J Intell Inf Syst (2011) 37:217–244
DOI 10.1007/s10844-010-0133-4

Using ontology databases for scalable query answering, inconsistency detection, and data integration

Paea LePendu · Dejing Dou

Received: 8 October 2009 / Revised: 21 July 2010 / Accepted: 1 September 2010 /
Published online: 22 September 2010

Abstract An ontology database is a basic relational database management system that models an ontology plus its instances. To reason over the transitive closure of instances in the subsumption hierarchy, for example, an ontology database can either unfold views at query time or propagate assertions using triggers at load time. In this paper, we use existing benchmarks to evaluate our method—using triggers—and we demonstrate that by forward computing inferences, we not only improve query time, but the improvement appears to cost only more space (not time). However, we go on to show that the true penalties were simply opaque to the benchmark, i.e., the benchmark inadequately captures load-time costs. We have applied our methods to two case studies in biomedicine, using ontologies and data from genetics and neuroscience to illustrate two important applications: first, ontology databases answer ontology-based queries effectively; second, using triggers, ontology databases detect instance-based inconsistencies—something not possible using views. Finally, we demonstrate how to extend our methods to perform data integration across multiple, distributed ontology databases.

Keywords Ontology · Database · Knowledge base · Scalability · Semantic Web

P. LePendu · D. Dou
Computer and Information Science Department,
University of Oregon, Eugene, OR 97403, USA

P. LePendu
e-mail: paea@cs.uoregon.edu

D. Dou
e-mail: dou@cs.uoregon.edu

Present Address:
P. LePendu (✉)
National Center for Biomedical Ontology,
Stanford Center for Biomedical Informatics Research,
Stanford University, Stanford, CA, USA
e-mail: plependu@stanford.edu

1 Introduction

Researchers are using Semantic Web ontologies extensively in intelligent information systems to annotate their data, to drive decision-support systems, to integrate data, and to perform natural language processing and information extraction. Ontologies provide a means of formally specifying complex descriptions and relationships about information in a way that is expressive yet amenable to automated processing and reasoning. As such, they offer the promise of facilitated information sharing, data fusion and exchange among many, distributed and possibly heterogeneous data sources. However, the uninitiated often find that applying these technologies to existing data can be challenging and expensive. In this paper, we present an easy method for users to manage an ontology plus its instances using an off-the-shelf database management system like MySQL. We call these sorts of databases *ontology databases*.

We demonstrate that ontology databases using our trigger-based method scale well. Most importantly, ontology databases are useful for handling ontology-based queries. That is, users will get answers to queries that take the subsumption hierarchy into account along with other features such as restrictions. Moreover, we can perform instance-based inconsistency detection, which is not possible using view-based methods. Ontology engineers benefit from instance-based inconsistency detection because such inconsistencies could indicate possible errors in the ontology, which are difficult to detect otherwise. Finally, our method extends easily to perform integration across distributed, heterogeneous ontology databases using an inference-based framework.

We evaluated our system to carefully characterize its performance benefits (and limitations) so that users can decide how applicable it is for their domain. In particular, knowledge base systems, like ontology databases, will commonly pay an amortized penalty up-front by materializing inferences (e.g., forward chaining, materializing views, computing transitive closures, etc.) such that queries run much faster—so system architects need to understand the tradeoffs. Therefore, we used the Lehigh University Benchmark (LUBM) (Guo et al. 2005) to measure the query-time and load-time performance for our implementation, where triggers act to forward compute inferences. The results on query performance show the expected gains, but the load-time costs were surprising: we saw no apparent cost for pre-computing the transitive closure of the instances using triggers. We hypothesized that the benchmark was inadequately characterizing the space-time tradeoffs, so we designed a new series of benchmark ontologies and re-tested the system to illuminate load-time costs that account for the wide variability among ontologies. As a result, we are able to carefully characterize for what kind of ontology and size of data our system works well.

Like similar systems do, our system answers queries and takes the subsumption hierarchy into account, as demonstrated in our case study on neuroscience data. What distinguishes our work in large part is the use of active database technologies (i.e., triggers) in an off-the-shelf database system to maintain the knowledge model, which only a few other systems have done. In addition to supporting integrity constraint checking for domain and range restrictions, we also incorporate limited, instance-based inconsistency checking by introducing explicit negations via *not-gadgets* into the ontology database structure. In biomedical domains where data is gathered empirically and annotated by ontologists, we can use this technology to support the ontology engineering process, i.e., we can use the biological evidence to refute claims about the ontology definition. Doing so, we found 75 logical inconsistencies in the Gene Ontology (GO) (Gene Ontology Consortium 2006).

This manuscript extends our previously published work (LePendu et al. 2008, 2009) in three ways. First, we explore in greater depth the scalability, performance and expressiveness of a trigger-based approach via our case studies. We developed a method for generating new benchmark ontologies which improves upon existing benchmarks by taking the wide diversity among ontologies into consideration. Second, we add proof reconstruction capability to our system in order to explain logical inconsistencies. Finally, we extend the trigger-based method to perform data integration: using inferential data integration theory, we can propagate inferences through a network of ontology databases via triggers and message-passing.

We organized the rest of this paper as follows. Section 2 outlines the general goal and main contributions of our work; it also presents the motivation we obtain from other, related works. Section 3 summarizes the main idea behind and key implementation details for our system. We end Section 3 with a short discussion on what differentiates our system from other, similar systems. Sections 4–6 present case studies on questions of scalability, ontology-based query answering, and inconsistency detection. Section 7 describes how to extend our system to perform integration. Finally, we conclude in Section 8.

2 Our goal and motivation

An ontology defines terms and specifies relationships among them, forming a logical specification of the semantics of some domain. Most ontologies contain a hierarchy of terms at minimum, but many have more complex relationships to consider. Researchers in biomedicine use ontologies heavily to annotate their data and to drive decision support systems that translate to applications in clinical practice (Shah et al. 2009). What makes this work challenging in part is to have systems that will handle basic reasoning over the relationships in an ontology in a transparent manner and that will scale to large data sets.

Our goal is not new, just our approach: We need the capabilities of an efficient, large-scale knowledge base system (using Semantic Web ontologies), but we want a solution as transparent as managing a regular relational database system, such as MySQL. Luckily, we do not have to port legacy systems (like EKS-V1 (Vieille et al. 1992)) to suite our needs because many of the descendant technologies have made their way into everyday relational database management systems, like MySQL. In other words, when used in the proper context, regular databases can behave like efficient, deductive systems. While perhaps obvious to researchers of logic and databases, it is not as obvious to many biomedical informaticians who require intelligent information systems like these.

An *ontology database* takes a Semantic Web ontology as input and generates a database schema based on it. When individuals in the ontology are asserted in the input, the database tables are populated with corresponding records. Internally, the database management systems processes the data and the ontology in a way that maintains the knowledge model, much like a basic knowledge base would. As a result, after the database is bootstrapped in this way, users may pose SQL queries to the system declaratively, based on terms from the ontology, and they get answers in return that incorporate the term hierarchy or other logical features of the ontology. Our system, which we call *OntoDB*, includes the following features:

Ease	Users merely require an off-the-shelf database system like MySQL.
Scalability	Users can input large data sets under medium-sized ontologies.
Subsumption	Researchers can ask declarative, ontology-based queries.
Inconsistency	Ontologists can detect logical errors arising from asserted data.
Integration	Analysts can integrate heterogeneous data using our method.

2.1 Related work

2.1.1 Knowledge-based systems

Knowledge-based systems (KBs) use a knowledge representation framework, having an underlying logical formalism (a language), together with inference engines to deductively reason over a given set of knowledge. Users can *tell* statements to the KB and *ask* it queries (Levesque and Lakemeyer 2001), expecting reasonable answers in return. An ontology, different from but related to the philosophical discipline of Ontology, is one such kind of knowledge representation framework (Guarino 1998). In the Semantic Web (Berners-Lee et al. 2001), description logic (DL) (Baader et al. 2003) forms the underlying logic for ontologies encoded using the standard Web Ontology Language[1] (OWL). One of the major problems with Semantic Web KBs is they do not scale to very large data sets (Haarslev and Möller 2001).

2.1.2 Reasoning

Researchers in logic and databases have contributed to the rich theory of deductive database systems (Gallaire et al. 1977; Gallaire and Nicolas 1990). For example, Datalog (Ullman 1988) famously uses views for reasoning in Horn Logic. We already mentioned EKS-V1 (Vieille et al. 1992). Reasoning over negations and integrity constraints has also been studied in the past (Clark 1977; Kowalski et al. 1987). Of particular note, one of the side-remarks in one of Reiter's papers (Reiter 1977) formed an early motivation for building our system: Reiter saw a need to balance time and space in deductive systems by separating extensional from intensional processing. However—33 years later—space has become expendable.

Other works move beyond Datalog views to incorporate active rules for reasoning, such as ConceptBase (Jarke et al. 1995). An active rule, like a trigger in a database, is a powerful mechanism using an event-condition-action model to perform certain actions whenever a detected event occurs within a system that satisfies the given condition. Researchers in object-oriented and deductive database systems use active technologies in carefully controlled ways to also manage integrity constraints and other logical features (Buchmann et al. 1992; Ceri et al. 1992; Chakravarthy et al. 1992; Curé and Squelbut 2005; Dietrich et al. 1992; Vasilecas and Bugaite 2007). Researchers are studying how to bring database theory into the Semantic Web (Calvanese et al. 2005; Motik et al. 2007), but more work is needed in that regard.

