The OWL Instance Store: System Description

Sean Bechhofer, Ian Horrocks, and Daniele Turi

Information Management Group, School of Computer Science,
The University of Manchester, Manchester, UK
{lastname}@cs.manchester.ac.uk

Abstract. We describe the instance store, a system for reasoning about individuals (i.e., instances of classes) in OWL ontologies. By using a hybrid reasoner/database architecture, our system is able to perform efficient reasoning over large volumes of instance data, as required by many real world applications.

1 Introduction

Ontologies, with their intuitive taxonomic structure and class based semantics, are widely used in domains like bio- and medical-informatics, where there is a tradition in establishing taxonomies of terms. The recent W3C recommendation of *OWL* [8] as the language of choice for web ontologies also underlines the long term vision that ontologies will play a central role in the semantic web. Most importantly, as shown in [9], most of the available OWL ontologies can be captured in *OWL-DL*—a subset of OWL for which highly optimised Description Logic [5] reasoners can be used to support ontology design and deployment.

Unfortunately, existing reasoners (and tools), while successful in dealing with the (relatively small and static) *class* level information in ontologies, fail when presented with the large volumes of *instance* level data often required by realistic applications, hampering the use of reasoning over ontologies beyond the class level.

The system we present—the *instance Store* (*iS*)—addresses this problem using a hybrid database/reasoner architecture: a relational database is used to persist instances, while a class level (i.e. 'TBox') reasoner is used to infer ontological information about the classes they belong to; moreover, part of this ontological information is also persisted in the database. The *iS* only supports a very limited form of reasoning about individuals, i.e., answering instance retrieval queries w.r.t. an ontology and a set of axioms asserting class-instance relationships, and it is clear that from a theoretical point of view this could be reduced to pure TBox reasoning. The *iS* is, however, able to process *much* larger numbers of individuals than it is currently possible using standard Description Logic reasoners. Moreover, this kind of reasoning turns out to be useful in a wide range of applications, in particular those where domain models are used to structure and investigate large data sets.

2 Architecture and Interface

There is a long tradition of coupling databases to knowledge representation systems in order to perform reasoning, most notably the work in [10]. However, in our architecture

R. Nieuwenhuis (Ed.): CADE 2005, LNAI 3632, pp. 177–181, 2005.

```
initialise(database: Database, reasoner: OWLReasoner, ontology: OWLOntology)
addAssertion(instance: URI, class: OWLDescription)
retrieve(query: OWLDescription): Set⟨URI⟩
```

Fig. 1. The *iS* API

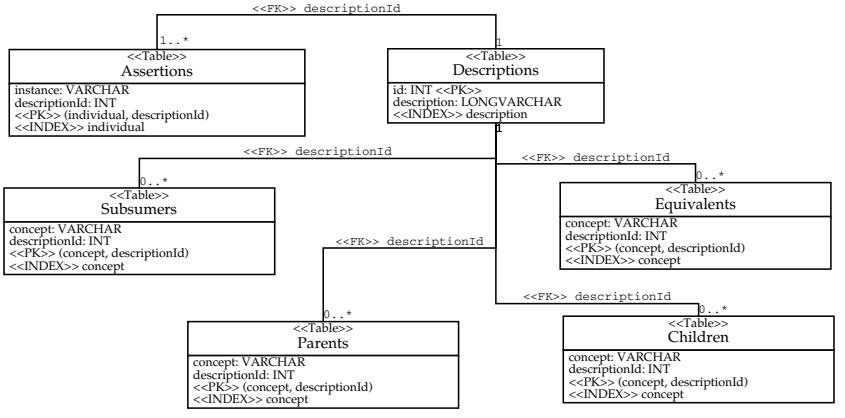

Fig. 2. Database Schema for *iS*

we do not use the standard approach of associating a table (or view) with each class and property. Instead, we have a fixed and relatively simple schema that is independent of the structure of the ontology and of the instance data. The *iS* is, therefore, agnostic about the provenance of data, and uses a new, dedicated database for each ontology (although the schema is always the same).

The basic functionality and the database schema of the *iS* system are illustrated in Figure 1 and Figure 2 respectively. At start-up, the initialise method is called w.r.t. a relational database, an OWL class reasoner, and a class level (i.e., not containing instances) OWL ontology. The method creates the schema for database if needed (ie if the *iS* is new), parses ontology and loads it into the reasoner.

