The Instance Store: DL Reasoning with Large Numbers of Individuals

Ian Horrocks, Lei Li, Daniele Turi and Sean Bechhofer
University of Manchester, UK
<lastname>@cs.man.ac.uk

Abstract

We present an application – the Instance Store – aimed at solving some of the scalability problems that arise when reasoning with the large numbers of individuals envisaged in the semantic web. The approach uses well-known techniques for reducing description logic reasoning with individuals to reasoning with concepts. Crucial to the implementation is the combination of a description logic terminological reasoner with a traditional relational database. The resulting form of inference, although specialised, is sound and complete and sufficient for several interesting applications. Most importantly, the application scales to sizes (over 100,000s individuals) where all other existing applications fail. This claim is substantiated by a detailed empirical evaluation of the Instance Store in contrast with existing alternative approaches.

Introduction

The Semantic Web [6] aims at making Web resources more accessible to automated processes by adding "semantic annotations"—metadata that describes their content. It is envisaged that the semantics in semantic annotations will be given by ontologies, which will provide a source of precisely defined terms (vocabularies) that are amenable to automated reasoning.

A standard for expressing ontologies in the Semantic Web has already emerged: the ontology language *OWL* [9], which recently became a W3C recommendation. One of the main features of OWL is that there is a direct correspondence between (two of the three "species" of) OWL and *Description Logics* (*DL*s) [19]. This means that DL reasoners can be used to reason about OWL ontologies and about annotations that are instances of concept descriptions formed using terms from an ontology.

Unfortunately, while existing techniques for *TBox* reasoning (i.e., reasoning about the concepts in an ontology) seem able to cope with real world ontologies [18, 14], it is not clear if existing techniques for *ABox* reasoning (i.e., reasoning about the individuals in an ontology) will be able to cope with realistic sets of instance data. This difficulty arises not so much from the computational complexity of ABox reasoning, but from the fact that the number of individuals (e.g., annotations) might be extremely large.

In this paper we describe the *instance Store* (*iS*), an approach to a restricted form of ABox reasoning that combines a DL reasoner with a database. The result is a system that can deal with very large ABoxes, and is able to provide sound and complete answers to instance retrieval queries (i.e., computing all the instances of a given query concept) over such ABoxes.

While *iS* can be highly effective, it does have limitations when compared to a fully fledged DL ABox. In particular, *iS* can only deal with a *role-free* ABox, i.e., an ABox that does not contain any axioms asserting role relationships between pairs of individuals. Although this may seem a rather severe restriction, the functionality provided by *iS* is precisely what is required by many applications, and in particular by applications where ontology based terms are used to describe/annotate and retrieve large numbers of objects. Examples include the use of ontology based vocabulary to describe documents in "publish and subscribe" applications [10], to annotate data in bioinformatics applications [12] and to annotate web resources such as web pages [11] or web service descriptions [20] in Semantic Web applications. Indeed, we have successfully applied *iS* to perform web service discovery [8], to search over the gene ontology [12] and its associated instances (see below), and in an application to guide gene annotation [4].

Using a database in order to support ABox reasoning is certainly not new (see below), but to the best of our knowledge *iS* is the first such system that is general purpose (i.e., can deal with any TBox and role-free ABox without customising the database schema), provides sound and complete reasoning, and places no a-priori restriction on the size of the ABox.

In order to evaluate the design of *iS*, and in particular its ability to provide scalable performance for instance retrieval queries, we have performed a number of experiments using *iS* to search over a large (50,000 concept) gene ontology and its associated very large number (up to 650,000) of individuals – instances of concept descriptions formed using terms from the ontology. In the absence of other specialised ABox reasoners we have compared the performance of *iS* with that of RACER [15] (the only publicly available DL system that supports full ABox reasoning for an expressive DL) and of FaCT [18] (using TBox reasoning to simulate reasoning with a role-free ABox).

Related Work As already mentioned, the idea of supporting DL style reasoning using databases is not new. One example is [7], which can handle DL inference problems by converting them into a collection of SQL queries. This approach is not limited to role-free ABoxes, but the DL language supported is much less expressive, and the database schema must be customised according to the given TBox. Another example is the Parka system [2]. Parka is not limited to role-free ABoxes and can deal with very large ABoxes. However, Parka also supports a much less expressive language, and is not based on standard DL semantics, so it is not really comparable to *iS*. Finally, [21] describes a "semantic indexing" technique that is very similar to the approach used in *iS* except that files and hash tables are used instead of database tables, and optimisations such as the use of equivalence sets are not considered.