[1] http://www.w3.org/TR/owl-features/

2.1.3 Scalability

Because reasoning in general poses scalability concerns, system designers use the Lehigh University Benchmark to evaluate and compare KB systems. We would characterize the Description Logic Database (DLDB) (Guo et al. 2004) as a kind of ontology database, which is similar in many regards to our OntoDB implementation: it uses a decomposed storage model (Abadi et al. 2009; Copeland and Khoshafian 1985; Horrocks et al. 2004) and mimics a KB using various features of a basic relational database system like MySQL. Other approaches build upon the very popular (but the less expressive) Resource Description Framework Schema (RDF[S]) to manage very large RDF triplestores (Broekstra et al. 2002; Christophides et al. 2004; Neumann and Weikum 2009).[2] Some of these triplestores use relational database backends, but they often treat the database as passive but efficient storage–query agents, when databases can do much more. Not all triplestores support reasoning, and, when they do, they are mostly limited to computing transitive closures precisely because RDFS is not very expressive. Thus, researchers have argued that query languages more robust than SPARQL are required (O'Connor and Das 2008).

2.1.4 Biomedical informatics

Many KB logics are probably more complex than users really require in biomedical domains (Baader and Morawska 2009). This community actively uses ontologies in numerous application areas ranging from phenotype and anatomy to neuroscience and disease; they collaborate on and share hundreds of ontologies (Noy et al. 2009); they use ontologies to assist with decision support (O'Connor and Das 2008), hypothesis evaluation (Racunas et al. 2004), information search and retrieval (Shah et al. 2009) and other intelligent, KB-oriented tasks.

The National Center for Biomedical Ontologies maintains a repository of several important resources and tools, including the BioPortal (Noy et al. 2009), which biomedical informaticians use in their intelligent information systems. A large community of researchers in the biomedical domain contribute and use these tools and services. In particular, the Gene Ontology Consortium (Gene Ontology Consortium 2000) has contributed the well-known Gene Ontology (GO), one of the most actively used ontologies. This community uses the GO to annotate large, manually curated repositories of gene and protein data based on published research related to model organisms, such as the zebrafish (*danio rerio*), fruit fly (*drosophilia*) and nematode (*C. elegans*). Each model organism database, such as for the mouse (MGI) or zebrafish (ZFIN) (Bult et al. 2008; Sprague et al. 2007), has its own web-based search capability for the species and their associated data. Moreover, this community shares large amounts of data using scripted import and export utilities—they could use better information integration tools. As another example, the Neural ElectroMagnetic Ontologies (NEMO) Consortium (Dou et al. 2007; Frishkoff et al. 2009) uses ontologies for integration and meta-analysis of brainwave data to better understand human brain function.

[2] http://esw.w3.org/LargeTripleStores

2.1.5 Information integration

Another important motivation for using ontologies is the promise they hold for integrating information. Researchers in biomedical informatics have taken to this idea with some fervor (Goble and Stevens 2008; Wache et al. 2001). One system in particular, OntoGrate, offers an *inferrential information integration* framework using ontologies which integrates data by translating queries across ontologies to get data from target data sources using an inference engine (Dou and LePendu 2006; Dou et al. 2006a, b). The same logical framework can be extended to move data across a network of repositories.

3 Ontology databases

In the following sections, we use a simple running example, the Sisiters–Siblings example, to illustrate how we implemented ontology databases. We begin with the basic idea as a whole, then explain how we structure the database schema and implement each kind of logical feature using triggers and integrity constraints.

3.1 The basic idea

We can perform rudimentary, rule-based reasoning using either views or triggers. For example, suppose we assert the statement (a rule): "All sisters are siblings." Then we assert the fact: "Mary and Jane are sisters." Logically, we may deduce using *modus ponens* (MP) that Mary (M) and Jane (J) are siblings:[3]

$$\frac{\text{Sisters}(x, y) \rightarrow \text{Siblings}(x, y) \quad \text{Sisters}(M, J)}{\text{Siblings}(M, J)} \ \text{MP}\{x/M, y/J\}$$

If sibling and sister facts are stored in two-column tables (prefixed with "a_" to denote an asserted fact), then we can encode the rule as the following SQL view:

```
CREATE VIEW siblings(x,y) as
SELECT x,y FROM a_siblings
UNION
SELECT x,y FROM sisters
```

In the view-based method, every inferred set of data necessarily includes its asserted data (e.g., siblings contains a_siblings and sisters contains a_sisters). Note: when the view is executed, the subquery retrieving sisters will unfold to access all asserted sisters data. Recursively, if sisters subsumes any other predicate, it too will be unfolded. Database triggers can implement the same kind of thing:

```
CREATE TRIGGER subproperty_sisters_siblings
ON INSERT (x,y) INTO sisters
FIRST INSERT (x,y) INTO siblings
```

[3]The notation {x/M,y/J} denotes that the variable x gets substituted with M, y with J, and so on, as part of the unification process.

The deduction is reflected in the answer a query such as "Who are the siblings of Jane?" Of course, the answer returned—in both cases—is Mary. We easily formulate the SQL query:

```
SELECT x FROM siblings WHERE y=Jane
```

What differentiates these two methods is that views are goal-driven: the inference is performed at query time by unfolding views. Whereas, triggers forward propagate facts along rules as they are asserted, i.e., at load time. In the remainder of this paper, we advocate a trigger-based approach as our preferred method of implementation and we provide some justification for this approach in our case studies. Most importantly, recall Reiter's space–time tradeoff: essentially, triggers use more space to speed up query performance; the technique has some of the advantages of materialized views but it differs in some important ways, cf., Section 3.3. Finally, aside from rule-based reasoning, triggers support other logical features we find important in biomedical domains, such as domain and range restrictions and inconsistency detection—we describe the methods for handling each case in the following implementation details.

3.2 Implementation details

3.2.1 Decomposition storage model

We use the decomposition storage model (Abadi et al. 2009; Copeland and Khoshafian 1985) because it scales well and makes expressing queries easy. Arbitrary models result in expensive and complicated query rewriting, so we do not even consider them. The two other suitable models in the literature are the horizontal and vertical models. Designers rarely use the horizontal model because it contains excessively many null values and is expensive to restructure: The administrator halts the system to add new columns to service new predicates. The vertical model is quite popular because it avoids those two drawbacks. Also, the vertical model affords fast inserts because records are merely appended to the end of the file. In fact, Sesame (Broekstra et al. 2002) and other RDF stores use the vertical storage model.

Unfortunately, the vertical storage model is prone to slow query performance because queries require many joins against a single table, which gets expensive for very tall tables. Furthermore, type-membership queries are somewhat awkward. As a typical workaround, designers first partition the vertical table to better support type-membership queries, then they partition it further along other, selected predicates that will optimize certain joins based on some informed heuristic. However, this leads back toward complicated query rewriting because the partitioning choices have to be recorded and unwound somehow.

We view the decomposition storage model as a fully partitioned vertical storage model, where the single table is completely partitioned along every type and every predicate. That is, each type and each predicate gets its own table. When taken to this extreme, query rewriting becomes straight forward again because each table corresponds directly to a query predicate. Therefore, the decomposition storage model keeps the advantages of the vertical model while improving query performance (because of the partitions) without introducing difficult query rewriting. Figure 1 illustrates the three different models using the Sisters–Siblings example.

Object	Type	Sister-of	Sibling-of
maryDoe	Female	janeDoe	janeDoe
johnDoe	Male	*null*	maryDoe

(a)

Predicate	Subject	Object
Type	janeDoe	Female
Type	maryDoe	Female
Sister-of	maryDoe	janeDoe
Sibling-of	maryDoe	janeDoe

(b)

Female

ID
janeDoe
maryDoe

SisterOf

Subject	Object
maryDoe	janeDoe

SiblingOf

Subject	Object
maryDoe	janeDoe

(c)

Fig. 1 The Sisters–Siblings examples using the **a** horizontal, **b** vertical, and **c** decomposition storage model

3.2.2 Subsumption

Ontology engineers often specify subclass relationships in Semantic Web ontologies, which form a subsumption hierarchy, constituting the majority of reasoning for biomedical ontologies (Baader and Morawska 2009). As we mentioned previously when describing the basic idea (cf. Section 3.1), subclass relationships are handled in much the same way as views are used in Datalog, but we use triggers instead.

Although they are very similar, Datalog views differ from inclusion axioms in description logic (Baader and Nutt 2003). In other words, the semantics of these two logical formalisms differ epistemically:

$$\text{Sisters} \rightarrow \text{Siblings} \;\; \text{vs.} \;\; \text{Siblings} \sqsubseteq \text{Sisters}$$

The literature suggests that these differences are formally captured using modal logic (Baader and Nutt 2003), but to us, the essence comes down to ensuring that the contrapositive of the rule is enforced as an integrity constraint (Reiter 1992) (not a rule): "if Siblings(M,J) is not true, then Sisters(M,J) cannot possibly be true (otherwise, raise an inconsistency error)." We therefore implement the contrapositive as a foreign-key constraint as follows:

```
CREATE TABLE Siblings (
  subject VARCHAR NOT NULL,
  object VARCHAR NOT NULL,
  CONSTRAINT fk-Sisters-Siblings FOREIGN KEY (subject,object)
    REFERENCES Sisters(subject,object)
    ...)
```

Figure 2a illustrates the two parts of an inclusion axiom graphically. The trigger rule event is indicated in the figure by the a star-like symbol, denoting that the detected assertion causes a trigger to fire. In this example, rather than Sisters–Siblings per se, we are enforcing another related rule so that we can build-up the example a bit: “All females are person(s), i.e., Female → Person.” Therefore, asserting Female(Mary) causes the trigger to actively assert Person(Mary). Finally, the contrapositive is checked using the foreign key. Note: consistency requires that forward-propagations occur *before* integrity checking, which explains using the keywords “before” or “first” in our trigger definitions (see above).

3.2.3 Domain and range restrictions

Another important feature of Semantic Web ontologies are domain and range restrictions. These restrict the possible set of instances that participate in property assertions. For example, “Only person(s) may participate in the sisters relationship.” Again, restrictions are formalized using modal logics and they correspond to integrity constraints. Therefore, we implement them as foreign key constraints on the subject or object (i.e., domain and range) of the property, similar to the contrapositive example above:

```
CREATE TABLE Sisters (
  subject VARCHAR NOT NULL,
  object VARCHAR NOT NULL,
  CONSTRAINT fk-Sisters-Subject-Person FOREIGN KEY (subject)
    REFERENCES Person(id)
    ...)
```

As in the contrapositive example for subsumption, the check marks in Fig. 2b denote that domain and range restrictions are an integrity check, implemented using a foreign-key. Before the assertion Sisters(Lily,Zena) is loaded, the database first verifies that Lily and Zena are already in the Person table, otherwise we raise an error. Note: the restriction can be even more specific, if we wanted, e.g., “Only *females* may participate in the sisters property.”

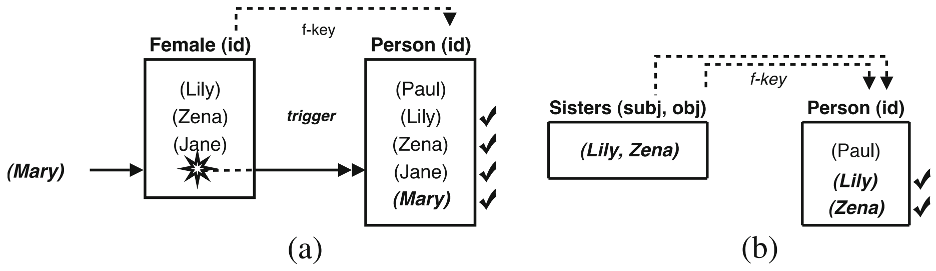


Fig. 2 The *star-like* symbol denotes an event fires a trigger rule. The *checkmark* symbol denotes an integrity check occurs. **a** Subsumption is implemented using a combination of triggers and integrity constraints. **b** Domain and range restrictions are implemented using foreign-key (f-key) constraints

3.2.4 Cardinality

Cardinality is another feature of moderate importance in biomedical domains. We support only limited cardinality constraints on properties: zero or one, for minimal or maximal participation. For example, the rule, "People have at most one social security number (SSN)," is a maximal cardinality constraint of one on the property hasSSN. Whereas, "Everyone has an SSN," expresses minimal cardinality on hasSSN.

In the first case, we use a uniqueness constraint (a key) to enforce at most one object is possible for any given subject for some relationship:

```
CREATE TABLE hasSSN (
  subject VARCHAR NOT NULL,
  object VARCHAR NOT NULL,
  UNIQUE KEY maxcard-hasSSN (subject)
  CONSTRAINT fk-hasSSN-Subject-Person FOREIGN KEY (subject)
    REFERENCES Person(id)
    ...)
```

The latter case, minimal cardinality, is slighlty more interesting. The feature suggests that an object value for some subject exists (at least one), but we may not know what it is, only that it exists. As Reiter suggests (Reiter 1992), we use a *null* value. V-tables (Imieliński and Lipski 1984) offer an alternative, more powerful approach that we did not choose because they are not widely adopted in database management systems. (Generally speaking, the problem corresponds to existential quantification and skolemization.) Therefore, we implement minimal cardinality using a trigger and null value whenever the value is unkown:

```
CREATE TRIGGER mincard_hasSSN
ON INSERT (x) INTO Person
FIRST INSERT (x,null) INTO hasSSN IF NOT EXISTS
```

3.2.5 Negation

Finally, we incorporate negative assertions as explicit negations which increases the kind of reasoning we can perform. To implement these, we introduce additional, negative tables (e.g., ¬Female) into the database structure, one for every positive table (e.g., Female).[4] For example, asserting that "Bob is not a Female" means putting Bob into the ¬Female table. This method doubles the total number of tables, but negative tables are usually much smaller than positive ones in our experience (cf. Section 6). Users can easily disable this optional feature.

We assume the law of the excluded middle, so a fact cannot be true and false at the same time, otherwise a contradiction occurs and we raise an exception. To enforce this, for every corresponding pair of positive and negative tables, we include what we call an *exclusion dependency* which we can implement universally on most systems

[4]We denote a negative table in the database such as ¬Female using an underscore prefix, e.g., _Female.

using a set of triggers (some systems might support using special kinds of foreign keys) as follows:

```
CREATE TRIGGER excl_female
ON INSERT (x) INTO female,
IF EXISTS (SELECT id FROM _female WHERE id=x)
THEN RAISE ERROR

CREATE TRIGGER excl_not_female
ON INSERT (x) INTO _female,
IF EXISTS (SELECT id FROM female WHERE id=x)
THEN RAISE ERROR
```

Graphically, negative tables and exclusion dependencies create distinctive structures, which we call *not-gadgets*. For example, if we consider the Sisters–Siblings subproperty graphically, it would look like the following diagram:

$$\begin{tikzcd}\text{Sisters} \arrow[r] & \text{Siblings}\end{tikzcd}$$

Adding the exclusion dependencies as undirected, dashed arrows, the concept graph takes on the distinctive not-gadget structure, as in the following diagram:

$$\begin{tikzcd}Sisters \arrow[r] \arrow[d, dash, dashed] & Siblings \arrow[d, dash, dashed] \\ \neg Sisters & \neg Siblings\end{tikzcd}$$

Not-gadgets help to detect inconsistencies as well as to answer a limited form of disjunctive query because they help to do basic refutation. The former is examined as a case study in Section 6. Although we do not go into great detail on the latter, suppose we have a simple ontology which states that, “All Persons are either Female or Male,” and the facts (i.e., its extension), “Bob and Jane are Persons. Jane is a Female. Bob is not a Female.”

We can model the disjunction using trigger rules and exclusion dependencies as shown in Fig. 3. The key idea is to use the first-order axiom which transforms a

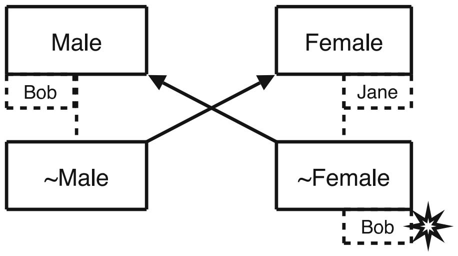


Fig. 3 The Male–Female covering axiom as a *not-gadget*

disjunction into a negative implication: $(\phi \vee \psi) \Rightarrow (\neg\phi \rightarrow \psi)$. In SQL, if we add the context that we are talking about Person instances, the implementation would be the following triggers:

```
CREATE TRIGGER disj_female_male
ON INSERT (x) INTO _female,
FIRST INSERT (x) INTO male WHERE (x) IN
  (SELECT y FROM person)

CREATE TRIGGER disj_male_female
ON INSERT (x) INTO _male,
FIRST INSERT (x) INTO female WHERE (x) IN
  (SELECT y FROM person)
```

Figure 3 also displays the data after the triggers have finished firing. Therefore, if we ask the disjunctive query, "Is Bob either a Male or Female?" a simple union of the Male and Female tables gives the correct answer: Yes! Of course, in theory, a reasoner should not require the negative fact, "Bob is not female," to reach that conclusion. Therefore, our method is clearly not capable of complete reasoning with disjunctions; it is limited.

Altogether, our running Sisters–Siblings example (merely eight predicates) constitutes a richly interacting set of tables, triggers and foreign-key constraints as illustrated in Fig. 4. One can imagine that for an ontology of 1,000 predicates (like NEMO) or over 70,000 predicates (like GO) can get quite complex!

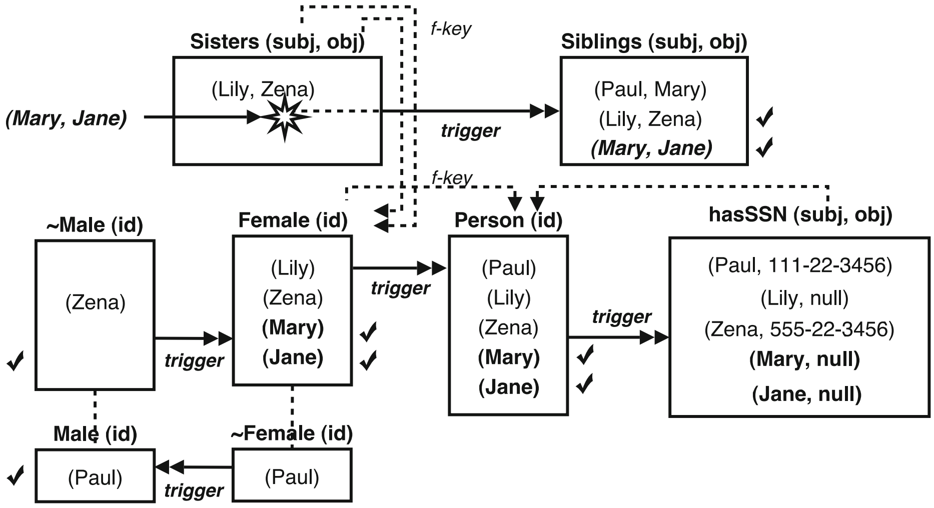


Fig. 4 The Sisters–Siblings example depicting restrictions, subsumption, cardinality constraints and not-gadgets all together

3.2.6 Proof reconstruction

To assist with analyzing the errors generated by integrity constraints and not-gadgets, we developed a method for logging and reconstructing the proof trees that result in raised exceptions. Users can use these proofs to determine the source of the error and decide what knowledge needs to be corrected. Users can also easily disable this optional feature.

In our implementation of this feature, each tuple in the database is accompanied by a boolean (1-bit) flag that represents whenever a fact is explicitly asserted versus inferred by deduction. Furthermore, every trigger rule logs the three main components of the inference being performed: premise, instance and conclusion. Using this log and the asserted data, we can reconstruct the steps that the database took to infer data.

To do this, we implemented a goal-oriented, backward-chaining algorithm (see Algorithm 1) such as those described for Horn Logic using *modus ponens*. The goal of the algorithm is to prove some statement, $\psi(\alpha)$, is true by providing the assertions that lead to making it true. Since we are assuming reasoning using *modus ponens*, we can easily reconstruct the proof tree from those assertions.

Algorithm 1 Proof Reconstruction for $\psi(\alpha)$.

```
GOAL: prove $\psi(\alpha)$ is true [?]
$C \Leftarrow \psi$ {conclusion}
$I \Leftarrow \alpha$ {instance}
**if** is_asserted $\psi(\alpha)$ **then**
  $\psi(\alpha)$ is asserted to be true [done!]
**else**
  $P \Leftarrow$ SELECT premise FROM log WHERE conclusion = $C$ AND instance = $I$
  **for all** $\phi(\beta)$ in $P$ **do**
    GOAL: prove $\phi(\beta)$ is true [?]
  **end for**
**end if**
```

3.3 Discussion: triggers are different than views

Inevitably, we encounter the need to update or delete data from a database. Deletions make maintaining the knowledge model more difficult, but they also elicit the key difference between view-based methods and trigger-based methods. Donini et al. formalize what is going on (Donini et al. 2002), and similar issues are discussed by Kowalski et al. (1987). We state the essence of the difference between views and triggers in ontology databases as a simple claim:

Claim A trigger-based ontology database has a distinctly different operational semantics from a view-based implementation with respect to deletions.

Proof By counterexample, assert the following in the given order: A→B, insert A(a), delete A(a). Now ask the query B(?x). A trigger-based implementation returns "{x/a}." A view-based implementation returns "null." □

Therefore, with respect to ontology databases, we can say definitively that a trigger-based approach is distinctly different from a materialized view-based approach. Indeed, as the literature attests, triggers are highly expressive. For example, views cannot reach the correct deduction in the following:

assert A→B
insert A(a)
negate B(a)
now ask the query B(?x)

A view-based approach returns "{x/a}" whereas a trigger-based approach (using not-gadgets) raises a contradiction as expected. The explanation is, simply, that views cannot differentiate negation from deletion, but triggers give us that power.

Scientists need to distinguish between deletion and negation since they often separate what they assume is the case from what they *know* is the case—as in hypotheses testing (Racunas et al. 2004). Our case study in Section 6 explores this in more detail.

4 Evaluating scalability using LUBM

Knowledge-based systems often pay an amortized penalty up-front my materializing inferences (e.g., forward chaining, materializing views, computing transitive closures, etc.) so that queries will run much faster—but at what cost? An ontology database using triggers is such a system. To better understand its capabilities, we compared our system, OntoDB, against the Description Logic Database (DLDB).

DLDB is a knowledge base system developed at Lehigh University that uses database features to assist with query answering on large sets of data. Much like OntoDB, it takes an ontology and a set of data as input; it creates and loads a relational database schema using the decomposition storage model; it stores the data in tables; and it uses database features to maintain the knowledge model and answer queries. DLDB differs from OntoDB in one major regard: it unfolds views at query time rather than materializes inferences at load time like OntoDB does. Clearly, forward-propagating knowledge will result in faster query response time. We confirmed this hypothesis by testing both systems using the LUBM, but we were surprised to find that our trigger-based approach paid no apparent penalty for the speed up. Further analysis reveals that the benchmark does not account well for larger or deeper ontologies, where the penalties become more apparent.

4.1 Lehigh University benchmark results

LUBM comprises of the univ-bench ontology, a data generator, and a set of 14 queries for evaluating the load time and query time of knowledge bases. The ontology is fixed: it has 43 classes and depth 6, and it describes a typical university environment (e.g., courses, students, faculty, departments). Although the ontology

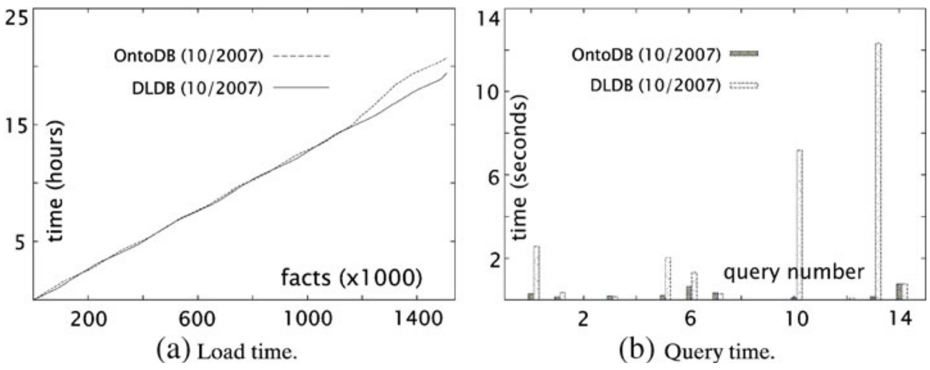


Fig. 5 LUBM(10,20) benchmark results for **a** load time and **b** query answering

does not change, the data generator creates a variable number of individuals based on two inputs: the number of universities (U) and the number of departments (D). The generated assertions, i.e., *data instances*, can be saved as a set of OWL files and loaded into a KB for evaluation. Therefore, the main idea is to vary the size of the data instances to quantify the scalability of a Semantic Web KB. Therefore, measuring both load time and query time with respect to the input parameters provides a picture of the total performance for LUBM(U,D).

We replicated the DLDB system and then experimentally confirmed[5] via the LUBM(10,20) our hypothesis that, by using triggers to materialize inferences, query performance clearly benefits by several orders of magnitude (cf. Fig. 5b) (LePendu et al. 2008). However—surprisingly—the gains come at no apparent costs as shown in Fig. 5a. The slope of the overlapping lines indicates the same *constant cost* per assertion for both systems.[6]

OntoDB requires roughly three times the disk space of DLDB, which makes sense considering the univ-bench ontology has, on average, a depth of three (maximum of six). Therefore, OntoDB makes roughly three copies of data as it propagates copies up the hierarchy. Based on the fact that the univ-bench ontology is so small and shallow, we designed a set of new benchmark ontologies that would expose the true load-time cost for precomputing inferences. That is, larger and deeper ontologies should demonstrate a non-constant cost per assertion.

4.2 Exposing load-time costs

Researchers have indicated before that the LUBM benchmark can be improved by reflecting real-world workloads based on various dimensions such as reasoning complexity (Ma et al. 2006), data distribution (Wang et al. 2005), and even ontology

[5]Experiment performed on a 1.8 GHz Centrino laptop with 1GB RAM in 10/2007.

[6]We confirmed by looking at system logs that the short divergence at about 1.2 million facts in Fig. 5a was due to virus scanning, a background interference. The constant cost per assertion clearly resumes after virus scanning terminates.

variation (Tempich and Volz 2003). However, to our knowledge, no other benchmark takes the diversity of ontologies into account to adequately characterize and expose previously opaque materialization costs like we do in this case study.

To test our hypothesis that the univ-bench ontology is too small and shallow, we need ontologies that vary in size and depth. Therefore, we created OntoGenerator, a tool that synthesizes ontologies that vary in size and depth. Based on our own experience studying biomedical ontologies, ontologies vary considerably: many are small like univ-bench, having hundreds of terms or less; a significant number have around a couple thousand terms; some are large like GO, having 25,000 terms or more; and very few are extraordinarily large like SNOMED-CT (Bodenreider et al. 2007), having 250,000 terms or more. Hence, we synthesized nine different ontologies that vary categorically by size (small–medium–large) and depth (shallow–mid–deep) (cf. Table 1).

OntoGenerator creates a synthetic ontology given the following parameters: a seed, the maximum number of classes, maximum number of siblings (i.e., span), density, and number of individuals. The density parameter introduces a degree of randomization in the fullness of the tree structure. It denotes the probability that the maximum number of siblings or the maximum depth will be reached along any path to a leaf node. We use the seed value to prime the randomizing function which allows us to reproduce the same ontology given the same parameters, or, conversely, to construct a new ontology (of the same kind) by using a different seed. Finally, the tool creates the given number of individuals as instances of randomly chosen classes in the ontology, thus distributing the data uniformly at various depths in the hierarchy.

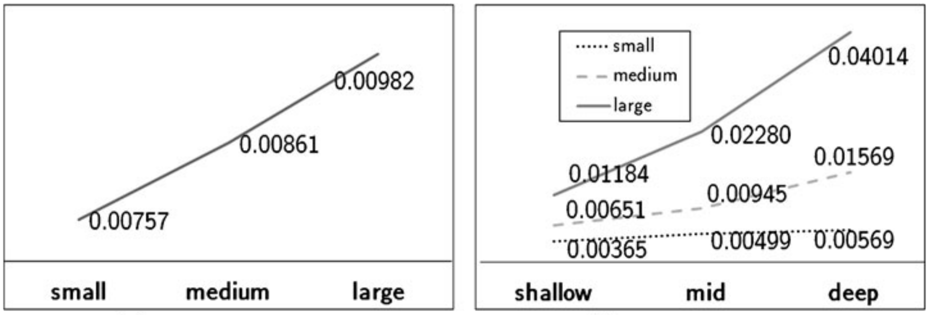
Finally, we tested our hypothesis that size and depth have a significant effect by populating OntoDB (the KB that performs materialization) with the nine ontologies and data instances that we generated. We measured load time in two phases: (1) the time to transform the ontology into a schema and load it into the database (averaged per class), and (2) the average time for loading a single instance assertion (taken 1,000 at a time). Table 1 notes the resulting load times.

