A Tool for Storing OWL Using Database Technology

Maria del Mar Roldan-Garcia and Jose F. Aldana-Montes

University of Malaga, Computer Languages and Computing Science Department
Malaga 29071, Spain,
(mmar,jfam)@lcc.uma.es,
WWW home page: http://khaos.uma.es

Abstract. This paper presents a tool that helps developers to design and implement storage models for OWL ontologies. This tool uses xml both to represent the knowledge given by the ontology, and define the rules that generate the storage model. It presents a graphical interface that helps developers to use it. The storage models created by this tool will be used in our research studies.

1 Introduction

OWL language [1] is being widely used to define ontologies in the Web. Its xml based syntax together with its correspondence with Description Logics [2], make this language a candidate to be the standard language for defining ontologies that will be used by Semantic Web applications. However, there are still a few tools that allow us to manipulate, store and query ontologies defined using this language. Description logic based tools, like FaCT [3] or RACER [4] allow us to query simple OWL ontologies, but they do not deal with large amounts of information, and their results can only be applied to very small knowledge bases (with a small number of instances) and these are not the knowledge bases we expect to find in the Semantic Web. In consequence, reasoning algorithms are not scalable and are usually main memory oriented algorithms. Querying and reasoning on instances of ontologies will make the Semantic Web useful.

On the other hand, the database research community has successfully developed a wide theory corpus and a mature and efficient technology to deal with large and persistent amounts of information. Due to the size of the Semantic Web ontologies (very large ontologies with very large amount of instances), we think that it will be necessary to use database technology in order to provide persistence to the knowledge described by the ontologies, and scalability to the queries and reasoning on this knowledge.

Unfortunately, the application of database technology to reason with instances of OWL ontologies is not a trivial matter. Databases work with a closed-world assumption, while ontology systems apply an open-world assumption. In a database, instances are accepted only if they fully comply with the definitions and constraints stated in the schema, while ontology systems accept instances as long as they do not explicitly contradict the knowledge already in the ontology, without requiring that all expected data be present. This must be taken into account in order to implement the reasoning mechanisms using database technology.

Our research objective is to study different storage models for OWL ontologies. Efficient Storage models for OWL ontologies are necessary for the future of the Semantic Web. The complex reasoning mechanisms that should be implemented for the Semantic Web applications will need optimized storage model in order to be efficient and scalable. Therefore, we must define knowledge storage methods beyond a simple correspondence with a database logical schema. That is, it is necessary to define both a physical knowledge representation and access path (indexes) in order to access the knowledge efficiently. We believe this to be an open research issue that should be studied.

To do so, we have developed a tool that helps us to design and implement these storage models, which will be used in our research studies. In this paper we present this tool. The tool is built as a plugin for the Protégé [5] environment. Protégé is a free, open source ontology editor and knowledge-base framework. It is based on Java, is extensible by means of plugins, and provides a foundation for customized knowledge-based applications systems. We exploit the features of Protégé in order to manipulate OWL ontologies easily.

2 OWL Persistent Storage

In the past few years there has been an groing interest in the development of systems for storing large amounts of knowledge in the Semantic Web. Firstly, these systems were oriented to RDF storage [6] [7] [8] [9]. Nowadays, research is oriented to massive OWL storage. Several alternative approaches have been presented, DLDB [10], DLP [11] and Instance Store [12].

DLDB is a knowledge base that extends a relational database management system (RDBMS) with additional capabilities for making OWL inferences. The main objective of this system is to study how description logic reasoning mechanisms can be combined with an RDBMS, in order to support extensional queries on OWL documents. Ontologies are stored using Microsoft Access as RDBMS. Specifically, the system stores RDF in a relational database. Ontologies are stored creating a table for each class or property definition. The class hierarchy is stored in the system using views. The view of a class is defined recursively, and consists of the union of its table and all the views of its direct sub-classes. Therefore, the view of a class includes the instances of that class plus the inferred instances using the taxonomic reasoning mechanism. The sub-property relationship is stored in a similar way. The system is optimized for medium size ontologies (hundreds of classes and properties). The system provides an API, implemented in java, to query the database. It supports conjunctive queries in a format similar to KIF (http://www.cs.umbc.edu/kse/kif/kif101.shtml). The query is translated to SQL and is sent to the database using JDBC. The query is evaluated by the RDBMS which returns the results. The reasoning mechanisms are implemented using the FaCT reasoner coupled to the RDBMS. FaCT only supports Tbox reasoning; therefore DLDB only implements those reasoning mechanisms which can be reduced to concept subsumption (concept and property taxonomy reasoning). These reasoning mechanisms are implemented by pre-computing the class/property hierarchy and storing it in the database using views. Using FaCT, we obtain the sub-classes/sub-properties of a given class/property needed to generate the views. The system does not support other OWL reasoning mechanisms.

