontology Name	**Size**	**TBox**	**ABox**	
			Concept assertions	**Role assertions**
people-pets.owl	39.4KB	Yes	Yes	Yes
wine.owl	78.7KB	Yes	Yes	Yes
bike7.owl	164KB	Yes	Yes	No
umls-1.owl	220KB	Yes	Yes	No
galen.owl	2.31MB	Yes	No	No
umls-2.owl	7.47MB	Yes	No	No

Table 6-1 Typical ontology files for system testing

6.2 Testing results

We implemented our tool in Java with NetBeans 4.1. JRacer acts as the bridge to deal with the communication between Java and RacerPro. In the system testing process, we compare our results with those from RacerPorter using Racer commands in order to confirm correctness of the results.

6.2.1 TBox

All the ontology files used have a TBox, which includes concepts and roles. We can retrieve all the information from the detailed evaluation part. Table 6-2 illustrates the most important results for TBoxes.

	people-pets	wine	bike7	umls-1	galen	umls-2
Number of concepts	61	208	122	297	2795	9477
Number of concept levels	9	10	6	7	17	10
Average number of concepts	6.778	20.2	20.667	42.714	161.824	947.9
Average number of parents	1.689	1.614	1.347	1.689	1.441	1.156
Number of roles	24	27	43	240	422	9550
Number of role levels	2	2	2	4	10	5

Table 6-2 Main TBox results

From the results, one can see that having more concepts does not mean having a deeper concept hierarchy. Although the number of concepts in '*galen*' is less than those of '*ulms-2*', and it has a deeper hierarchy with a maple-leave like distribution, while the hierarchy of '*ulms-2*' looks like a sword, where 7740 concepts out of 9477 are in the 1st level and much less concepts in other levels. Figure 6-1 illustrates the Xmas-tree figures.

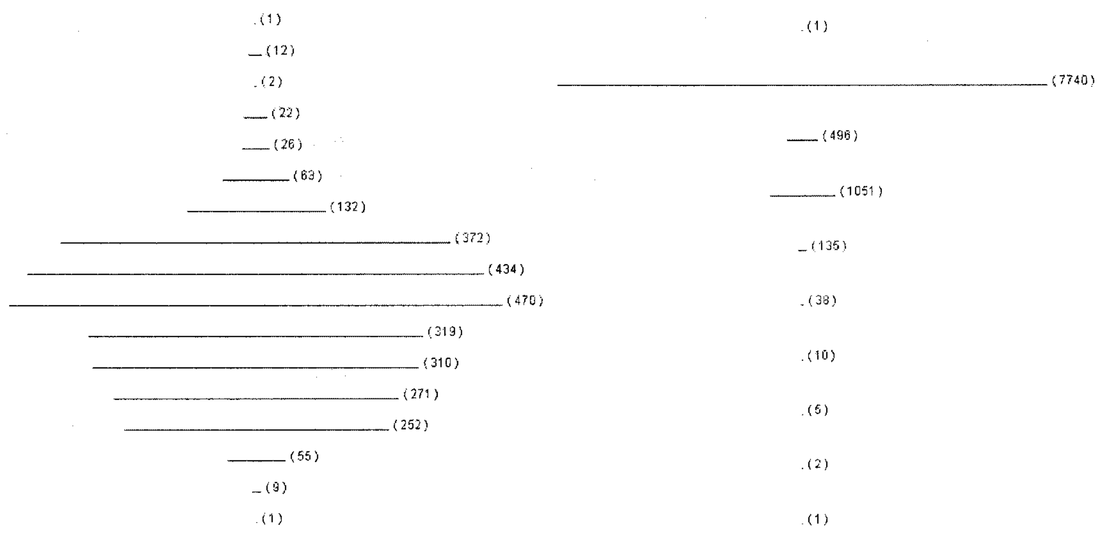


Figure 6-1 Xmas-tree figures (a) '*galen*' (b) '*umls-2*'

We mentioned in the previous chapter that if the number of concepts or the number of levels exceeds 1000, we adjust the coordinate scale to be logarithmic in order to reduce the untrue representation due to the big difference between the longest and the shortest lines. As shown in Figure 6-1 (b), the 1st level of '*ulms-2*' has 7740 concepts; so the coordinate figure changes to a non-linear one, in which the y-coordinate scales are marked as 10, 100, 1000 and 7740 (the total number of concepts in this level) is shown in Figure 6-2.

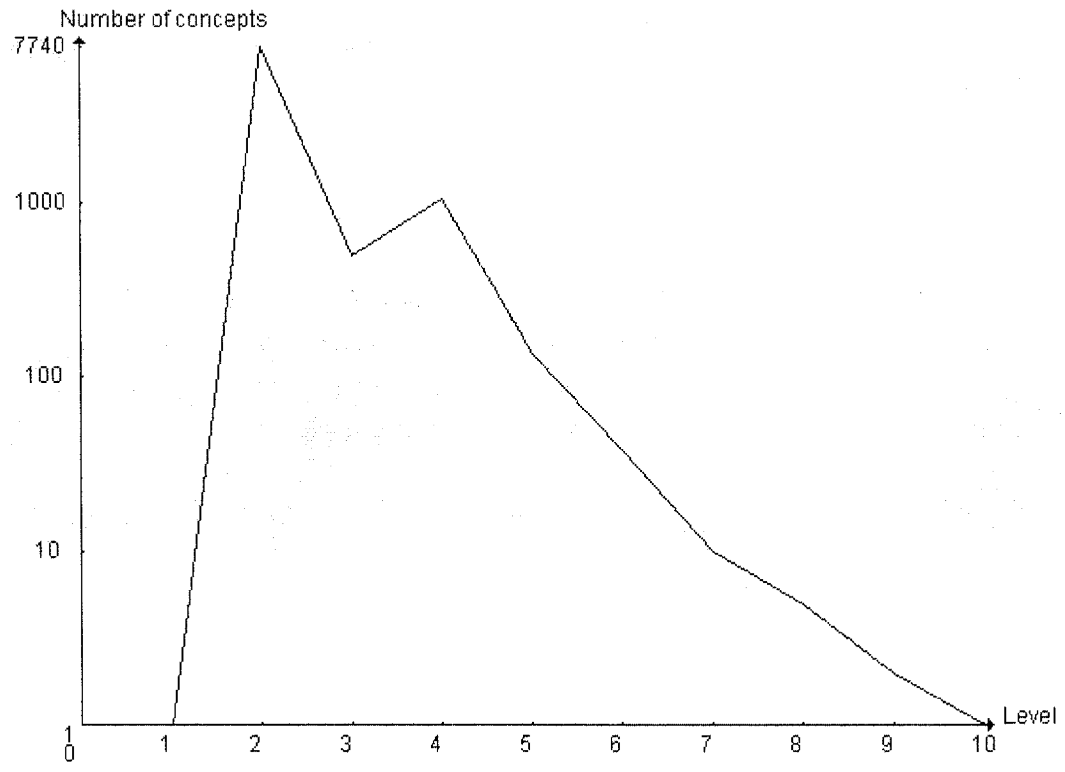


Figure 6-2 Logarithmic scale coordinate figure of '*umls-2*'

With all these testing ontologies, Xmas-tree and coordinate tree figures work very well. However, if we get thousands of levels, the Xmas-tree might become a totally 'black' filled figure and cannot reflect the true situation of each level. However, the coordinate-tree figure may partly solve this problem with a logarithmic scale. It works better to deal with huge ontologies with thousands of levels.

6.2.2 ABox

	people-pets	**wine**	**bike7**	**umls-1**	**galen**	**umls-2**
Number of individuals	21	208	32	206	0	9339

Table 6-3 The number of individuals

For the ABox, we only consider the results of '*people-pets*' and '*wine*' here.

For the detailed evaluation, we always get our expected results. However, for the ABox graphs we get some unexpected results in some circumstances.

- The individual graph based on tuples: that of '*wine*' is unreadable when ontologies have more than 120 tuples as illustrated in Figure 6-3 (167 tuples), compared with '*people-pets*' with 34 tuples. By now, we do not have better solutions to solve this problem. In the '*wine*' case, an alternative way is to refer to the detailed tuple evaluation part for the number of tuples, individuals in each tuple, and role assertions related to these individuals.