As a result, our new hypothesis was confirmed: the results demonstrate that cost is *not constant* per assertion, but it depends on the size and depth of an ontology. The positive, crooked slope of the lines displayed in Fig. 6 show a clear, super-linear

Table 1 The nine different ontology parameters together with the corresponding schema load time and the data instance load time for each

	Ontology parameters		Schema load time		Instance load time	
	Size	Depth		Mean (ms)		Mean (ms)
Small	78	5	6.95 ms	7.57	3.65 ms	4.78
	81	10	7.12 ms		4.99 ms	
	72	20	8.64 ms		5.69 ms	
Medium	1,623	5	8.21 ms	8.61	6.51 ms	10.55
	1,555	10	8.93 ms		9.45 ms	
	1,827	20	8.68 ms		15.69 ms	
Large	19,992	5	9.57 ms	9.82	11.84 ms	24.93
	22,588	10	9.59 ms		22.80 ms	
	19,578	20	10.28 ms		40.14 ms	

Times are measured in seconds



(a) Schema load-time. (b) Instance load-time.

Fig. 6 **a** The average time (in seconds) that it takes to load a single term for small (≈100 terms), medium (≈2,000 terms) and large (≈20,000 terms) ontologies. **b** The average time (in seconds) that it takes to load a single data instance for small, medium and large sized ontologies of shallow (5), mid (10) and deep (20) depth

cost dependency. Furthermore, size and depth have a cumulative effect: the larger the ontology, the larger the role of depth.

In conclusion, although performance will gradually degrade for larger and deeper ontologies, ontology databases using triggers will scale well to handle data for most biomedical ontologies, including larger ones like GO. For the extraordinarily large ontologies, such as SNOMED-CT, the decomposition storage model will pose problems for the MySQL storage engine because of the sheer size of the schema, i.e., the number of tables, that will be created. For these exceptional ontologies, customized solutions are required.

5 Answering questions deductively in NEMO

Our first case study using Neural ElectroMagnetic Ontologies (NEMO)[7] (Dou et al. 2007; Frishkoff et al. 2009) demonstrates that ontology databases are useful for answering queries that take subsumption into account. NEMO is a medium-sized ontology having a higher degree of complexity than most biomedical ontologies. It comprises roughly of 1,000 classes and 70 properties arranged in a hierarchy having a depth of 16. At the time of the case study, NEMO recorded experimental measurements from brainwave studies on human-subjects and represented that data using object and datatype properties with domain and range restrictions, which comprise the majority of data instances.

5.1 NEMO data

NEMO focuses on brainwave studies which decompose, classify, label, and annotate event related potentials (ERP) data using ontological terms across many laboratories and experiments. ERPs are measures of brain electrical activity—electro-encephalographic (EEG) or "brainwave" activity—that are time-locked to

[7] http://bioportal.bioontology.org/ontologies/virtual/1321

functional, experimental events (e.g., the appearance of a word). These measures provide a powerful technique for studying brain function, because they are acquired non-invasively and can therefore be used in a variety of populations (e.g., children and patients, as well as healthy adults).

In addition, ERPs provide detailed information about the time dynamics, as well as the scalp spatial distribution, of neural activity during various cognitive and behavioral tasks. The NEMO ontology aims to capture these spatial, temporal and functional aspects of ERP studies so that results can be integrated across data sets for large-scale meta-analyses to better understand basic, human brain function.

Figure 7 displays a partial, graphical representation of classes, properties and relationships in an early version of the NEMO ontology used in this case study. In Fig. 7a is a representation of concepts from the NEMO ERP ontology used for this preliminary case study with P100 pattern and medial-frontocentral (MFRON) channel groups highlighted; (b) is a 128-channel EEG waveplot with positive voltage plotted up showing responses to words versus non-words; (c) is a time course of P100 pattern factor for same dataset, extracted using Principal Components Analysis; (d) is a topography of the P100 factor with negative on top and positive at bottom (cf. details in Frishkoff 2007); (e) is an international 10–10 layout with electrode

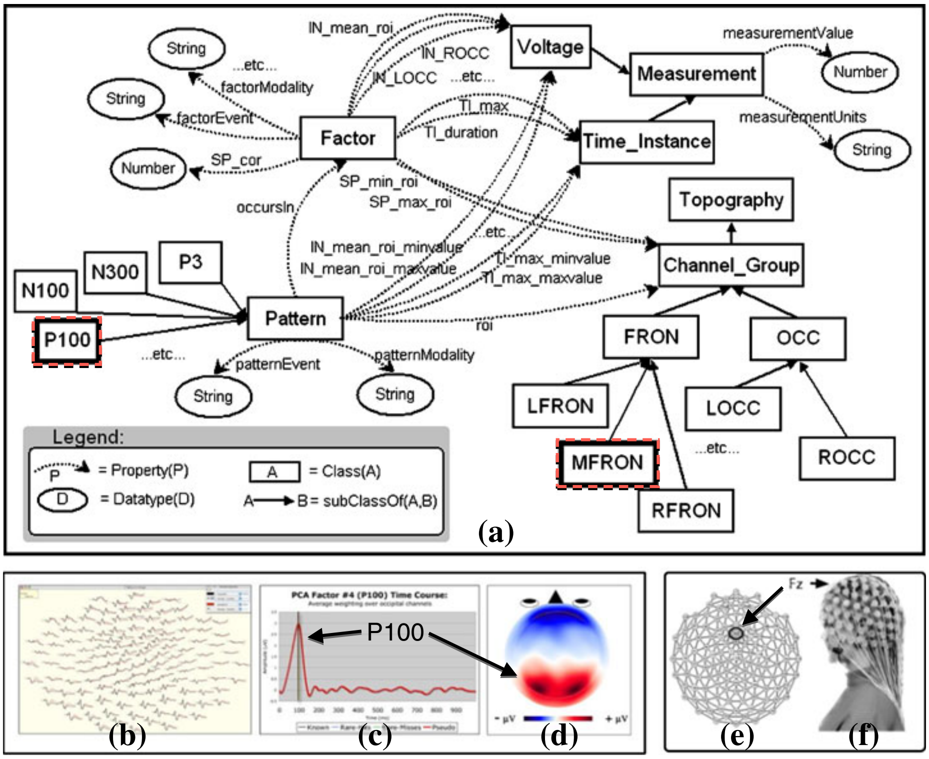


Fig. 7 Neural ElectroMagnetic Ontologies (NEMO)

location Fz highlighted, which is placed on the medial-frontocentral scalp region (top–down view); and (f) shows an EEG net applied to the scalp surface of a person (lateral view) with Fz location indicated.

5.2 Answering ontology-based queries

The ontology-database component stores the resulting data for large numbers of ERP data sets collected from multiple research sites. The database supports ontology-based querying and reasoning for queries such as the following, which require taking the subsumption hierarchy into account:

> Return all data instances that belong to ERP pattern classes which have a surface positivity over frontal regions of interest and are earlier than the N400.

In this query, "frontal region" can be unfolded into constituent parts (e.g., right frontal, left frontal; see Fig. 7). At an even more abstract level, the "N400" is a pattern class that is also associated with spatial, temporal, and functional properties. The pattern class labels can be inferred by applying a set of conjunctive rules. We used our internal rule language (Web-PDDL (Dou et al. 2005)) in this preliminary study, but the idea can also be implemented using the Semantic Web Rule Language, which we are currently exploring.

Preliminary results on the application of ontology databases for NEMO, using over 100,000 data instances, have been very promising. In particular, the neuro-scientists were attracted by the ability to pose queries at the conceptual level, without having to formulate SQL queries that take the complex logical interactions and reasoning aspects into consideration. Those high-level, logical interactions are modeled only once by specifying the ontology. We tested several other queries similar to the one above—examining ease of formulation, aggregation, subsumption, and total number of instances—against data that was annotated using an ontology similar to the one in Fig. 7. For example, we measured the time it takes to answer the following queries:

> Which patterns have a region of interest that is left-occipital and manifests between 220 and 300 ms?

> What is the range of intensity mean for the region of interest for N100?

In conclusion, in every case, we found it easy to formulate the neuroscientist's queries using terms from the ontology, not having to worry about the subsumed terminology. Furthermore, the system achieved 100% precision and recall, providing exactly the answers expected by our domain experts. This result was not surprising, given that, although the NEMO ontology used OWL features other than subclassing, only Horn Logic was required for reasoning. Finally, the performance for every query was extremely fast, on the order of only five to ten milliseconds, even for aggregations. Newer iterations of the NEMO ontology (cf. Frishkoff et al. 2009) will include disjoint subclasses, which will increase the complexity of the logic—we plan to approach this using not-gadgets, as we examine in the following study.