DLP is a proposal for combining rules with ontologies in the Semantic Web. It is based on logical databases, since they provide a declarative knowledge representation language and persistent data storage. DLP define a mapping between a description logic (DL) subset and logic programs (LP). This intersection between DL and LP, called DLP (Description Logic Programs), covers RDF schema completely, and part of OWL. In [13] an alternative mapping with less computational complexity is presented. This approach allows for greater representation flexibility. Following the nomenclature defined in [13], we refer to the first correspondence as direct correspondence and to the second one as meta correspondence. Ontologies are described by a subset of OWL, the subset that has a correspondence with LP. Ontologies are stored by translating them to LP. In direct correspondence, each class or property definition corresponds to a rule, and each class or property instance to a fact. In meta correspondence a meta-level is defined. Class and property names are meta-predicates. This meta-level consists of a set of facts representing the ontology content. The storage model, the query language is determined by Datalog. Therefore, the reasoning mechanisms are the ones translatable to Datalog rules.

Instance Store (IS) is an approach to a restricted form of Abox reasoning that combines a description logic reasoner with a database. The Instance Store can only deal with free-role Aboxes, ie Aboxes that do not contain any axioms asserting role relationships between pairs of individuals. Ontologies are described using OWL. Instance Store only offers persistence to the Abox. Abox assertions are stored in a relational database. An identifier (ids) is assigned to each description (concept) and a table stores individuals and the ids of their associated description. Another table contains description ids and all the primitive concepts in the ontology which subsume them. The primitive concepts which are equivalent to, parent of and child of a given description are also stored in a table. Instance Store provides an API written in Java. This API contains a retrieval method that retrieves all the instances of a given concept. The query is translated to SQL and is sent to the relational database. Instance Store does not provide Tbox reasoning mechanisms. That is, it is not possible to reason about the structure of the ontology. The only Abox mechanism is, as we said above, instance retrieval.

If we observe the storage model, all systems choose a database which will ensure the persistence of the knowledge, and the scalability to the queries on this knowledge. However, the logical models that represent the ontologies are not completely refined, and are usually naive representations of the Tbox. Furthermore, no systems take into account the physical representation of the knowledge base, and thus do not choose the best storage structures or an efficient access path (indexes).

Our proposal is not a system to store OWL ontologies. It is a tool for helping developers to create a storage model for OWL ontologies. The tool assists you to define how your model will represent the different OWL ontology elements in the chosen storage model.

2.1 A tool for Storing OWL Ontologies

The development of tools for storing and querying ontologies is a field currently under investigation. The necessity of new advanced reasoning mechanisms that allow us to infer knowledge to be used in Semantic Web applications entails the development of efficient storage models in order to implement these reasoning mechanisms efficiently.

Our main research objective is that reasoning/querying in a knowledge base may be scalable and efficient. We want to fulfil these requirements not in a general case but in the Semantic Web environment. We believe that, in this context, we will find a fairly large amount of instances and we are convinced that, in the Semantic Web, not only reasoning on concepts is necessary, but also reasoning at the instance level and efficient instance retrieval. We would also like to allow queries and even more the combination of reasoning and querying procedures. In order to achieve this, we need to study different storage models for OWL ontologies. We believe that a specific physical representation of the instances of the ontology, and an ad hoc index structures will be necessary. Both representation and index will depend on the expressiveness of the query language and the reasoning implemented. Therefore, the solution will not be unique, but rather a trade-off between expressiveness and efficiency.

In this paper we present a tool that helps us to generate different (physical) storage models, which will be used in our research studies. The tool is built as a plugin for the Protégé tool. We exploit the features of Protégé in order to manipulate OWL ontologies easily. Figure 1 shows the architecture of the tool. The storage model generation has two phases. First, the OWL ontology is parsed in order to obtain all the information it contains. This process generates several XML files. We have chosen XML because programming with Java and XML is very easy. The use of RDF or RFDS could be also an option, but there are more tools for managing XML than RDF. We will consider changing to RDF in the next versions of the tool.

