An Open Source Database Backend for the OWL API and Protégé 4

Timothy Redmond

Stanford Center for Biomedical Informatics Research, Stanford University, USA
tredmond@stanford.edu

Abstract. In this paper we describe the design and implementation of a relational database back-end for the OWL API [3]. The motivation for this work is to allow servers, such as the NCBO BioPortal [4] and WebProtégé[1], to use the OWL API to access several large ontologies at the same time while maintaining a small memory footprint. Our database backed implementation of some key OWL API interfaces allows such a server to use the OWL API to access several large OWL ontologies within a limited memory footprint. This database backend for the OWL API was implemented independently of but in parallel with a very different implementation of a database backend [2, 1] for the OWL API based on Hibernate. This paper will describe the design of the Protégé database backend and discuss some of the key design decisions and differences from the Hibernate-based database backend. We have tested this database backend with the MySql and Postgres databases and it is now being tested on the BioPortal with the MySql database.

The primary purpose of the Protégé OWL API database backend is to support the use of the OWL API on servers that need to be able to load many large ontologies into main memory at the same time. The default implementation of the *OWLOntology* class provided by the Manchester OWL API[2] loads the content of an OWL ontology into several hash maps that live entirely in the applications memory. This implementation is very fast and has been optimized to use a relatively small memory footprint. A large ontology - one million axioms in a 150MB ontology - can be loaded into memory in about 70 seconds and after loading requires approximately 500MB of 32-bit memory. These are very reasonable numbers for such a large ontology for users who are loading one or two ontologies in a desktop applications. However, this setup would not work for ontology servers. The NCBO BioPortal is a repository of over 200 biomedical ontologies that can be accessed through a Web interface or using RESTful Web Services. It is a common situation that the BioPortal server to

[1] http://protegewiki.stanford.edu/wiki/WebProtege
[2] http://owlapi.sourceforge.net/

have fifty ontologies open at the same time several of which may be quite large. Due to resource limitations, it is not feasible for servers such as BioPortal to load a large number of ontologies into memory. The problem is two-fold. First these servers have many demands on their resources. They will quickly run out of resources if they preload all the available ontologies. Second, even if memory would be sufficient for on-demand loading of needed ontologies, the time required to load an ontology is incompatible with a reasonable response time for users. The Protégé database backend allows a developer to very quickly load and access an implementation of the OWL API *OWLOntology* interface for even the largest OWL ontologies being considered.

The primary disadvantages of putting ontology data into a database backend are: (1) performance of the initial dump of the ontology into the database, (2) the performance of queries to the database and (3) the complexity of the implementation. As an example of the first problem, the same large ontology that loaded into memory in 70 seconds requires 19 minutes to load into the database format. Once in the database format, the database implementation of the *OWLOntology* interface will be instantly available on demand; all that is needed to populate the database version of the *OWLOntology* interface are a couple of SQL queries. In the case of the servers at Stanford, this shortcoming is acceptable. When a large new ontology is submitted to the BioPortal, the user will not see that ontology immediately show up as ready on the BioPortal page. Instead the ontology undergoes some processing which may take some time. This processing can be done on a separate machine from the main server and therefore will not degrade server processing.

The second problem occurs because database queries are very expensive when compared with their analogous in-memory operations. We often see times just under a millisecond to perform even very simple queries in a database. A primary design goal of the Protégé database backend is to try to improve performance by avoiding SQL queries when the result can be calculated in Java code. To this end, we have decided to perform all serialization and deserialization of OWL axioms in Java code. The serialization of an axiom is stored in the database but never queried or indexed. This means that OWL axioms never need to be pieced together by joining the contents of several different tables in the database. The downside is that a significant amount of time during read operations is spent deserializing representations of axioms. This is a significant difference between our implementation and the Hibernate database backend.