To populate the *iS*, one calls the addAssertion method repeatedly. Each assertion states that instance (a URI) belongs to class, which is an arbitrary OWL description (not involving other instances). Once one has populated the *iS* with some—possibly millions of—instances, one can query it using the retrieve method. A query again consists of an arbitrary OWL class description, and the result is the set of all instances belonging to the query class.

3 Implementation

We have implemented our system in Java[1]. The communication with the reasoner is implemented using the DIG interface [7]. This allows the *iS* system to be, again, fully

[1] Source code, binaries, GUI, and test suites are publicly available from SourceForge [3].

agnostic of the actual reasoner used; indeed, we have used FaCT [14], Racer [12], and FaCT++ [1] in our applications, sometimes using all of them at different times for the same *iS*. As for the database system, we have used *MySQL*, *Oracle* and *Hypersonic*, accessed either through *JDBC* or through *Hibernate*

The key algorithms in the Java code itself are those for addAssertion and retrieve. Our starting point is the 'semantic indexing' of [15], taking the atomic classes in the ontology as indexing concepts. In order to improve performance we also cache additional information about descriptions: for every description D used in a class-instance assertion or query, we store D in the `Descriptions` table, compute (using a TBox reasoner) the location of D in the class hierarchy, and cache all named (atomic) concepts that

- subsume D (storing them in the `Subsumers` table);
- are equivalent to D (storing them in the `Equivalents` table);
- are parents (direct subsumers) of D (storing them in the `Parents` table); or
- are children (direct subsumees) of D (storing them in the `Children` table).

Caching this information avoids the need to traverse the class hierarchy (and issue many DB queries) when answering instance retrieval queries. With this data in place, the speed of retrieval for a query Q depends on whether:

1. Q is referenced in `Equivalents` ($\Rightarrow$ virtually immediate answer);
2. Q subsumes the conjunction of its parents[2] ($\Rightarrow$ fast answer);
3. there is a set I of individuals, each of which is an instance of *all* of the parents of Q and not an instance of *any* child of Q ($\Rightarrow$ speed of answer depends on size of I).

Note that almost all the reasoning needed in retrieval is performed by means of (single) SQL queries, with the exception of the last case where the reasoner is needed for as many subsumption tests as the size of I. For more details visit the *iS* website [4].

Clearly, the performance gains obtained by caching classified descriptions come at the expenses of maintenance: changes at the class level of the ontology require costly updates to the *iS*. The whole system, however, is geared towards scalability and fast retrieval times, and the applications below demonstrate that this is useful in realistic scenarios.

4 Applications and Performance

The first application we describe illustrates the performance of the *iS* w.r.t. a real world problem with more that half a million instances. The Gene Ontology (*GO*) consortium publishes every month a database [2] of gene products referring to terms in a large (tens of thousands of classes) ontology. The structural simplicity of the ontology (little more than a taxonomy of classes) means that its transitive closure can be precomputed and stored in the database so that, when a client searches for the gene products whose descriptions are subsumed by a set of terms, the answer can be returned without using

[2] Every concept is always subsumed by the conjunction of its parents, hence this effectively checks whether Q is equivalent to the conjunction of its parents.

any reasoner. Together with other functionality provided by the database, this provides biologists with a service which is highly valued and widely used.

To test the *iS*, we mined (the SWISS-PROT fragment of) the Gene Ontology database extracting 653,762 gene product descriptions which we loaded in the *iS* using the addAssertion method (in 23,750 seconds using FaCT++). In our mining we exploited the fact that gene terms form three more or less separate taxonomies of 'processes', 'components' and 'functions'. We therefore added three corresponding new properties (also known as *roles*) to the gene ontology and described gene products using them. For instance, we asserted that *1433 CANAL* is an instance of the class of gene products that **take part in** *intracellular signalling cascade*, are **part of** *chloroplast*, and have the **function** of *protein domain specific binding activity*. (We denote roles in bold and classes in italics.)

This does not take into account annotations and other information present in the GO database, but our aim was simply to test a large set of realistic and interesting data. Extensions in the structure of the ontology (as envisaged in GONG [17]) would allow more complex assertions to be made and more complex queries to be asked .

Our results are very encouraging. We have tested our GO *iS* against various queries formulated by domain experts. Their descriptions are similar in structure to the description of the assertion for the above gene product *1433 CANAL*, i.e. a conjunction of processes, components and functions (each conjunct possibly empty), and the retrieval times range between less than a second and few seconds depending on the factors discussed in Section 3. The queries cover all three cases mentioned in the previous section, thus including run-time calls to the reasoner for subsumption checks.