In the second phase, the ontology information stored in these XML files is shown to the user in a graphical way, using the Protégé interface. The storage model is defined by means of several configuration files. These are also XML files. The model developer has to create these files, which contain the rules defining a concrete storage model. The storage model is shown graphically together with the ontology. The developer selects the storage rules that he/she wants to apply. Finally, using all this information the particular storage model for the Data Base Management System (DBMS) selected is generated.

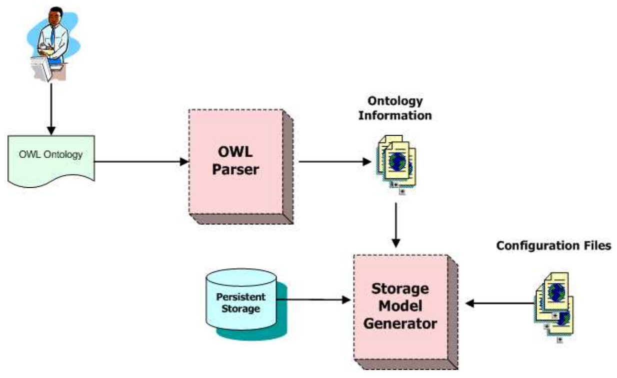


Fig. 1. Architecture of the Tool

OWL Ontologies Parser. This component uses the Protégé API to generate four XML documents for each OWL ontology. These XML files are:

- **classes.xml**: Stores all the information about named classes defined by the ontology. This information includes name, superclasses, subclasses and properties.
- **properties.xml**: Stores all the information about properties defined by the ontology. This information includes name, type, domain, range, whether the property is transitive or symmetric and whether the property has an inverse property.
- **restrictions.xml**: Stores all the information about restrictions defined by the ontology. This information includes name, type, restricted class, etc.
- **instances.xml**: Stores all the information about instances defined by the ontology. This information includes name, type, properties, etc.

Storage Model generator. This component shows the information defined in the configuration files. The developers chose which rules they want to apply and the storage model is generated. Both rules and implementation of these rules are defined in a configuration file. We define three kinds of rules:

- **Default rules**. By applying these rules a default storage model is generated. These rules must be applied all together.
- **Generic rules**. These rules are applied to the default storage model in order to modify it. They are defined for all elements in the ontology that fulfil the rule antecedent. For example, all classes, all properties, etc.

– **Parametric rules**. These rules also are applied to the default storage model. However, they are defined for a specific element of the ontology, which must be specified by the developer before the rule application.

Configuration files. All information about the storage model is defined using several configurations files. These files are xml files. The tool allows us to modify the files using the graphical interface. It is also possible to create a new storage model and all its associated files. The Models.xml file contains all the possible storage models that are defined. It stores the model's name, and the name of the rest of the files that define the model. These files define the storage model rules and the implementation of each rule in a specific DBMS. The same storage model can be implemented using different DBMS. Furthermore, it is really easy to modify an existing storage model. The only we have to do is to modify the rules in the configuration files. This allow us to create easily two physical implementations for the same storage model in order to study the performance of a reasoning mechanism using both implementations.

The tool also provides a component that implements the necessary functions which recover the information about the ontology by accessing the xml files that represent it. For example, the name of the classes or properties defined by the ontology. This component can be used by all developers that want to define a new storage model, due to this information being necessary independently of the desired storage model. These functions will be used for implementing the rules.

3 Storing OWL Ontologies in Relational Databases

In order to demonstrate our tool, we have developed a model that stores OWL ontologies in relational databases. Specifically, it uses Oracle as DBMS. It is a simple example to illustrate the use of the tool. In this chapter we will show briefly the configuration files for this particular model. The aim of this section is not to present a new storage model but to present how, and how easy, it is to use and to configure our tool.

The default storage model is based on the different proposals for storing RDF in relational databases [6] [7]. We create a table for each class and for each property. A table for storing the classes and properties' name is also created. Another table stores each class together with its subclasses. The features of the properties, such as domain, range, if it is transitive, etc. are also stored using a table. All these tables are heaps (by default). Each instance is stored in its corresponding table as a tuple.

The xml syntax to specify the default rule that stores each class in a table is shown in figure 2. A default rule consists of a name, an antecedent, a consequent and a text. When the developer selects a rule in the graphical interface, the content of the text tag is shown in order to give him or her information about what the rule does.