Since the serialized form of the axioms is only retrieved and never examined during SQL queries, the database must hold enough information to rapidly find the right axiom in the database when needed. In addition the database implementation has been optimized for use with the OWL API - other possible uses of the data have not been considered. For example, one of the interfaces that the Protégé database backend needed to implement was *getReferencingAxioms* which retrieves all axioms that have the specific entity in their signature. In order to retrieve these axioms, the Protégé database backend makes use of three tables. The first table contains a list of all the entities in the signature of the OWL ontology, providing a database identifier for the entity, the name (IRI) of the entity and the type of the entity. The second table lists all the axioms in the OWL ontology providing each axiom with a database identifier for the axiom, the serialized form of the axiom and several other properties of the axiom. The third table links the first two tables by indicating which entities appear in which axioms. These three tables can easily be joined together and queried to retrieve, in a single database query, the set of axioms required by the OWL API call. The choice of what data needs to be included in these tables is purely driven by the question of what is needed to efficiently provide the information needed by the OWL API interfaces.

The third issue with a database backend is the added complexity. From a user point of view, the database needs to be installed and the configuration of the database is very important. Then the user needs to figure out how to provide the right connection strings for the database and how to use and protect the database passwords. But for our target applications, web servers that load many ontologies, this complexity already exists and is part of what a web server must deal with.

But there is also additional complexity in the implementation of the database backend itself. Instead of simple lookups in a hash table, the database backend designer has to work with SQL queries and must additionally understand the different datatypes that are used by different databases and differences between supported SQL queries. For example the fact that Postgres supports the "SELECT DISTINCT ON" but MySql does not meant that we needed to provide different versions of the same query for the two databases. In addition, the database backend needs to avoid storing large amounts of data in memory. For example, in the Protégé database backend, a lot of effort went into ensuring that the OWL API call that retrieves all the axioms in an ontology provides a set that is constructed on demand and does not use all the memory that would

be required to load these axioms at the same time. Finally, in-memory calls to Hash tables rarely fail and usually then only because of an out-of-memory condition. In contrast, database queries can fail in numerous ways.

In the area of complexity, we believe that the Hibernate approach probably has a natural advantage over the approach that we took. It provides a layer that protects the developer from low level details. To implement our database backend, we needed to carefully craft over seventy SQL queries. A mistake in any one of these queries may lead to errors or serious performance loss. In addition, some of these queries can easily use features that are available in one database but not in another. In many cases the same queries for the Hibernate implementation will be generated by Hibernate and not the developer.

We are already using the database backend in the BioPortal server and work is in progress to use a database backend in WebProtégé. Most, if not all, of this work will work equally well with either database backend. Since it is easy to switch between the two implementations it is important . So we plan on doing some testing work to compare how the database backends perform both in terms of memory use and in terms of performance. Since the two database backends have very different design approaches it will be interesting to see how their behavior differs. Both database backends are open source available from SVN[3][4]. There is also a proof of concept implementation of a plugin that integrates that database backend into Protégé 4.1.

References

1. Jörg Henss, Joachim Kleb, and Stephan Grimm. A Protégé 4 backend for native owl persistence. 2009 International Protégé Conference, Amsterdam, Netherlands, June 2009.
2. Jörg Henss, Joachim Kleb, Stephan Grimm, and Jürgen Bock. A database backend for OWL. In Rinke Hoeksta and Peter F. Patel-Schneider, editors, *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States, October 23-24, 2009*, volume 529, 2009.
3. Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for working with OWL 2 ontologies. In Rinke Hoekstra and Peter F. Patel-Schneider, editors, *OWLED*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
4. Natalya F. et al Noy. BioPortal: ontologies and integrated data resources at the click of a mouse. *Nucleic Acids Research*, 10.1093/nar/gkp440, 2009.

[3] http://smi-protege.stanford.edu/repos/protege/protege4/libraries/org.protege.owlapi/trunk
[4] https://owldb.svn.sourceforge.net/svnroot/owldb