1 Instance Store

Description Logics are a family of knowledge representation formalisms evolved from early *frame systems* and *semantic networks*. We assume the reader to be familiar with DLs—see [3] for a detailed discussion of DLs.

An ABox $\mathcal{A}$ is role-free if it contains only axioms of the form $x : C$. We can assume, without loss of generality, that there is exactly one such axiom for each individual as $x : C \sqcup \neg C$ holds in all interpretations, and two axioms $x : C$ and $x : D$ are equivalent to a single

axiom $x : (C \sqcap D)$. It is well known that, for a role-free ABox, instantiation can be reduced to TBox subsumption [16, 22]; i.e., if $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, and $\mathcal{A}$ is role-free, then $\mathcal{K} \models x : D$ iff $x : C \in \mathcal{A}$ and $\mathcal{T} \models C \sqsubseteq D$. Similarly, if $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ and $\mathcal{A}$ is a role-free ABox, then the instances of a concept D could be retrieved simply by testing for each individual x in $\mathcal{A}$ if $\mathcal{K} \models x : D$. However, this would clearly be very inefficient if $\mathcal{A}$ contained a large number of individuals.

An alternative approach is to add a new axiom $C_x \sqsubseteq D$ to $\mathcal{T}$ for each axiom $x : D$ in $\mathcal{A}$, where C_x is a new atomic concept; we will call such concepts *pseudo-individuals*. Classifying the resulting TBox is equivalent to performing a complete realisation of the ABox: the most specific atomic concepts that an individual x is an instance of are the most specific atomic concepts that subsume C_x and that are not themselves pseudo-individuals. Moreover, the instances of a concept D can be retrieved by computing the set of pseudo-individuals that are subsumed by D. The problem with this latter approach is that the number of pseudo-individuals added to the TBox is equal to the number of individuals in the ABox, and if this number is very large, then TBox reasoning may become inefficient or even break down completely (e.g., due to resource limits).

The basic idea behind *iS* is to overcome this problem by using a DL reasoner to classify the TBox and a database to store the ABox, with the database also being used to store a complete realisation of the ABox, i.e., for each individual x, the concepts that x realises (the most specific atomic concepts that x instantiates). The realisation of each individual is computed using the DL (TBox) reasoner when an axiom of the form $x : C$ is added to the *iS* ABox.

A retrieval query Q to *iS* (i.e., computing the set of individuals that instantiate a concept Q) can be answered using a combination of database queries and TBox reasoning. Given an *iS* containing a KB $\langle \mathcal{T}, \mathcal{A} \rangle$ and a query concept Q, retrieval involves the computation of sets of concepts and individuals which we denote as follows:

- $Q{\downarrow_{\mathcal{T}}}$ denotes the set of atomic concepts in $\mathcal{T}$ subsumed by Q; these are the equivalents and descendants of Q in $\mathcal{T}$.
- $\lceil Q \rceil_{\mathcal{T}}$ denotes the set of most specific atomic concepts in $\mathcal{T}$ subsuming Q; if Q is itself an atomic concept in $\mathcal{T}$ then clearly $\lceil Q \rceil_{\mathcal{T}} = \{Q\}$.
- I_1 denotes the set of individuals in $\mathcal{A}$ that realise *some* concept in $Q{\downarrow_{\mathcal{T}}}$;
- I_2 denotes the set of individuals in $\mathcal{A}$ that realise *every* concept in $\lceil Q \rceil_{\mathcal{T}}$.

The *iS* algorithm to retrieve the instances of Q can be then described as follows:

1. use the DL reasoner to compute $Q{\downarrow_{\mathcal{T}}}$;
2. use the database to find the set of individuals I_1;
3. use the reasoner to check whether Q is equivalent to any atomic concept in $\mathcal{T}$; if that is the case then simply return I_1 and *terminate*;
4. otherwise, use the reasoner to compute $\lceil Q \rceil_{\mathcal{T}}$;
5. use the database to compute I_2;

6. use the reasoner and the database to compute I_3, the set of individuals $x \in I_2$ such that $x : C$ is an axiom in $\mathcal{A}$ and C is subsumed by Q;

7. return $I_1 \cup I_3$ and *terminate*.

Proposition. The above procedure is sound and complete for retrieval, i.e., given a concept Q, it returns all and only individuals in $\mathcal{A}$ that are instances of Q.

The above is easily proved using the fact that we assume, without loss of generality, that for each individual there is only one axiom associated to it.

An Optimised Instance Store

In practice, several refinements to the above procedure are used to improve the performance of *iS*. In the first place, as it is potentially costly, we should try to minimise the DL reasoning required in order to compute realisations (when instance axioms are added to the ABox) and to check if individuals in I_1 are instances of the query concept (when answering a query).

One way to (possibly) reduce the need for DL reasoning is to avoid repeating computations for "equivalent" individuals, e.g., individuals x_1, x_2 where $x_1 : C_1$ and $x_2 : C_2$ are ABox axioms, and C_1 is equivalent to C_2. Since checking for semantic equivalence between two concepts would require DL reasoning (which we are trying to avoid), the optimised *iS* only checks for syntactic equality using a database lookup. (The chances of detecting equivalence via syntactic checks could be increased by transforming concepts into a syntactic normal form, as is done by optimised DL reasoners [17], but this additional refinement has not yet been implemented in *iS*.) Individuals are grouped into equivalence sets, where each individual in the set is asserted to be an instance of a syntactically identical concept, and only one representative of the set is added to the *iS* ABox as an instance of the relevant concept. When answering queries, each individual in the answer is replaced by its equivalence set. Similarly, we can avoid repeated computations of sub and super-concepts for the same concept (e.g., when repeating a query) by caching the results of such computations in the database.

Finally, the number and complexity of database queries also has a significant impact on the performance of *iS*. In particular, the computation of I_1 can be costly as $Q\downarrow_{\mathcal{T}}$ may be very large. One way to reduce this complexity is to store not only the most specific concepts instantiated by each individual, but to store *every* concept instantiated by each individual. As most concept hierarchies are relatively shallow, this does not increase the storage requirement too much, and it greatly simplifies the computation of I_1: it is only necessary to compute the (normally) much smaller set of most general concepts subsumed by Q and to query the database for individuals that instantiate some member of such set. On the other hand, the computation of I_2 is slightly more complicated, because I_1 must be subtracted from the set of individuals that instantiate every concept in $\lceil Q \rceil_{\mathcal{T}}$. Empirically, however, the savings when computing I_1 seems to far outweigh the extra cost of computing I_2.

2 Implementation

We have implemented *iS* using a component based architecture that is able to exploit existing DL reasoners and databases. The core component is a Java application [1] talking to a DL

reasoner via the DIG interface [5] and to a relational database via JDBC. We have tested it with FaCT [18] and RACER reasoners and MySQL, Hypersonic, and Oracle databases.

```
initialise(Reasoner reasoner, Database db, TBox t)
addAssertion(Individual i, Concept C)
retract(Individual i)
retrieve(Concept Q): Set⟨Individual⟩
```

Figure 1: Basic functionality of *iS*

The basic functionality of *iS* is illustrated by Figure 1. The four basic operations are `initialise`, which loads a TBox into the DL reasoner, classifies the TBox and establishes a connection to the database; `addAssertion`, which adds an axiom $i : D$ to *iS*; `retract`, which removes any axiom of the form $i : C$ (for some concept C) from *iS*; and `retrieve`, which returns the set of individuals that instantiate a query concept Q. Since an *iS* ABox can only contain one axiom for each individual, asserting $i : D$ when $i : C$ is already in the ABox is equivalent to first removing i and then asserting $i : (C \sqcap D)$.

In the current implementation, we make the simplifying assumption that the TBox itself does not change. Extending the implementation to deal with monotonic extensions of the TBox would be relatively straightforward, but deleting information from the TBox might require (in the worst case) all realisations to be recomputed.

Database. For the basic *iS*, the database schema is straightforward: a table with all the assertions stored as pairs individual/concept (with individual as primary key), and a table of pairs individual/*atomic* concept. The latter table holds the asserted individuals together with the most specific atomic concepts instantiated by them.

For the optimised *iS*, the database schema is illustrated in Figure 2. There is a main

```
Concepts(id, concept)
Assertions(individual, conceptId)
Types(conceptId, atomicConcept)
Equivalents(conceptId, atomicConcept)
Parents(conceptId, atomicConcept)
Children(conceptId, atomicConcept)
```

Figure 2: Database Schema for the Optimised *iS*

`Concepts` table assigning a unique `id` to every asserted or retrieved concept; the `conceptId` in the other tables is a foreign key referencing `id`. Apart from the evident `Assertions` table, the remaining tables hold TBox information inferred using the reasoner: the `Types` table holds all ancestors and equivalents of the asserted/retrieved concepts, while the position of the concepts in the taxonomy is recorded by either storing their equivalents if they exist or both their children and parents in the corresponding tables.

3 Empirical Evaluation

To illustrate the scalability and performance of *iS* we describe the tests we have performed using the gene ontology and its associated instance data. We also illustrate how this compares

with existing non-specialised ABox reasoning techniques by describing the same tests performed using RACER and FaCT (the latter using the pseudo-individual approach discussed in Section 1).

The gene ontology (*GO*) itself, an ontology describing terms used in gene products and developed by the Gene Ontology Consortium [23], is little more than three taxonomies of gene terms, with a single role being used to add "part-of" relationships. However, the ontology is large (47,012 atomic concepts) and the instance data, obtained by mining the GO database [13] of gene products, consists of 653,762 individual axioms involving 48,581 distinct complex DL expressions using three more roles.

The retrieval performance tests use two sets of queries. The first set (Q1-Q5) was formulated with the help of domain experts and consists of five realistic queries that might be posed by a biologist. The second set (Q6-Q11) consists of six artificial queries designed to test the effect on query answering performance of factors such as the number of individuals in the answer, whether the query concept is equivalent to an atomic concept (if so, then the answer can be returned without computing I_3), and the number of candidate individuals in I_2 for which DL reasoning is required in order to determine if they form part of the answer. The characteristics of the various queries with respect to these factors is shown in Table 1.

Table 1: Query characteristics

Query	Equivalent to Atomic Concept	No. of Instances in Answer	No. of "candidates" in I_2
Q1	Yes	2,641	n/a
Q2	No	0	284
Q3	No	3	284
Q4	Yes	7,728	n/a
Q5	Yes	25	n/a
Q6	No	13,449	551
Q7	No	11,820	116
Q8	No	12	603
Q9	No	19	19
Q10	Yes	4,543	n/a
Q11	Yes	1	n/a

3.1 Loading and Querying Tests

In these tests, we compared the performance of *iS* with that of RACER using the GO TBox and different sized subsets of the GO ABox. The *iS* was first initialised with the GO TBox, then for each ABox, we measured the time (in CPU seconds) taken to load the ABox into it. A comparison with RACER is shown in Table 2.

The time taken by the *iS* to load the ABoxes increases more slowly than their size: for ABox size 200, *iS* takes about 1s to add each individual axiom; by the time the ABox size has reached 400,000 this has fallen to approximately 0.25s per axiom. In view of the equivalent individuals optimisation employed by *iS*, however, it may be more relevant to consider the

Table 2: *iS* and RACER load and realise times (CPU seconds)

Number of Individuals	Distinct Descriptions	Load & Realise (s)	
		iS	RACER
200	155	189	180
500	330	405	3,420
1,000	591	804	22,320
2,000	1,017	1,395	fault
5,000	2,024	2,906	fault
10,000	3,299	5,988	fault
20,000	5,364	11,057	fault
50,000	9,760	21,579	fault
100,000	15,147	33,456	fault
200,000	23,387	56,613	fault
400,000	35,800	96,503	fault
653,762	48,581	140,623	fault

time taken per distinct description: this increases from about 1s per description for the size 200 ABox (which contains 155 distinct descriptions) to approximately 3s per description for the size 653,762 ABox (which contains 48,581 distinct descriptions).

The time taken by RACER to realise the smallest ABox is roughly the same as that taken by *iS*. As the ABox size grows, however, the time taken by RACER increases rapidly, and at ABox size 1,000 it is already taking approximately 22s per axiom. For larger ABoxes, RACER broke down due to a resource allocation error in the underlying Lisp system.

Next, we measured retrieval times. For RACER, we carried out the same tests in two different ways. In both cases we first initialised RACER with the GO TBox, then loaded the ABox. In the first test, we used the *realize-abox* function to force RACER to compute a complete realisation of the ABox before answering any queries; if the realisation was successfully completed, we then timed how long it took to answer each of the queries. In the second test, we simply timed how long it took RACER to answer each of the queries without first forcing it to realise the ABox.

The results for *iS* when answering each of the five realistic queries and six artificial queries are plotted against the size of the ABox in Figure 3; note the logarithmic scales on both axes.

As can be seen, the time taken to answer queries like Q6 and Q8 becomes quite large. In these cases, since the number of individuals in I_2 is large the time taken to check if these individuals (roughly 0.2s per individual) dominates other factors. The number of "distinct" individuals in the answer also has a significant impact on performance: when there are many such individuals, the database query required in order to compute the complete answer set can be quite time consuming.

The results for RACER when answering the same sets of queries are also taken, both in the case where the ABox has been realised and where it has not. Timings are only approximate, as precise measurements were not possible when using RACER under Windows. When the ABox

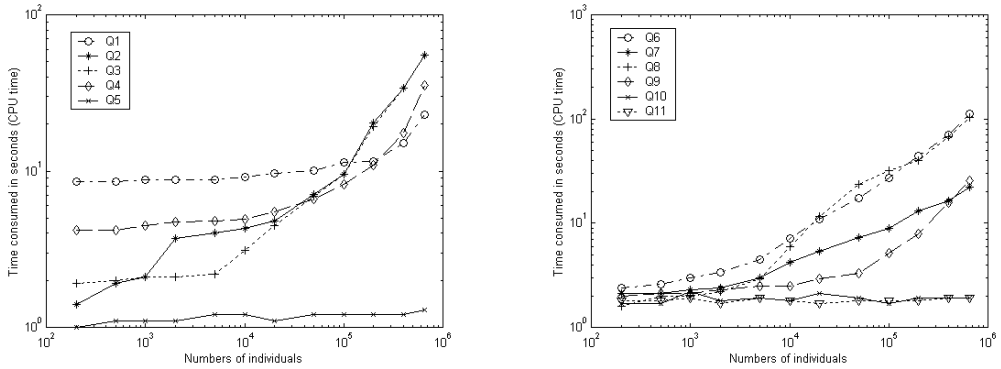

Figure 3: *iS* realistic (left) and artificial (right) query times -v- ABox size

had been realised, queries were answered almost *instantly*, but results are only available for the relatively small ABoxes that RACER was able to realise (up to 1,000 individuals). When the ABox was not realised, answers were again returned almost *instantly* for smaller ABoxes, but when the ABox size exceeded 1,000 individuals the answer times increased dramatically, and for ABoxes larger than 10,000 individuals (larger than 5,000 in the case of Q9) RACER again broke down due to a resource allocation error in the underlying Lisp system.

It should be mentioned that the results for *iS* include significant communication overheads (both with the database and DL reasoner), which was not the case for RACER since queries were posed directly via its command line interface.

3.2 Pseudo-individual Tests

As discussed in §1, one way to deal with role-free ABoxes is to treat individuals as atomic concepts in the TBox (pseudo-individuals). To test the feasibility of this approach, we again used the GO TBox and ABox, and the set of queries described above. To make the comparison fair, only the distinct instantiated concept expressions are used. The FaCT system was used in these tests as RACER broke down when trying to classify the GO TBox augmented with the pseudo-individuals, again due to a resource allocation error in the underlying Lisp system.

In order to investigate how the pseudo-individual approach would scale with increasing ABox (and hence TBox) size, we tried computing the concepts subsumed by each query with the GO TBox alone (which contains 47,012 concept names) and with the TBox augmented with the pseudo-individuals derived from the GO ABox (a total of 95,593 concept names). The results of these tests are given in Table 3. It is important to note that they do not include the time required to expand answers to include sets of equivalent individuals—as discussed above, this can be quite time consuming for some queries (e.g., 19.5s in the case of Q9 with the largest ABox).

As one can see, the time taken to compute the answers to the queries is heavily dependent on the size of the answers, and in the case of Q4 with the pseudo-individual augmented TBox, the time was over 600s. This is in contrast to *iS*, where the size of answer had comparatively little effect on the time taken to answer queries. For queries with relatively small answers,

Table 3: Pseudo-individual query time (CPU seconds) and answer size

Query	GO TBox		GO TBox + ABox	
	Time	Answer Size	Time	Answer Size
Q1	8.1	220	233.3	2,861
Q2	1.3	1	1.2	1
Q3	0.2	1	1.4	4
Q4	26.0	881	631.8	8,609
Q5	0.5	2	5.2	27
Q6	4.3	86	176.6	2,450
Q7	1.4	1	10.0	147
Q8	1.3	1	1.5	7
Q9	1.4	1	3.5	22
Q10	4.2	109	114.4	1,407
Q11	0.5	1	2.0	2

however, the pseudo-individual approach was highly effective, even for queries that were time consuming to answer using *iS*.

4 Discussion and Future Work

Our experiments show that *iS* provides stable and effective reasoning for role-free ABoxes, even those containing very large numbers of individuals. In contrast, full ABox reasoning using the RACER system exhibited accelerating performance degradation with increasing ABox size, and at least the current RACER release was not able to deal with the larger ABoxes used in our evaluation. The pseudo-individual approach to role-free ABox reasoning was more promising, and may be worth further investigation.

The acceptability of the performance of *iS* would obviously depend on the nature of the application and the characteristics of the KB and of typical queries. It is likely that the performance of *iS* can be substantially improved simply by dealing with constant factors such as communication overheads.

Future work includes the investigation of additional optimisations and enhancements, such as providing a more sophisticated query interface. We are also investigating ways to extend *iS* to ABoxes that are not completely role-free. This may be possible in restricted cases by applying some form of *precompletion* [16] to the ABox.

Acknowledgements.

Thanks to Phil Lord for help with the implementation and to Chris Wroe for help with the GO ontology and the formulation of realistic queries.

References

[1] Instance Store website. `http://instancestore.man.ac.uk`.

[2] W. A. Andersen, K. Stoffel, and J. A. Hendler. Parka: Support for extremely large knowledge bases. In G. Ellis et al, editor, *Proc. First Int'l KRUSE Symposium*, pages 122–133, 1995.

[3] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. CUP, 2003.

[4] 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.

[5] Sean Bechhofer. The DIG description logic interface: DIG/1.1. In *Proceedings of the 2003 Description Logic Workshop (DL 2003)*, 2003.

[6] Tim Berners-Lee. *Weaving the Web*. Harpur, San Francisco, 1999.

[7] Alexander Borgida and Ronald J. Brachman. Loading data into description reasoners. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 217–226, 1993.

[8] Olga Caprotti, Mike Dewar, and Daniele Turi. Mathematical service matching using Description Logic and OWL. Technical Report IST-2001-34145, MONET Consortium, March 2004.

[9] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, and L.A. Stein. OWL web ontology language 1.0 reference, July 2002.

[10] M. Uschold et al. A semantic infosphere. In D. Fensel et al, editor, *Proc. ISWC 2003*, number 2870 in LNCS, pages 882–896. Springer, 2003.

[11] Stephen Dill et al. Semtag and seeker: Bootstrapping the semantic web via automated semantic annotation. In *Proc. WWW 2003*, 2003.

[12] GO project. European Bioinformatics Institute. `http://www.ebi.ac.uk/go`.

[13] Gene Ontology Database, 2003. `http://www.godatabase.org/dev/database/`.

[14] V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. IJCAI 2001*, 2001.

[15] Volker Haarslev and Ralf Möller. RACER system description. In *Proc. IJCAR 2001*, volume 2083 of *LNAI*, pages 701–705. Springer, 2001.

[16] Bernhard Hollunder. Consistency checking reduced to satisfiability of concepts in terminological systems. *Ann. of Mathematics and Artificial Intelligence*, 18(2–4):133–157, 1996.

[17] I. Horrocks. Implementation and optimisation techniques. In F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 306–346. CUP, 2003.

[18] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[19] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In *Proc. of the 2nd International Semantic Web Conference (ISWC)*, 2003.

[20] Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology. In *Proc. WWW 2003*, pages 331–339. ACM, 2003.

[21] 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.

[22] Sergio Tessaris. *Questions and Answers: Reasoning and Querying in Description Logic*. PhD thesis, University of Manchester, Department of Computer Science, April 2001.

[23] The Gene Ontology Consortium. Gene ontolgy: Tool for the unification of biology. *Nature Genetics*, 25(1):25–29, 2000.