Each default rule is implemented in Oracle. The implementation file specifies how each rule must be implemented. For implementing the previous default rule

```
<rule>
        <name> default1 </name>
        <antecedent> class </antecedent>
        <consequent> heap </consequent>
        <text> stores each class in a heap </text>
</rule>
```

Fig. 2. Default rule example

we used the *get-classes-name* function to recover all the classes defined by the ontology, and for each class we execute the SQL sentence defined in figure 3, where classname is each of the names that the function returns.

```
create table classname (
      id varchar(100)
)
```

Fig. 3. SQL code for the default rule of Figure 2

Once the default storage model is created, we can apply generic and parametric rules. In our case, we had defined generic rules that create indexes for some tables. For example, the rule in figure 4 create an index for the range column of the table that represents a property, if there is a restriction defined for this property. A generic rule has the same structure as a default rule.

```
<rule>
        <nombre> rule1 </nombre>
        <antecedent> restriction </antecedent>
        <consequent> index for the range </consequent>
        <text> creates an index for the range column of the table that represent the
                restricted property
        </text>
</rule>
```

Fig. 4. Generic rule example

Finally, we defined some parametric rules. These rules are applied to a specific element of the storage model. Thus, the structure of these rules is different from the rest of the rules, because it is necessary to specify the parameters. When the developer chooses one of these rules, the values for the parameters must be given. This is also done using the graphical interface. Figures 5 and 6 show two parametric rules. The first one creates a Btree index for the specified column. The second one inserts the specified table in the specified cluster.

```
<rule>
        <name> btreerule </name>
        <antecedent> table </antecedent>
        <consequent> btree on column</consequent>
        <parameter> table_name </parameter>
        <parameter> column_name </parameter>
        <text> creates a Btree index on the table column  </text>
</rule>
```

Fig. 5. Parametric rule example

```
<rule>
        <name> clusterrule </name>
        <antecedent> table </antecedent>
        <consequent> insert into cluster </consequent>
        <parameter> table_name </parameter>
        <parameter> column_name </parameter>
        <parameter> cluster_name </parameter>
        <text> inserts the table into the cluster using the specified column  </text>
</rule>
```

Fig. 6. Parametric rule example

As previously mentioned, the graphical interface allows us to modify, insert and delete new rules in an existing model, and also create a new storage model from scratch. In figure 7 the graphical interface of our tool is shown. This interface is currently in Spanish, but it is being translated to English.

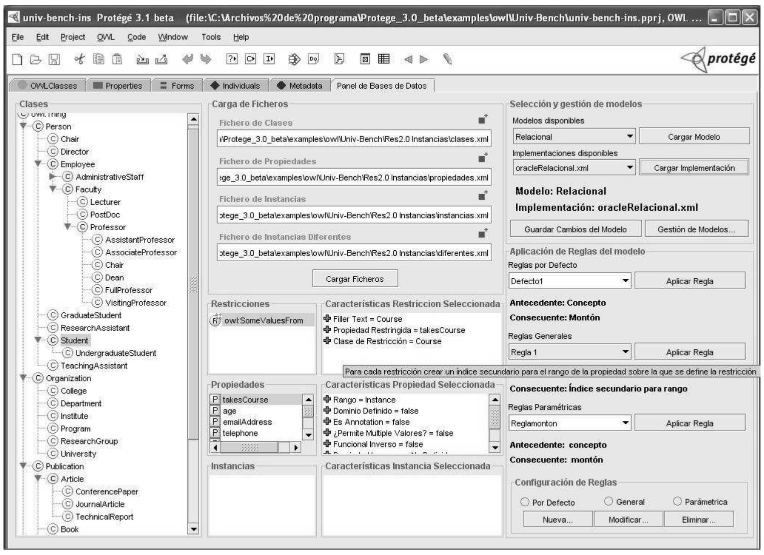
4 Conclusions and Future Work

In this paper, we present a tool that helps developers to design and implement storage models for OWL ontologies. This tool uses xml both to represent the knowledge given by the ontology, and to define the rules that generate the storage model. Using this tool it is possible to store the same OWL ontology using different approaches, which allows us to compare them regarding the reasoning mechanisms performance. We also present an example of how to use this tool for storing an OWL ontology in a relational database.

We are currently developing a new plugin that allows us to access the ontology stored using our tool, in order to query and reason with it. This new plugin will provide a graphical interface to create the queries/reasoning mechanisms, and to execute them. It will also use xml files to be configured.