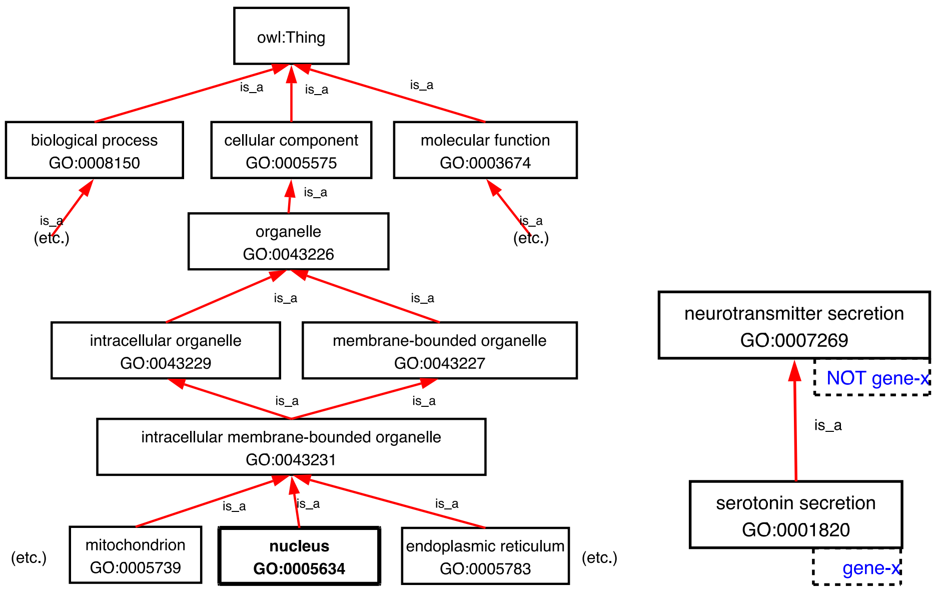
6 Detecting inconsistencies in GO

The Gene Ontology provides a unique scenario: we can use ontology databases with not-gadgets to detect inconsistencies that arise from negatively annotated data instances. As of March 2009, GO has over 25,000 classes arranged in a hierarchy having an average depth of 8 (maximum of 14). There are over 27 million known GO annotations—and growing—spanning the dozen or so model organisms. Hill et al. (2008) defined this problem as a generalization of what we call the *serotonin example*, which goes as follows:

> [GO annotations sometimes] point to errors in the type-type relationships described in the ontology. An example is the recent removal of the type serotonin secretion as an is_a child of neurotransmitter secretion from the GO Biological Process ontology. This modification was made as a result of an annotation from a paper showing that serotonin can be secreted by cells of the immune system where it does not act as a neurotransmitter.

6.1 GO annotations

In the serotonin example, two conflicting data instances, called "annotations," for genes influencing the biological process of serotonin secretion alluded to a problem in the ontology, which led to a correction in the GO hierarchy. For example, Fig. 8a illustrates a small portion the GO term hierarchy for nucleus (GO:0005634); whereas, Fig. 8b illustrates the inconsistency arising from the serotonin example. In


owl:Thing
is_a
biological process GO:0008150
cellular component GO:0005575
molecular function GO:0003674
(etc.)
organelle GO:0043226
intracellular organelle GO:0043229
membrane-bounded organelle GO:0043227
intracellular membrane-bounded organelle GO:0043231
mitochondrion GO:0005739
nucleus GO:0005634
endoplasmic reticulum GO:0005783
neurotransmitter secretion GO:0007269
NOT gene-x
serotonin secretion GO:0001820
gene-x

(a) The GO concept graph for nucleus. (b) The serotonin example.

Fig. 8 **a** A sample of the GO hierarchy highlighting the term nucleus. **b** An illustration of the serotonin secretion inconsistency the serotonin example, some gene (for simplicity, call it “gene-x”) was annotated as both being an instance of serotonin secretion while *not* being an instance of neurotransmitter secretion, causing the logical inconsistency based on the type-type (i.e., is_a) hierarchy. In these cases, an annotation means the gene influences a biological process in some way. Hill et al. explain how difficult it is in general for gene scientists to detect such data-driven inconsistencies in the GO, leaving it as an open problem to find ways to identify inconsistencies in the ontology based on annotations from the model organism databases such as ZFIN (Sprague et al. 2007) and MGI (Bult et al. 2008).

Of particular note, a logical inconsistency of this sort does not necessarily entail a type-type inconsistency. In fact, there are three possible explanations: (1) the positive annotation is incorrect, (2) the negative annotation is incorrect, or (3) the subsumption relationship is incorrect. We may even allow a fourth possibility: (4) the inconsistency is admissible, i.e., there is an exception or anomaly occurring in the biology of an organism for which we cannot account. Therefore, each inconsistency raises new hypotheses that should be investigated further by biologists—some may lead to new biological insights, refined models, or improved automated techniques.

We applied ontology databases with not-gadgets to this problem, using ZFIN and MGI annotations, and within two hours discovered 75 logical inconsistencies for the entire dataset. Furthermore, to the best of our ability, in conjunction with a domain expert using the GOOSE[8] database, we manually confirmed that each of these 75 results appear to maintain 100% precision and 100% recall (LePendu et al. 2009). There is no gold standard that we know of for this kind of data.

6.2 Detecting inconsistent annotations

Using ontology database to detect inconsistent annotations involves the following steps: (1) run the GO ontology through our tool to create the ontology database, with the not-gadgets option enabled; (2) load the generated relational schema into the MySQL database; (3) pre-process the ZFIN and MGI gene–term annotations to form instance-of assertions; (4) load the asserted instances into the database; and finally, (5) check the error log for the detected inconsistencies. It took about 30 min to load the schema and 80 min to load all the ZFIN and MGI annotations. ZFIN contained 91,000 annotations, 40 of which were negative ones. Whereas, MGI contained 154,000 positive and 292 negative facts.

OntoDB logged the inconsistencies, which fell into three categories. We provide the following examples of each case of inconsistency:

1. Intra-species logic inconsistencies between experimentally supported manual annotations: The zebrafish *p2rx2*[9] gene is annotated as both having (inferred from a genetic interaction) and *not* having (inferred from a direct assay) ATP-gated cation channel activity (GO:0004931).
2. Inter-species logic inconsistencies between experimentally supported manual annotations: The zebrafish *bad*[10] gene is annotated (inferred from a direct assay)

[8]http://www.berkeleybop.org/goose

[9]http://zfin.org/cgi-bin/webdriver?MIval=aa-markergoview.apg&OID=ZDB-GENE-030319-2

[10]http://zfin.org/cgi-bin/webdriver?MIval=aa-markergoview.apg&OID=ZDB-GENE-000616-1

as *not* being involved in the positive regulation of apoptosis (GO:0043065) in the zebrafish. Meanwhile, annotation of the corresponding mouse gene, *Bad*,[11] indicates it *is* involved in this biological process for the mouse (inferred from a mutant phenotype).

3. Logic inconsistencies between experimentally supported manual annotations and automated electronic annotations (between or within species): The zebrafish *lzic*[12] gene has been electronically annotated (inferred by electronic annotation) as having the function beta catenin binding (GO:0008013) and also *not* having the function beta catenin binding (inferred from physical interaction).

In conclusion, one of the most significant outcomes of this case study was the discovery of numerous intra-species inconsistencies arising from direct evidence versus automated electronic annotations (case 3). What this means is that ontology databases provide a method for validating millions of annotations that are automatically generated from human-defined transfer rules which may be incorrect. Another interesting case (case 2) illustrates possibly interesting evolutionary differences based on inconsistencies among species, e.g., between mice and zebrafish. Our discoveries were so relevant that our collaborator, an expert on GO annotations from ZFIN, submitted a tracking request to report of these kinds errors to other GO curators on an ongoing basis.[13]

7 Integrating information among OntoDBs

We extended our event-driven architecture for ontology databases so that it will also integrate two KBs. The key idea is to map ontology terms together, then to reason over them as a whole, i.e., as a merged ontology. We first use namespaces to distinguish terms from each KB; then, we map the ontologies together using *bridging axioms*; and finally, we reason over the entire, *merged ontology* to achieve integration. We have adapted the theory of *inferential information integration* developed by Dou et al. (Dou and LePendu 2006; Dou et al. 2005, 2006a, b) to define *inferential ontology database exchange*, a special kind of information integration in which data is exchanged, actively, among ontology databases.

7.1 Inferential information integration

In inferential information integration, query translation and data translation are formally defined as logical entailments with respect to a merged ontology having bridging axioms. By performing sound inference over the merged ontology's bridging axioms, the entailments under a target ontology can be inferred automatically (because $KB \vdash \phi$ implies $KB \models \phi$).[14] This method works well for our purposes because we only require sound (not complete) inference to achieve our desired results.

[11] http://www.informatics.jax.org/javawi2/servlet/WIFetch?page=markerGO&key=33374

[12] http://zfin.org/cgi-bin/webdriver?MIval=aa-markergoview.apg&OID=ZDB-GENE-040718-342

[13] http://sourceforge.net/tracker/?func=detail&aid=2686444&group_id=36855&atid=469833

[14] Please read the symbol $\vdash$ as infers and the symbol $\models$ as entails.

A merged ontology is similar to the notion of a global view over local schemas (global-as-view) (Lenzerini 20002) for data integration. It consists of the union of elements from a source and target ontology but also defines the semantic mappings between them as bridging axioms. A merged ontology allows all the relevant symbols in a domain to interact so that facts can be translated from one ontology to another using inference over the bridging axioms. Discovering bridging axioms is difficult and many claim that it cannot be fully automated. However, there are some semi-automatic systems, including ours (Qin et al. 2007).

Query translation is one way to integrate data. It applies mostly to scenarios in which one system mediates queries among various sources. A typical example would be a federated database (Sheth and Larson 1990). On the other hand, *data translation* is more common to data migration or exchange scenarios, where information is exported from one location to another (Kolaitis 2005). A typical example would be a data warehouse (Bernstein and Rahm 2000). We extend this idea to ontology databases by defining the following:

Definition 1 (Inferential Ontology Database Exchange) Let $\mathcal{S}$ be an ontology database source and $\mathcal{T}$ be an ontology database target specified by ontologies O_S and O_T, respectively. Let $\mathcal{M} = (O_S, O_T, \Sigma)$ be a merged ontology containing bridging axioms Σ, such that Σ is a set of Horn rules relating terms from O_S and O_T qualified with namespaces. Let d_S be a set of tuples in $\mathcal{S}$. The *inferential ontology database exchange* of d_S is the largest set of assertions d_T entailed by d_S with respect to $\mathcal{M}$.

By simply extending ontology databases with the corresponding namespace prefixes (supported by most DBMS platforms), we can translate and exchange any data asserted under a source ontology database $\mathcal{S}$ into data under $\mathcal{T}$ using active trigger rules for each bridging axiom expressed in Horn form. As data is inserted into a table in $\mathcal{S}$, the relevant event is detected, fires the appropriate triggers, and inserts the corresponding data into $\mathcal{T}$. The result is a network of ontology databases exchanging data via triggers.

7.2 Example implementation

We implemented two fabricated ontologies in the Teacher–Student domain and defined a merged ontology using bridging axioms as depicted in Fig. 9. In the figure, the ontology on the right (considered the source ontology) uses the Faculty term, whereas the target ontology on the left uses the Teacher term. For example, there is an axiom relating first and last name in the source ontology to full name in the target ontology. That axiom can be expressed as a rule using the built-in string concatenation function:

$$\forall x, y, z.\ \ \text{S:}\mathit{firstname}(x, y) \wedge \text{S:}\mathit{lastname}(x, z) \Rightarrow \text{T:}\mathit{fullname}(x, \text{concat}(y, \text{‘ ’}, z))$$

Because the rule is a conjunction, we implement it as a set of triggers, one for each conjunct. The reason we need two triggers is that satisfying either predicate in the conjunct could potentially fire the rule, so we need to set a listener on each predicate

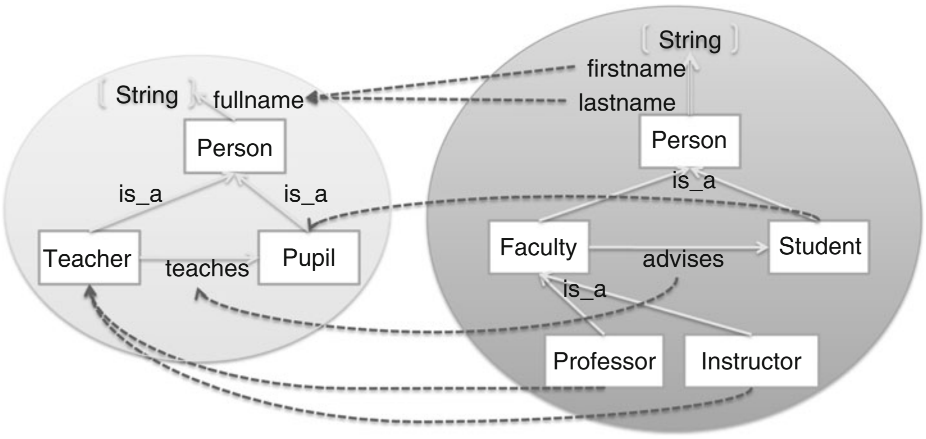


Fig. 9 *Dashed arrows* indicate mappings for merged Teacher-Student ontologies

(i.e., table) which checks to see if the other predicate is also satisfied. We generated the triggers as part of the source ontology schema as follows:

```
CREATE TRIGGER trg_fn_AFTER_INSERT AFTER INSERT ON firstname
FOR EACH ROW
trig:BEGIN
    -- enforce bridging axiom: (mysource)fn + ln -> (mytarget)n
    INSERT INTO mytarget.fullname(subject,object)
      SELECT fn.subject, concat(fn.object, ' ', ln.object)
      FROM mysource.f\/irstname fn, mysource.lastname ln
      WHERE fn.subject = ln.subject
      AND fn.subject = NEW.subject;
END trig

CREATE TRIGGER trg_ln_AFTER_INSERT AFTER INSERT ON lastname
FOR EACH ROW
trig:BEGIN
    -- enforce bridging axiom: (mysource)fn + ln -> (mytarget)n
    INSERT INTO mytarget.fullname(subject,object)
      SELECT fn.subject, concat(fn.object, ' ', ln.object)
      FROM mysource.firstname fn, mysource.lastname ln
      WHERE fn.subject = ln.subject
      AND ln.subject = NEW.subject;
END trig
```

7.3 Evaluation

Similar to what we did when testing scalability, we wrote a program which generated some uniformly distributed data instances for the source ontology. Both ontologies are small enough to have a negligible schema load time (cf. Section 4.2), taking under 100 milliseconds on average to load their respective ontology database schemas. We generated four sets of data instances under the source ontology semantics, each

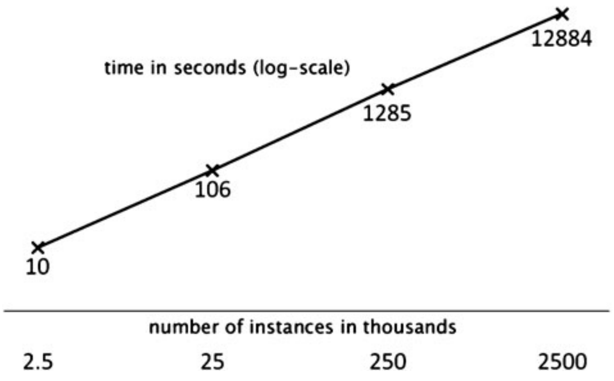


Fig. 10 Integration performance for ontology databases

dataset greater than the last by a factor of ten. Then, we measured the total time it takes to load the dataset into the source ontology database.

Because the source ontology database contains the regular ontology rules together with the bridging axioms rules, the total time measured includes three major parts: (1) the time it takes to forward propagate data within the source ontology database, (2) the time it takes to forward propagate data across to the target ontology database (via the bridging axioms), and (3) the time it takes to forward propagate data within the target ontology database. Figure 10 summaries the performance results, which shows the near-linear performance we expect from small ontologies (based on our prior load-time observations, cf. Section 4.2).

In conclusion, ontology databases using triggers works naturally for integrating distributed, heterogenous data. We can build a network of ontology databases which logically exchange data with each other using simple technologies such as message passing with persistent queues (Ceri and Widom 1993) and triggers. Furthermore, because it uses disk-based methods for forward-reasoning, OntoDB scales to much larger data sets than other (memory-based) ontology-based integration systems, such as OntoEngine (Dou and LePendu 2006), which crashes when trying to load more than 250,000 data instances (even with 8GB of RAM).

8 Conclusion

We presented ontology databases, a tool for modeling ontologies plus large numbers of instances using off-the-shelf database management systems such as MySQL. Ontology databases are useful for answering ontology-based, scientific queries that require taking the subsumption hierarchy and other constraints into account, as demonstrated by our NEMO case study. Furthermore, using our specific method—triggers—scales extremely well for small ontologies and it does well-enough for most other biomedical ontologies, including larger ones like GO. Our method pre-computes inferences for the subsumption hierarchy, so larger and deeper ontologies will incur more costly up-front penalties.

Because triggers are highly expressive, our method also works well for detecting inconsistencies that arise from instance-based data. We used not-gadgets, which reason over explicit negations, to help solve this problem. Our methods have detected

75 inconsistencies in the Gene Ontology based on GO annotation data, resulting in a regular reports to biological curators.

Finally, we extended the asynchronous, event-driven framework that we used to perform inferential data exchange. The main idea is that mapping rules between ontologies also can be implemented as trigger-rules, giving us a new, efficient and scalable way to exchange data among a network of ontology databases using inferential data integration theory.

Our next steps include studying ontology evolution and concept drift to propagate changes within an ontology database. Changes in the ontology affect the structure, rules and data for an ontology database, which makes efficiently managing the knowledge model extremely difficult. In the worst case, an ontology database is discarded and re-loaded using the new ontology—but that can take significant time for large data sets. The alternative is to surgically update the ontology database to reflect the incremental changes in the ontology. In the integration scenario, we will require retracting information across the network.

Acknowledgements This work was supported in part by grant R01 EB007684 from the National Institutes of Health. We thank Doug Howe and Jiawei Rong for their contributions on our GO and NEMO case studies. We also thank the ZFIN group, Zena M. Ariola and Gwen A. Frishkoff for their feedback on and contributions.

References

Abadi, D. J., Marcus, A., Madden, S. R., & Hollenbach, K. (2009). SW-Store: A vertically partitioned DBMS for Semantic Web data management. *VLDB Journal, 18*(2), 385–406.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., & Patel-Schneider, P. F. (Eds.) (2003). *The description logic handbook: Theory, implementation, and applications*. Cambridge University Press.

Baader, F., & Morawska, B. (2009). Unification in the description logic EL. In *Rewriting techniques and applications*.

Baader, F., & Nutt, W. (2003). Basic description logics. In *Description logic handbook* (pp. 43–95).

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). *The Semantic Web*. Scientific American.

Bernstein, P. A., & Rahm, E. (2000). Data warehouse scenarios for model management. In *ER* (pp. 1–15).

Bodenreider, O., Smith, B., Kumar, A., & Burgun, A. (2007). Investigating subsumption in snomed ct: An exploration into large description logic-based biomedical terminologies. *Artificial Intelligence in Medicine, 39*(3), 183–195.

Broekstra, J., Kampman, A., & van Harmelen, F. (2002). Sesame: A generic architecture for storing and querying RDF and RDF schema. In *International Semantic Web conference* (pp. 54–68).

Buchmann, A. P., Branding, H., Kudrass, T., & Zimmermann, J. (1992). Reach: A real-time, active and heterogeneous mediator system. *IEEE Data Engineering Bulletin, 15*(1–4), 44–47.

Bult, C. J., Eppig, J. T., Kadin, J. A., Richardson, J. E., & Blake, J. A. A. (2008). The Mouse Genome Database (MGD): Mouse biology and model systems. *Nucleic Acids Research, 36* (Database issue), D724–D728.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., & Rosati, R. (2005). DL-Lite: Tractable description logics for ontologies. In *AAAI '05: Proceedings of the 20th national conference on artificial intelligence* (pp. 602–607).

Ceri, S., Fraternali, P., Paraboschi, S., & Tanca, L. (1992). Constraint enforcement through production rules: Putting active databases at work. *IEEE Data Engineering Bulletin, 15*(1–4), 10–14.

Ceri, S., & Widom, J. (1993). Managing semantic heterogeneity with production rules and persistent queues. In *VLDB* (pp. 108–119).

Chakravarthy, S., Hanson, E. N., & Su, S. Y. W. (1992). Active data/knowledge bases research at the University of Florida. *IEEE Data Engineering Bulletin, 15*(1–4), 35–39.

Christophides, V., Karvounarakis, G., Plexousakis, D., Scholl, M., & Tourtounis, S. (2004). Optimizing taxonomic Semantic Web queries using labeling schemes. *Journal of Web Sematics, 1*, 207–228 (Elsevier).

Clark, K. L. (1977). Negation as failure. In *Logic and data bases* (pp. 293–322).

Copeland, G. P., & Khoshafian, S. N. (1985). A decomposition storage model. In *SIGMOD '85: Proceedings of the ACM SIGMOD international conference on management of data* (pp. 268–279). New York: ACM.

Curé, O., & Squelbut, R. (2005). A database trigger strategy to maintain knowledge bases developed via data migration. In *EPIA '05: Proceedings of the 12th Portuguese conference on artificial intelligence* (pp. 206–217).

Dietrich, S. W., Urban, S. D., Harrison, J. V., & Karadimce, A. P. (1992). A dood ranch at ASU: Integrating active, deductive and object-oriented databases. *IEEE Data Engineering Bulletin, 15*(1–4), 40–43.

Donini, F. M., Nardi, D., & Rosati, R. (2002). Description logics of minimal knowledge and negation as failure. *ACM Transactions on Computational Logic, 3*(2), 177–225.

Dou, D., Frishkoff, G., Rong, J., Frank, R., Malony, A., & Tucker, D. (2007). Development of NeuroElectroMagnetic Ontologies (NEMO): A framework for mining brainwave ontologies. In *Proceedings of the 13th ACM international conference on knowledge discovery and data mining (KDD)* (pp. 270–279).

Dou, D., & LePendu, P. (2006). Ontology-based integration for relational databases. In *ACM symposium on applied computing (SAC)* (pp. 461–466).

Dou, D., LePendu, P., Kim, S., & Qi, P. (2006a). Integrating databases into the Semantic Web through an ontology-based framework. In *International workshop on Semantic Web and databases (SWDB)* (p. 54). Co-located with ICDE 2006.

Dou, D., McDermott, D. V., & Qi, P. (2005). Ontology translation on the Semantic Web. *Journal of Data Semantics, 2*, 35–57.

Dou, D., Pan, J. Z., Qin, H., & LePendu, P. (2006b). Towards populating and querying the Semantic Web. In *International workshop on scalable Semantic Web knowledge base systems (SSWS)* (pp. 129–142). Co-located with ISWC 2006.

Frishkoff, G., LePendu, P., Frank, R., Liu, H., & Dou, D. (2009). Development of Neural Electromagnetic Ontologies (NEMO): Ontology-based tools for representation and integration of event-related brain potentials. In *ICBO '09: Proceedings of the international conference on biomedical ontology* (pp. 31–34).

Frishkoff, G. A. (2007). Hemispheric differences in strong versus weak semantic priming: Evidence from event-related brain potentials. *Brain and Language, 100*(1), 23–43.

Gallaire, H., Minker, J., & Nicolas, J.-M. (1977). *Logic and data bases*. New York, NY, USA: Association for Computing Machinery.

Gallaire, H., & Nicolas, J.-M. (1990). Logic and databases: An assessment. In *ICDT* (pp. 177–186).

Gene Ontology Consortium (2000). Gene Ontology: Tool for the unification of biology. *Nature Genetics, 25*, 25–29.

Gene Ontology Consortium (2006). The Gene Ontology (GO) project in 2006. *Nucleic Acids Research, 34* (Database issue), D322–D326.

Goble, C., & Stevens, R. (2008). State of the nation in data integration for bioinformatics. *Journal of Biomedical Informatics, 41*(5), 687–693.

Guarino, N. (1998). Formal ontology in information systems. In *International conference on formal ontology in information systems*.

Guo, Y., Pan, Z., & Heflin, J. (2004). An evaluation of knowledge base systems for large OWL datasets. In *ISWC '04: Proceedings of the international Semantic Web conference* (pp. 274–288).

Guo, Y., Pan, Z., & Heflin, J. (2005). LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics, 3*(2–3), 158–182.

Haarslev, V., & Möller, R. (2001). High performance reasoning with very large knowledge bases: A practical case study. In *IJCAI '01: Proceedings of the international joint conferences on artificial intelligence* (pp. 161–168).

Hill, D. P., Smith, B., McAndrews-Hill, M. S., & Blake, J. A. (2008). Gene Ontology annotations: What they mean and where they come from. *BMC Bioinformatics, 9*(5), S2.

Horrocks, I., Li, L., Turi, D., & Bechhofer, S. (2004). The instance store: DL reasoning with large numbers of individuals. In *Description logics*.

Imieliński, T., & Lipski, W. Jr. (1984). Incomplete information in relational databases. *Journal of the ACM, 31*(4), 761–791.

Jarke, M., Gallersdörfer, R., Jeusfeld, M. A., & Staudt, M. (1995). ConceptBase—A deductive object base for meta data management. *Journal of Intelligence and Information Systems, 4*(2), 167–192.

Kolaitis, P. G. (2005). Schema mappings, data exchange, and metadata management. In *PODS '05* (pp. 61–75). New York: ACM.

Kowalski, R. A., Sadri, F., & Soper, P. (1987). Integrity checking in deductive databases. In *VLDB* (pp. 61–69).

Lenzerini, M. (2002). Data integration: A theoretical perspective. In *PODS '02* (pp. 233–246). New York: ACM.

LePendu, P., Dou, D., Frishkoff, G. A., & Rong, J. (2008). Ontology database: A new method for semantic modeling and an application to brainwave data. In *SSDBM '08: Proceedings of the international conference on statistical and scientific database management* (pp. 313–330).

LePendu, P., Dou, D., & Howe, D. (2009). Detecting inconsistencies in the gene ontology using ontology databases with not-gadgets. In *ODBASE '09: Proceedings of the international conference on ontologies, databases and application of semantics* (pp. 948–965).

Levesque, H. J., & Lakemeyer, G. (2001). *The logic of knowledge bases*. Boston, MA, USA: MIT Press.

Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., & Liu, S. (2006). Towards a complete OWL ontology benchmark. In *European sem. web conf. (ESWC)* (pp. 125–139).

Motik, B., Horrocks, I., & Sattler, U. (2007). Bridging the gap between OWL and relational databases. In *WWW 07': Proceedings of the 16th international conference on World Wide Web* (pp. 807–816).

Neumann, T., & Weikum, G. (2009). Scalable join processing on very large RDF graphs. In *SIGMOD '09: Proceedings of the ACM SIGMOD international conference on management of data* (pp. 627–640).

Noy, N., Shah, N., Whetzel, P., Dai, B., Dorf, M., Griffith, N., et al. (2009). BioPortal: Ontologies and integrated data resources at the click of a mouse. *Nucleic Acids Research, 1*(37), W372–W376.

O'Connor, M. J., & Das, A. K. (2008). SQWRL: A query language for OWL. In *OWLED* (Vol. 529). CEUR-WS.org.

Qin, H., Dou, D., & LePendu, P. (2007). Discovering executable semantic mappings between ontologies. In *Proceedings of the international conference on ontologies, databases and application of semantics* (pp. 832–849).

Racunas, S. A., Shah, N. H., Albert, I., & Fedoroff, N. V. (2004). Hybrow: A prototype system for computer-aided hypothesis evaluation. In *ISMB/ECCB (supplement of bioinformatics)* (pp. 257–264).

Reiter, R. (1977). Deductive question-answering on relational data bases. In *Logic and data bases* (pp. 149–177).

Reiter, R. (1992). What should a database know? *Journal of Logic Programming, 14*(1&2), 127–153.

Shah, N., Jonquet, C., Chiang, A., Butte, A., Chen, R., & Musen, M. (2009). Ontology-driven indexing of public datasets for translational bioinformatics. *BMC Bioinformatics, 10*, S1.

Sheth, A. P., & Larson, J. A. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys, 22*(3), 183–236.

Sprague, J., Westerfield, M., et al. (2007). The zebrafish information network: The zebrafish model organism database provides expanded support for genotypes and phenotypes. *Nucleic Acids Research, 36*, D768–D772.

Tempich, C., & Volz, R. (2003). Towards a benchmark for Semantic Web reasoners—An analysis of the DAML ontology library. In *Evaluation of ontology-based tools wkshp. (ISWC)*.

Ullman, J. D. (1988). *Principles of database and knowledge-base systems* (Vol. I). New York, NY, USA: Computer Science Press.

Vasilecas, O., & Bugaite, D. (2007). An algorithm for the automatic transformation of ontology axioms into a rule model. In *CompSysTech '07: Proceedings of the international conference on computer systems and technologies* (pp. 1–6). New York: ACM.

Vieille, L., Bayer, P., Küchenhoff, V., Lefebvre, A., & Manthey, R. (1992). The EKS-V1 system. In *LPAR '92: Proceedings of the international conference on logic programming and automated reasoning* (pp. 504–506). London: Springer.

Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., et al. (2001). Ontology-based integration of information—A survey of existing approaches. In H. Stuckenschmidt (Ed.), *IJCAI '01: Workshop on ontologies and information sharing* (pp. 108–117).

Wang, S., Guo, Y., Qasem, A., & Heflin, J. (2005). Rapid benchmarking for Semantic Web knowledge base systems. In *Int'l sem. web conf. (ISWC)* (pp. 758–772).