More bioinformatics applications of the *iS* include its use to guide gene annotation [6] and, more recently, to investigate the structure of data mined from the InterPro database of protein families [16].

We also built another example [11] of *iS* within the proof-of-concept project *MONET*, where mathematical web-services are envisaged to register to a broker using the *iS* to perform service matching. A typical service description specifies the 'GAMS' classification of the service, the problem it solves, input and output formats, the directives it accepts, the software used to implement it, and the algorithm it implements. All this involves several classes and roles in nested conjunctions from an ontology containing thousands of classes interconnected by means of tens of roles. The structural richness of the ontology means that services can then be matched using, e.g., a bibliographic reference to their implemented algorithm. The MONET *iS* contains too few instances for its performance to be significant, however it illustrates the expressivity of our approach.

5 Conclusions and Future Work

The architectural choices made in the implementation of *iS* ensure that we use appropriate technologies for appropriate tasks. It is clear that at some point the reasoner must be used in order to retrieve individuals, but in our approach it is only used when necessary. Databases are well suited to handling large volumes of data and are optimised for the performance of operations such as joins and intersections.

The functionality of the *iS* is limited, but is sufficient to support several interesting applications, and allows us to deal with volumes of instance data that cannot, to the best of our knowledge, be handled by any other reasoner.

In the present *iS*, roles are allowed to appear at class level as in the GO role **take part in**, but no role assertion between instances is allowed, i.e., we cannot assert that instance *x* is related via role **r** to instance *y*. We are currently working on an extension of the *iS* that uses the *precompletion* technique [13] to overcome this limitation (although at the cost of some restrictions on the structure of the ontology).

References

1. FaCT++. `http://owl.man.ac.uk/factplusplus`.
2. Gene Ontology Database. `http://www.godatabase.org/dev/database`.
3. Instance Store API. `http://sourceforge.net/projects/instancestore`.
4. Instance Store website. `http://instancestore.man.ac.uk`.
5. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook — Theory, Implementation and Applications*. Cambridge University Press, 2003.
6. M. Bada, D. Turi, R. McEntire, and R. Stevens. Using Reasoning to Guide Annotation with Gene Ontology Terms in GOAT. *SIGMOD Record (special issue on data engineering for the life sciences)*, June 2004.
7. Sean Bechhofer. The DIG description logic interface: DIG/1.1. In *Proceedings of the 2003 Description Logic Workshop (DL 2003)*, 2003.
8. Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference. Technical Report REC-owl-ref-20040210, The Worldwide Web Consortium, February 2004.
9. Sean Bechhofer and Raphael Volz. Patching Syntax in OWL Ontologies. In *Proceedings of 3rd International Semantic Web Conference (ISWC'04)*, Hiroshima, Japan, 2004.
10. Alexander Borgida and Ronald J. Brachman. Loading data into description reasoners. In *Procs ACM SIGMOD Int'l Conf. on Management of Data*, pages 217–226, 1993.
11. Olga Caprotti, Mike Dewar, and Daniele Turi. Mathematical service matching using Description Logic and OWL. In *Proceedings of 3rd International Conference on Mathematical Knowledge Management (MKM'04)*, volume 3119 of *LNCS*. Springer-Verlag, 2004.
12. V. Haarslev and R. Moller. Description of the RACER system and its applications. In R. Gore, A. Leitsch, and T. Nipkow, editors, *Procs of IJCAR 2001*, volume 2083 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag Inc., 2001.
13. Bernhard Hollunder. Consistency checking reduced to satisfiability of concepts in terminological systems. 18(2–4):133–157, 1996.
14. I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Procs 6th Int'l Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.
15. A. Schmiedel. Semantic indexing based on description logics. In F. Baader, M. Buchheit, M.A. Jeusfeld, and W. Nutt, editors, *Reasoning about structured objects: knowledge representation meets databases. Proceedings of the KI'94 Workshop KRDB'94*, September 1994.
16. K. Wolstencroft, P. Lord, L. Tabernero, A. Brass, and R. Stevens. Intelligent classification of proteins using an ontology. Submitted for publication, 2005.
17. Chris J. Wroe, Robert D. Stevens, Carole A. Goble, and Michael Ashburner. A methodology to migrate the gene ontology to a description logic environment using daml+oil. In *Proceedings of the 8th Pacific Symposium on Biocomputing (PSB)*, Hawaii, January 2003.