5 Acknowledgements

This work has been partially supported by the MCyT project TIC2002-04586-C04-04.



univ-bench-ins Protégé 3.1 beta (file:\C:\Archivos%20de%20programa\Protege_3.0_beta\examples\owl\Univ-Bench\univ-bench-ins.pprj, OWL ...

owl:Thing
- Person
  - Chair
  - Director
  - Employee
    - AdministrativeStaff
    - Faculty
      - Lecturer
      - PostDoc
      - Professor
        - AssistantProfessor
        - AssociateProfessor
        - Chair
        - Dean
        - FullProfessor
        - VisitingProfessor
  - GraduateStudent
  - ResearchAssistant
  - Student
    - UndergraduateStudent
  - TeachingAssistant
- Organization
  - College
  - Department
  - Institute
  - Program
  - ResearchGroup
  - University
- Publication
  - Article
    - ConferencePaper
    - JournalArticle
    - TechnicalReport
  - Book

Fichero de Clases: \Protege_3.0_beta\examples\owl\Univ-Bench\Res2.0 Instancias\clases.xml
Fichero de Propiedades: ge_3.0_beta\examples\owl\Univ-Bench\Res2.0 Instancias\propiedades.xml
Fichero de Instancias: otege_3.0_beta\examples\owl\Univ-Bench\Res2.0 Instancias\instancias.xml
Fichero de Instancias Diferentes: otege_3.0_beta\examples\owl\Univ-Bench\Res2.0 Instancias\diferentes.xml

Restricciones: owl:SomeValuesFrom

- Filler Text = Course
- Propiedad Restringida = takesCourse
- Clase de Restricción = Course

Para cada restricción crear un índice secundario para el rango de la propiedad sobre la que se define la restricción

Propiedades: takesCourse, age, emailAddress, telephone

- Rango = Instance
- Dominio Definido = false
- Es Annotation = false
- ¿Permite Multiple Valores? = false
- Funcional Inverso = false

Modelos disponibles: Relacional
Implementaciones disponibles: oracleRelacional.xml
Modelo: Relacional
Implementación: oracleRelacional.xml

Reglas por Defecto: Defecto1
Antecedente: Concepto
Consecuente: Montón

Reglas Generales: Regla 1
Consecuente: Índice secundario para rango

Reglas Paramétricas: Reglamonton
Antecedente: concepto
Consecuente: montón

Fig. 7. Tool graphical interface

References

1. OWL Web Ontology Language Reference. W3C Working Draft, 10 February 2004. http://www.w3.org/TR/owl-ref/, 2004.
2. A. Borgida, M. Lenzerini, and R. Rosati. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2003.
3. Horrocks, I. The FaCT system. International Conference Tableaux '98, number 1397 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 1998.
4. Haarslev, V., Möller, R. RACER System Description. Proceedings of International Joint Conference on Automated Reasoning, IJCAR'2001, Springer-Verlag, 2001.
5. The Protégé Ontology Editor and Knowledge Base Acquisition System. http://protege.stanford.edu/
6. Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D., Trolle, K. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. 2nd Internacional Workshop on the Semantic Web (SemWeb'01).
7. Broekstra, J., Kampman, A., Harmelen, F. (2002). Sesame: a Generis Architecture for Storing and Querying RDF and RDF Schema. 1st Internacional Semantic Web Conference (ISWC2002).
8. B. McBride. Jena: Implementing the RDF Model and Syntax Specification. Steffen Staab et al (eds.): Proceedings of the second international workshop on Semantic Web. SemWeb2001.
9. KAON. The Karlsruhe Ontology and Semantic Web Framework. Developer's Guide for KAON 1.2.7. January 2004. http://km.aifb.uni-karlsruhe.de/kaon2/Members/rvo/KAON-Dev-Guide.pdf
10. Pan, Z. and Heflin, J. DLDB: Extending Relational Databases to Support Semantic Web Queries. In Workshop on Practical and Scaleable Semantic Web Systms, ISWC 2003.

11. Grosof, B.N., Horrocks, I., Volz, R., and Decker, S. Description Logic Programmms: Combining Logic Programms with Description Logic. In Proceedings of the 12th International World Wide Web Conference. 2003.
12. Horrocks, I., Li, L., Turi, D., Bechhofer, S.. The Instance Store: Description Logic Reasoning with Large Numbers of Individuals. 2004.
13. Weithoener, T., Liebig, T., Specht, G. Storing and Querying Ontologies in Logic Databases. The first International Workshop on Semantic Web and Databases. VLDB'2003.