Storing OWL Ontologies in SQL3 Object-Relational Databases

IRINA ASTROVA, AHTO KALJA
Institute of Cybernetics
Tallinn University of Technology
Akadeemia tee 21, 12618 Tallinn
ESTONIA

Abstract: When a large amount of data is stored in OWL files, it is not efficient to maintain and query those data. The OWL syntax is based on XML, which is a meta-markup language. Thus, it is suitable for data description and data exchange, rather than for data storage and data management. Furthermore, enabling multiple users to work with the same ontology in parallel and make modifications mandates the use of databases (e.g. object-relational) for storing ontologies. Therefore, this paper proposes to extract data from OWL files and store those data in object-relational databases. Object-relational databases are good for data storage and data management. But OWL is semantically richer than SQL3. As a result, some of the semantics captured in ontologies can be lost when ontologies are stored in object-relational databases. In addition, some of the structures of ontologies can be lost. E.g. a class in an ontology can be stored as it were an atomic undifferentiated piece of data in a single CLOB column in an object-relational database. When the ontology is stored in such a way, the query and scalability features of the object-relational database cannot be fully exploited to access the data.

Key-Words: Ontologies, Object-relational databases, OWL, SQL3

1 Introduction
There are two basic techniques for storing ontologies [1]. The first technique is to use file systems for storing ontologies in OWL files. The main problem with this technique is that file systems do not provide scalability, sharability, or any query facility.

The second technique (that we follow) is to use database management systems for storing ontologies in databases. The main problem with this technique is that database management systems require that an ontology should have a fixed structure, which cannot be guaranteed as ontologies are often built in a distributed way. This means, for example, that one user may define a document as having a name, but not foresee a date of arrival. This will not stop, however, another user from adding a data type property dateOfArrival to a class Document.

There are several options for storing ontologies in databases; e.g. relational, object or object-relational. Storing ontologies in relational databases is less straightforward than storing ontologies in object or object-relational databases, because relational database management systems do not provide support for many constructs of ontologies; e.g. class inheritance, object properties and cardinalities. Even though some of these constructs can be mapped to relational databases, the efforts can prove laborious and time-consuming.

2 Motivation
There are four main reasons for storing ontologies in object-relational databases:

1. **Legacy data:** When stored in object-relational databases, ontologies can interoperate with data that already exist in the object-relational databases.
2. **Legacy applications:** When stored in object-relational databases, ontologies can be accessed from within existing object-relational database applications.
3. **Efficient querying:** When stored in object-relational databases, ontologies can be efficiently queried using SQL3.
4. **Large scale ontologies:** The ability of object-relational databases to store a large amount of data proves that the object-relational databases are also suitable for storing ontologies that can contain millions of instances.

A prerequisite for this storing is transformation of ontologies to object-relational databases, which is the objective of this paper.

3 Related Work
A majority of the related work has been done in transformation of ontologies to relational databases (see e.g. [2], [3] and [4]). Since object-relational databases have evolved from relational databases, naturally, all the mappings identified for relational databases can also be applied to object-relational databases.

The objective of this paper is, however, to present the different alternatives object-relational databases offer compared to relational databases.

4 Ontologies vs. Object-Relational Databases

Although ontologies and object-relational databases are two different things, and it seems like comparing apples to tomatoes, we relate ontologies to object-relational databases in order to better identify the mappings between the two (see Table 1).

Let’s think for a moment of an ontology as an object-relational database. The ontology has a structure (classes and properties) and may also have data (instances). The term “knowledge base” is often used as a synonym to denote an instantiated ontology (that has both the structure and the data). The structure of the ontology roughly corresponds to the schema of the object-relational database, whereas the data in the ontology roughly correspond to rows in the object-relational database. Hence, it is not unjustified to compare ontologies to object-relational databases.

Table 1. Comparison of ontology to object-relational database

<table>
<thead>
<tr>
<th></th>
<th>Ontology</th>
<th>Object-relational database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic concepts</td>
<td>Classes, data type properties, object properties, instances and restrictions</td>
<td>Object types, tables, columns, references, rows and constraints</td>
</tr>
<tr>
<td>Inheritance</td>
<td>Supported (through inheritance of classes and properties)</td>
<td>Supported (through inheritance of object types and tables)</td>
</tr>
<tr>
<td>Binary relationships</td>
<td>Supported (through object properties)</td>
<td>Supported (through foreign keys and references)</td>
</tr>
<tr>
<td>Bidirectional binary relationships</td>
<td>Supported (through inverse object properties)</td>
<td>Not supported (references allow for unidirectional navigation only)</td>
</tr>
<tr>
<td>Ternary and higher degree relationships</td>
<td>Not supported (a separate class is needed)</td>
<td>Not supported (a separate table is needed)</td>
</tr>
<tr>
<td>Data definition and manipulation language</td>
<td>OWL (DL, Light and Full)</td>
<td>SQL3 (none of object-relational database management systems today conforms to the standard SQL3 completely)</td>
</tr>
<tr>
<td>Query language</td>
<td>No standard query language; various options exist; e.g. RDQL, OQL, etc.</td>
<td>SQL3</td>
</tr>
</tbody>
</table>

There are also interesting relationships between ontology languages (e.g. OWL [5]) and object-relational database languages (e.g. SQL3 [6]). We consider one of them: language expressivity [7].