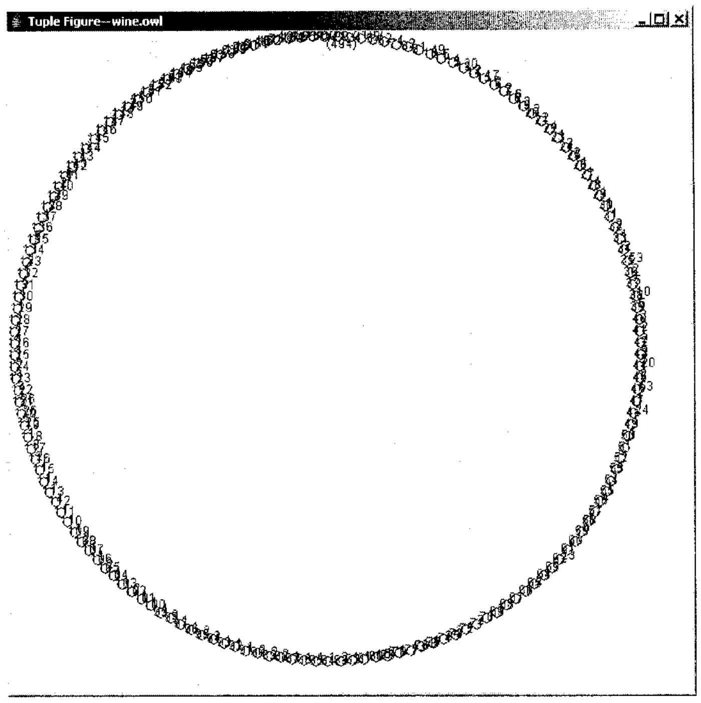


Figure 6-3 The individual graph based on tuples of '*wine*'

- The individual graph based on combined role assertions: users may encounter problems when there are too many circles. Figure 6-4 is this kind of graph of '*wine*'. We get many more circles with 2 members, which causes the circles size with 2 members to be smaller than that with 1 member. If even more circles exist, the graph will be more complicated and less informative like 'the individual graph based on tuples'. However, this is a specific case for 'individual cluster graph based on combined role assertions' with a cluster ratio of 100%. We can use the 'individual graph based on combined role assertions' function to separate the whole graph into parts in order to zoom in parts of the whole graph. It can partly solve this problem.

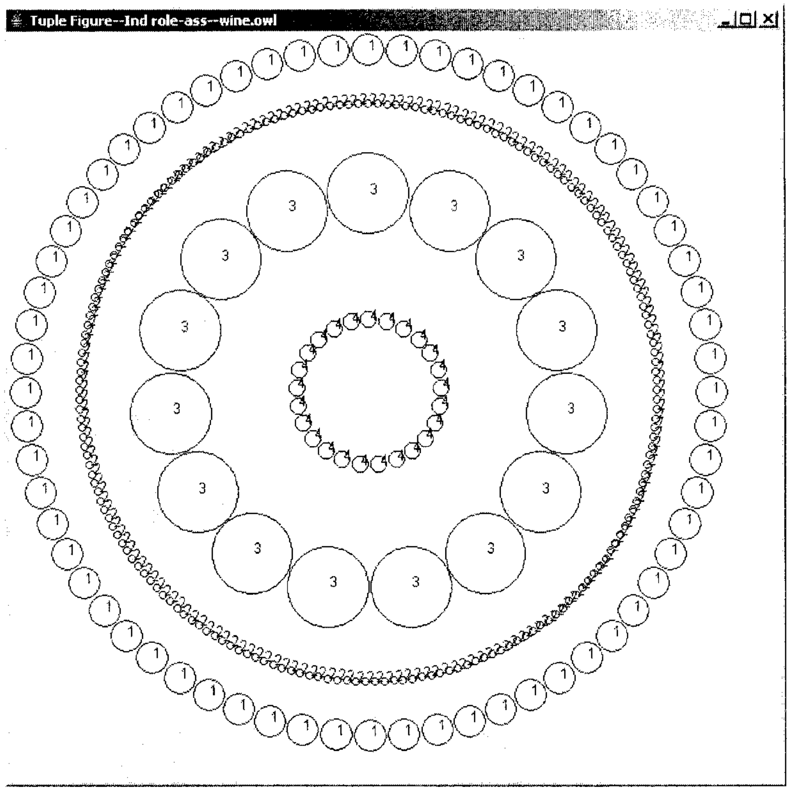


Figure 6-4 The individual graph based on combined role assertions of '*wine*'

- Individual cluster graph based on combined role assertions: This function partly solves the problem of the individual graph based on combined role assertions with a large number of circles by choosing to view parts of the circles with certain members. For these testing ontologies, this function can give us a good evaluation view of these ontologies. However, even if we only view clusters with a certain number of members, lots of relationship lines drawn can make the cluster graph unreadable, and more, the similar situation will occur as that of 'the individual graph based on combined role assertions', if there are hundreds of clusters.
- Role graph based on individuals: This function may encounter the same problems as that for the 'individual graph based on combined role assertions'. If we have a large

number of role assertions with hundreds of different roles, the graph may become too complex to read.

- Individual Xmas-tree figure based on assertions: this function presents structures similar to that of concept hierarchies. In our testing process, the number of levels is not deep; the '*wine*' only has 4 levels. When hundreds of levels are found, the problems of possible untruth presentations may occur as in the case of Xmas-tree figures.

6.3 Discussion

Ontologies are different: some ontologies have big TBoxes but small ABoxes such as '*galen*' and '*umls-2*'; some have weak TBoxes but strong ABoxes like '*people-pets*' and '*wine*'. The more data we get from ontologies, the more complex the figures and the less readable they become. Small TBoxes and ABoxes can be dealt with much easier. The large parts can be properly separated into several related parts. However, after analyzing the testing results, we find that the main problems occur when we create large data overview presentations. How to separate large one to relative smaller parts and how small for one part are difficult to decide due to various properties of ontologies.

Another problem caused by large ontologies is the speed of the hierarchy computation. For example, the '*umls-2*' is a large ontology. RacerPro can process it in 15-20 seconds (as to '*galen*', about 6-7 seconds), however, the computation of the concept hierarchy is time-consuming. The main problem is to assign the level number for each concept as in the case presented in Figure 5-4 (d). We have to search each concept to check if this case exists. As '*umls-2*' with 9477 concepts, we have 9476 checks to perform. In this case, users have to wait for several minutes to get the final results. We will try to develop better solutions to increase the performance of the computation process in our future work.

6.4 Summary

Through Chapter 4, we described other ontology evaluation support methodologies and tools for ontology evaluation. Different approaches take different methods to implement the evaluation for different stages of ontologies such as pre-modeling, modeling, and after-release. Modeling and after-release stages are more considered. For the modeling stage, evaluation work focuses on taxonomy problem checks and modifications. As a lot ontology models support multiple inheritance, inconsistencies may exist (for example, those described in Section 4.1), which cause a major problem for ontologies being used. For the after-released consistent ontologies on the web, we suppose no inconsistency exists and what we need to do is to find out how elements form structures and are classified by similar characteristics. Some of them provide the presentation of elements or hierarchies to help understand ontology structure and contents.

Although many methodologies were proposed, the ontology tools are very limited to facilitate evaluation especially the evaluation for every facet of ontologies. We find that most of these works only consider TBox evaluation. However, ABox is also a very important part of an ontology. Few of recent ontology evaluation support methodologies or tools cover ABox evaluation. For our OntoKBEval tool, besides TBox evaluation support, it provides illustrations for parts of ABox element relationships description. This can be a possible solution to improve ABox evaluation methods.

We implemented the OntoKBEval system with the help of DL technology using an OWL DL reasoner to access ontology retrievals. To the best of our knowledge, no other ontology evaluation support method or tool currently uses an OWL DL reasoner for the evaluation process.

Table 6-4 illustrates main features and difference of ODEval, OntoManager, OntoClean, CleanONTO, CORE, the methodology for ontology evolution and OntoKBEval.

	ODEval	OntoManager	OntoClean	CleanONTO	CORE	Methodology for evolution	OntoKBEval
General description	A tool to evaluate concept taxonomies from a knowledge representation point of view A complement to ontology parsers and ontology platforms, which can evaluate RDF(S), DAML+OIL, and OWL ontologies.	A workbench for verification and validation of ontologies in order to detect and resolve mismatches	A methodology for validating the ontological adequacy of taxonomic relationships	A tool to check inconsistency in an ontology and arrange it to a consistent one	A tool for collaborative ontology reuse and evaluation	A methodology to guide the ontology evolution by doing the ontology evaluation functions	A tool to evaluate OWL ontologies with the help of Description Logics

(Table 6-4 continues)

	ODEval	OntoManager	OntoClean	CleanONTO	CORE	Methodology for evolution	OntoKBEval
Main methodology description	Using a set of algorithms based on graph theory	To integrate existing ontology development tools and enable interoperability between them in order to facilitate the management of ontologies	Using a set of meta-properties named unity, identity, essence and dependence, which are used to characterize relevant aspects of the intended interpretation of the properties, classes, and relations that make up an ontology. These meta-properties dictate several constraints on the taxonomic structure of an ontology, which can be used to evaluate ontology	i) acquire descriptions for each concept ii) break inappropriate links iii) arrange to form a consistent tree	To provide automatic similarity measures by comparing Golden Standard to other ontologies Using rank fusion techniques to retrieve a ranked list of ontologies for each criterion	To provide the flexibility to essentially define arbitrary ontology evaluation functions for a variety of contexts and is open to embed new methods for change discovery leading to improved ontologies.	i) illustrate overall hierarchical structures of TBoxes computed from '*top*' to '*bottom*' and ABoxes with classified by tuple and cluster ii) detailed information browsing evaluation which help enforce the evaluation criteria

	ODEval	**OntoManager**	**OntoClean**	**CleanONTO**	**CORE**	**Methodology for evolution**	**OntoKBEval**
Main problems solved	Inconsistency (circularity issues and partition errors) Redundancy grammatical problems	To find the 'weak places' in the ontology, and modifying it, regarding the end-users' needs/ requirements	To evaluate concepts and roles with '*IS-A*' taxonomic relationship	To modify the inconsistency of an ontology	To get a rank list of ontologies by comparing with defined Golden Standard and users' feedback	To create a good ontology for a particular context	To evaluate ontologies by building overall hierarchical structures of TBoxes and ABoxes with detailed information as references and do statistics analysis to re-classify DL KB elements
State of ontology	when ontologies finish	after release or in usage	when ontologies finish	when ontologies finish	after release	during or before ontology evolution	after release
Lexical evaluation support	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Taxonomy evaluation support	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ABox evaluation support	No	Yes	No	No	No	Yes	Yes

(Table 6-4 continues)

	ODEval	**OntoManager**	**OntoClean**	**CleanONTO**	**CORE**	**Methodology for evolution**	**OntoKBEval**
Reasoning support	According to ontology parsers and ontology platforms	--	-- (methodology)	--	--	-- (methodology)	RacerPro1.9
Statistics	No	No	No	No	No	No	Yes
Element detail retrieve	No	No	No	No	No	No	Yes
Type of ontology	RDF(S) DAML+OIL OWL	RDF(S) DAML+OIL OWL	All	All	a set of ontologies form the Protégé OWL repository	OWL	OWL
Elements re-classification	No	No	No	No	No	No	Yes
GUI	No	No	No	Yes	Yes	No	Yes

Table 6-4 Main features of ODEval, OntoManager, OntoClean, CleanONTO, CORE, the methodology for ontology evolution and OntoKBEval

7. Conclusion and future work

In this chapter, we give a summary about our work on the OntoKBEval system. We also propose possible future work in this area.

7.1 Conclusion

We introduced the OntoKBEval system to implement OWL ontology evaluation with the help of Description Logics for getting information from knowledge bases. It can guide ontology reuse, modification or even possible further evolution after ontology release.

The tool provides the visualized figures for members of TBoxes and ABoxes, which give qualitative and quantitative data to evaluate the structures of ontologies. Users are guided to know how the elements distribute in the hierarchies. Ontologies are determined as good or bad ones from a user's point of view.

Within the process of evaluation, detailed information is necessary and important. It provides users with information to decide whether to continue the evaluation in certain parts of knowledge bases. The overall and detailed information integrates together to form a better evaluation solution.

To better use our tool, users should have a basic knowledge about ontologies. The tool can be applied as a complement to perform users' tasks related to ontologies.

7.2 Future work

By testing our tool, we propose mainly three possible ways to improve or further develop our work in the ontology evaluation area.

Although we can load large ontologies up to about ten MB, using larger ontologies is still limited. The algorithms for computing the hierarchical structures should be optimized to save processing time.

In our research, we only consider after-release ontologies. Actually, in the lifetime of ontologies, they come through mainly three stages: pre-modeling (main problem encountered—conflict), modeling (possible problems of inconsistency and logical errors), and after-release stage. To extend the evaluation functions, functions for these stages can be added. In this case, the evaluation tool can be integrated with ontology editors, where the evaluation process could be synchronized with ontology building.

The GUI can be improved to facilitate users' operations and navigate evaluation directions to achieve users' goals more efficiently. We could also extend the application to a web-based application, which can more easily implement ontology reusage and share the results of evaluation results among users. It is to say that one user can use the other users' results and feedbacks as guidance for ontology usage or further evaluation if the system can find the same or similar results, in which case users do not have to do the extra time-consuming re-evaluation work.

8. References

[1] A Microbial Ontology, http://bioinfo.unice.fr/ontologies/Introduction_to_ontologies.html

[2] Aslam, Javed A., Montague, Mark. *Models for metasearch.* 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2001). New Orleans, Louisiana, 2001, pp. 276-284.

[3] Baker, Christopher J.O., Warren, Robert H., Haarslev, Volker, Butler, Greg. *Status Quo of Ontologies in the Public Domain* (Submitted)

[4] Bechhofer, Sean, Crowther, Peter, and Möller, Ralf. *The description logic interface.* In D. Calvanese, G. De Giacomo, and E. Franconi, editors, International Workshop on Description Logics, pages 196–203, September 2003.

[5] Brank, Janez, Grobelnik, Marko, Mladenić, Dunja. *A survey of ontology evaluation techniques*, In: SIKDD 2005 at multiconference IS 2005, 17 Oct 2005, Ljubljana, Slovenia.

[6] Brank, Janez, Grobelnik, Marko, Mladenić, Dunja. *D1.6.1 Ontology evaluation*, www.sekt-project.org/rd/deliverables/wp01/, 2005

[7] Brewser, Christopher, Alan, Harith, Dasmahapatra, Srinandan, Wilks, Yorick. *Data driven ontology evaluation*, 2004. In Proceedings of International Conference on Language Resources and Evaluation, Lisbon, Portugal.

[8] Brickley D. and Guha R.V. RDF vocabulary description language 1.0: RDF Schema, http://www.w3.org/tr/2002/wd-rdf-schema-20020430/, 2002

[9] Corcho, Óscar, Gómez-Pérez, Asunción, González-Cabero, Rafael and Suárez-Figueroa, M. Carmen. *ODEval: a tool for evaluating RDF(S), DAML+OIL, and OWL*

concept taxonomies, IFIP WG12.6 -- First IFIP Conference on Artificial Intelligence Applications and Innovations (AIAI2004). Toulouse, France. August 2004.

[10] Description logics, http://dl.kr.org/

[11] Fellbaum, Christiane. *WordNet: An Electronic Lexical Database.* MIT Press, 1998.

[12] Fernández, Miriam, Cantador, Iván, Castells, Pablo. *CORE: A Tool for Collaborative Ontology Reuse and Evaluation.* 4th International EON Workshop, 2006.

[13] Flouris, Giorgos, Plexousakis, Dimitris. *Handling Ontology Change: Survey and Proposal for a Future Research Direction.* Technical Report 362, ICS-FORTH, Heraklion, Crete, Greece, September 2005

[14] Gómez-Pérez, Asuncion. *Evaluating ontologies: Cases of Study.* IEEE Intelligent Systems and their Applications. Special Issue on Verification and Validation of ontologies. March 2001, Vol 16, N° 3. Pag. 391 – 409.

[15] Goodaire, Edgar, Parmenter, Michael. *Discrete Mathematics with Graph Theory.* Ed. Prentice Hall. 1998.

[16] Gruber, Thomas R. *A translation approach to portable ontologies.* Knowledge Acquisition, 5(2):199-220, 1993.

[17] Guarino, Nicola and Welty, Christopher A. *An Overview of OntoClean.* In S. Staab and R. Studer, editors, Handbook on Ontologies in Inf. Sys., pages 151--172. Springer, 2004

[18] Haarslev, Volker, Möller, Ralf, *RACER System Description.* Proceedings of International Joint Conference on Automated Reasoning, IJCAR'2001, R. Goré, A.

Leitsch, T. Nipkow (Eds.), June 18-23, 2001, Siena, Italy, Springer-Verlag, Berlin,pp. 701-705.

[19] Haarslev Volker, Möller Ralf, Van Der Straeten R., and Wessel M. Extended query facilities for Racer and an application to software-engineering problems. In Proceedings of the International Workshop on Description Logics (DL-2004), Whistler, BC, Canada, June 2004

[20] Haase, Peter and Sure, York. *D3.1.2 Incremental Ontology Evolution-Evaluation.* www.aifb.uni-karlsruhe.de/WBS/pha/publications/, Nov. 2, 2005

[21] Hameed, Adil, Sleeman, Derek, Preece, Alun. *OntoManager: A Workbench environment to facilitate Ontology Management and Interoperability.* Proceedings of the EON-2002, Workshop on Evaluation of Ontology-based Tools at the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW-2002), pages 74-78, September 30-October 4, 2002, Sigüenza, Spain.

[22] Hartmann, Jens, Spyns, Peter, Giboin, Alain, Maynard, Diana, Cuel, Roberta, Suarez-Figueroa, Mari Carmen, Sure, York. *D1.2.3 Methods for ontology evaluation*, www.starlab.vub.ac.be/research/projects/knowledgeweb/, 2004

[23] Hartmann, Jens, Spyns, Peter, Giboin, Alain, Maynard, Diana, Cuel, Roberta, Suárez-Figueroa, Mari Carmen, Sure, York. *D1.2.3 Methods for ontology evaluation*, www.starlab.vub.ac.be/research/projects/knowledgeweb/, 2005

[24] Horrocks, Ian, Patel-Schneider, Peter F., and Harmelen, Frank van. *From SHIQ and RDF to OWL: the making of a web ontology language.* J. Web Sem., 1(1):7–26, 2003.

[25] Lassila O. and Swick R.R. Resource description framework (RDF) model and syntax specification. recommendation, W3C, February 1999

[26] Lozano-Tello, Adolfo, Gómez-Pérez, Asunción. *Ontometric: A method to choose the appropriate ontology*. J. Datab. Mgmt., 15(2):1–18 (2004).

[27] Maedche, Alexander, Motik, Boris, Stojanovic, Ljiljana, Studer, Rudi, Volz, Raphael. *Ontologies for Enterprise Knowledge Management*, IEEE Intelligent System, pp. 26-34, March/April 2003.

[28] Maedche, Alexander, Staab, Steffen, *Measuring similarity between ontologies*. In Proceedings of the European Conference on Knowledge Engineering and Knowledge Management (EKAW), pp. 251--263. Springer Verlag, 2002.

[29] Miller, George A. *Nouns in wordnet: a lexical inheritance system*. International Journal of Lexicography, 3(4):245-264, 1990.

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

[31] Noy, Natalya Fridman and McGuinness, Deborah L. *A Guide to Creating Your First Ontology*, http://protege.stanford.edu/publications/ontology_development/ ontology101-noy-mcguinness.html

[32] Noy N. F., Sintek M., Decker S., Crubezy M., Fergerson R. W., and Musen M. A. Creating semantic web contents with Protege-2000. IEEE Intelligent Systems, 16(2):60–71, 2001

[33] OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/, Feb.10, 2004

[34] Porter, the native, graphical user interface for RacerPro, http://www.racer-systems.c om/products/porter.phtml

[35] Porzel, Robert, Malaka, Rainer. *A task-based approach for ontology evaluation.* ECAI 2004 Workshop Ont. Learning and Population.

[36] RacerPro user's guide Dec. 2005. http://www.racer-systems.com

[37] Shaban-Nejad A., Baker C. J. O., Butler G., Haarslev V. *The FungalWeb Ontology: Semantic Web Application for Fungal Genomic.* 1st Canadian Semantic Web Interest Group Meeting (SWIG'04) , Montreal, Quebec, Canada (2004).

[38] Sleeman, Derek and Reul, Quentin. *CleanONTO: Evaluating Taxonomic Relationships in Ontologies.* 4th International EON Workshop, Edinburgh International Conference Center, Edinburgh, United Kingdom, May 22nd, 2006

[39] Smith, Barry, Williams, Jennifer and Schulze-Kremer, Steffen. *The Ontology of the Gene Ontology.* AMIA Symposium 2003.

[40] The DARPA Agent Markup Language Homepage http://www.daml.org/

[41] Witten, Ian, Frank, Eibe. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann, 1999.

[42] WordNet, http://wordnet.princeton.edu/

[43] W3C, http://www.w3.org/TR/owl-ref/

Appendix

A1. Racer file

A Racer file is a Racer-based ontology-description using nRQL from the ABox query language perspective. Figure A-1 illustrates the *'family.racer'* file.

```
(in-knowledge-base family smith-family)

(signature :atomic-concepts (human person female male woman man
                             parent mother father
                             grandmother aunt uncle
                             sister brother)
       :roles ((has-descendant :transitive t)
            (has-child :parent has-descendant)
            has-sibling
            (has-sister :parent has-sibling)
            (has-brother :parent has-sibling)
            (has-gender :feature t))
       :individuals (alice betty charles doris eve))

(implies *top* (all has-child person))
(implies (some has-child *top*) parent)

(implies (some has-sibling *top*) (or sister brother))
(implies *top* (all has-sibling (or sister brother)))
(implies *top* (all has-sister (some has-gender female)))
(implies *top* (all has-brother (some has-gender male)))

(implies person (and human (some has-gender (or female male))))
(disjoint female male)
(implies woman (and person (some has-gender female)))
(implies man (and person (some has-gender male)))

(equivalent parent (and person (some has-child person)))
(equivalent mother (and woman parent))
(equivalent father (and man parent))

(equivalent grandmother
          (and mother
                (some has-child
                    (some has-child person))))
(equivalent aunt (and woman (some has-sibling parent)))
(equivalent uncle (and man (some has-sibling parent)))

(equivalent brother (and man (some has-sibling person)))
(equivalent sister (and woman (some has-sibling person)))

(instance alice mother)
(related alice betty has-child)
(related alice charles has-child)
```

```
(instance betty mother)
(related betty doris has-child)
(related betty eve has-child)

(instance charles brother)
(related charles betty has-sibling)
(instance charles (at-most 1 has-sibling))

(related doris eve has-sister)

(related eve doris has-sister)
#|
(concept-subsumes? brother uncle)

(concept-ancestors mother)

(concept-descendants man)

(all-transitive-roles)

(individual-instance? doris woman)

(individual-types eve)

(individual-fillers alice has-descendant)

(individual-direct-types eve)

(concept-instances sister)

|#
```

Figure A-1 Racer file format of '*family*' ontology

A2. Related Racer commands

We only list Racer commands used to implement the OntoKBEval:

- **(all-atomic-concepts)**: to get all the concepts in the current TBox
- **(all-individuals)**: to get all the individuals in the current ABox
- **(all-role-assertions)**: to get all assertions in the current TBox
- **(all-roles)**:to get all roles in the current TBox
- **(classify-tbox)**: to classify the whole TBox

- **(concept-children** *concept_term***)**: to get the direct subsumees of the indicated concept in the TBox
- **(concept-equivalent?** *concept_term1, concept_term2***)**: to check if the two indicated concepts are equivalent in the TBox
- **(concept-instances** *concept_term***)**: to get individuals, which are the instances of the indicated concept
- **(concept-parents** *concept_term***)**: to get the direct subsumers of the indicated concept in the TBox
- **(owl-read-file** *path_of_ontology_file***)**: to load an ontology file by RacerPro
- **(realize-abox)**: to check the consistency of ABox to prepare assertion computations
- **(related-individuals** *role_term***)**: to get role assertions related to the indicated role
- **(role-children** *role_term***)**: to get the direct subsumees of the indicated role in the TBox
- **(role-equivalent?** *subsuming_ role_term, subsumed_ role_term***)**: to check if the two indicated roles are equivalent in the TBox
- **(role-inverse** *role_term***)**: to get the inverse of the indicated role
- **(role-parents** *role_term***)**: to get the direct subsumers of the indicated role in the TBox
- **(symmetric?** *role_term***)**: to check if the indicated role is symmetric in the TBox
- **(transitive?** *role_term***)**: to check if the indicated role is a transitive role in the Tbox