There is much overlap in language expressivity, including inheritance, relationships, collections, etc.
E.g. classes in ontologies map to object types in object-relational databases, data type properties map to columns, etc. For both OWL and SQL3 there is a vocabulary of terms with natural language definitions. These definitions are inline comments in ontologies but they are stored in separate data dictionaries for object-relational databases. Arguably, there is little or no obvious essential difference between OWL and SQL3. They are similar. The key difference is in language expressivity that varies in importance.

Ontologies are primarily used for interoperability: one or more parties commit to using the terms with their declared meaning. The primary use of object-relational databases is, however, to structure a set of data for querying. This difference impacts on the role of constraints.

In ontologies, constraints are called restrictions. Their main purpose is to express machine-readable meaning to support automated reasoning. This reasoning can also be used to ensure integrity of instances in a knowledge base. For object-relational database management systems, the main purpose of constraints is to ensure the integrity of data in object-relational databases. Constraints can also be used to optimize queries against object-relational databases and help humans infer the meaning of the terms. Cardinalities are important kinds of constraints that have a specific use in most object-relational database management systems; e.g. they are used to get a foreign key in the right direction and to ensure that an extra table is built for a many-to-many relationship. These constraints do express meaning, but this is of secondary importance. The main role of cardinalities in ontologies is, however, to express meaning and only then ensure consistency of data.

The ability to store ontologies depends on the capabilities of the object-relational database management system. These capabilities vary from vendor to vendor and version to version. While some of the capabilities are common to most object-relational database management systems, others are not. Even though the mapping rules are described below in the context of a particular object-relational database management system (namely, ORACLE 9i), they are general enough and not limited to the particular object-relational database management system.

5.1 Example
We use a simple example – an ontology for a virtual library – throughout this paper to demonstrate the ideas of mappings with concrete examples and code fragments. The ontology is targeted to maintain bibliographical information on documents; e.g. authors, abstracts, keywords, etc. The documents can be either printed or online.

Fig. 1 shows the ontology and the object-relational database that is produced from the ontology using the mapping rules.

```xml
<owl:Class rdf:ID="Document"/>
<owl:DatatypeProperty rdf:ID="keywords">
    <rdfs:domain rdf:resource="#Document"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:Restriction>
    <owl:onProperty rdf:resource="#keywords"/>
    <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">6</owl:maxCardinality>
</owl:Restriction>
<owl:Class rdf:ID="Author"/>
<owl:ObjectProperty rdf:ID="writes">
    <rdfs:domain rdf:resource="#Author"/>
    <rdfs:range rdf:resource="#Document"/>
</owl:ObjectProperty>
```

5 Transformation of Ontologies to Object-Relational Databases
This transformation is based on a set of rules that specify how to map each construct in an ontology to a corresponding construct in an object-relational database. Our previous work (see [4]) provides a concise overview of transformation of an ontology to a relational database. Thus, there is no point repeating here how to map data type properties, XML Schema data types, value restrictions and enumerated data types. For the purpose of this paper, we look at how to map classes, class inheritance, cardinalities and object properties. Their mappings are described in the following subsections and examples are provided.
5.3 Mapping Class Inheritance
Class inheritance maps to object type inheritance. As an example, take a class PrintedDocument in Fig. 1. This class is defined as a subclass of Document. Therefore, a corresponding object type is defined as a subtype of Document_t.

There is a bit of a problem with object type inheritance, however. Even though we can define multiple supertypes in ORACLE, the semantics of such a definition are very weak and do not address even basic problems such as duplicate column names in different supertypes.

5.4 Mapping Cardinalities
When a property is defined with a maximum cardinality greater than 1, it maps to a varying array; i.e. VARRAY(n) where n is the maximum number of values the property can have.

As an example, take a data type property keywords in Fig. 1. This property has a maximum cardinality of 6 for instances in a class Document, meaning that a document can have at most 6 keywords. Therefore, a corresponding column in an object type Document_t maps to VARRAY(6).

There is a bit of a problem with varying arrays, however. A limit on the number of elements in a varying array must be specified. But properties in the ontology can have zero or more values by default, meaning that there is no limit.

5.5 Mapping Object Properties
An object property maps to a reference. This reference is named with the name of the object property.

As an example, take object properties writes and isWrittenBy in Fig. 1 (one is an inverse of another). These two properties represent a many-to-many relationship between authors and documents: each author can write many documents and each document can be written by many authors. The object property writes maps to a varying array of references to Document rows in a table Author, whereas the object property isWrittenBy maps to a varying array of references to Author rows in a table Document. (The triggers will need to ensure that these two arrays stay synchronized, because the object-relational database management system does not keep automatically references up-to-date.)
There is a bit of a problem with references, however. References do not enforce referential integrity. If we delete a row, any references to it remain just as they are. Also, we cannot constrain operations such as DELETE to fail if there are outstanding references to the row. Therefore, if we use references, we should encapsulate the row to which they refer within a set of operations on the references. We can do this through triggers, for example.

6 Conclusion and Future Work
We have proposed an approach to automatic transformation of ontologies to object-relational databases. This approach can be applied to any object-relational database management system that supports the standard SQL3, because the approach does not rely on any SQL3 dialect. The approach can map all constructs of an ontology to an object-relational database, with the exception of those constructs that have no correspondences in the object-relational database (e.g. subproperties.). The approach names the constructs of an object-relational database using the names of ontology constructs (converting the names as appropriate or required by name length restrictions in the object-relational database management system).

The main problem with the approach is the naming strategy, in particular, when an ontology that imports another ontology is transformed into an object-relational database. Unlike OWL, SQL3 does not support namespaces. A simple solution to this problem is to keep class names unique over multiple ontologies (as done in the approach), but a more sophisticated naming strategy needs to be developed in the future.

References: