Department of Computer & Information Science

Technical Reports (CIS)

University of Pennsylvania *Year* 2007

Integrating Ontologies and Relational Data

Sören Auer* Zachary G. Ives†

*University of Pennsylvania, auer@seas.upenn.edu

†University of Pennsylvania, zives@cis.upenn.edu

This paper is posted at ScholarlyCommons.

http://repository.upenn.edu/cis_reports/716

Integrating Ontologies and Relational Data

Sören Auer
Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104, USA
auer@seas.upenn.edu

Zachary Ives
Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104, USA
zives@cis.upenn.edu

ABSTRACT

In recent years, an increasing number of scientific and other domains have attempted to standardize their terminology and provide reasoning capabilities through ontologies, in order to facilitate data exchange. This has spurred research into Web-based languages, formalisms, and especially query systems based on ontologies.

Yet we argue that DBMS techniques can be extended to provide many of the same capabilities, with benefits in scalability and performance. We present OWLDB, a lightweight and extensible approach for the integration of relational databases and description logic based ontologies. One of the key differences between relational databases and ontologies is the high degree of implicit information contained in ontologies. OWLDB integrates the two schemes by codifying ontologies' implicit information using a set of sound and complete inference rules for $\mathcal{SHOIN}$ – the description logic behind OWL ontologies. These inference rules can be translated into queries on a relational DBMS instance, and the query results (representing inferences) can be added back to this database. Subsequently, database applications can make direct use of this inferred, previously implicit knowledge, e.g., in the annotation of biomedical databases. As our experimental comparison to a native description logic reasoner and a triple store shows, OWLDB provides significantly greater scalability and query capabilities, without sacrificing performance with respect to inference.

1. INTRODUCTION

The problem of encoding information in a standardized way has been a challenge addressed in different ways by different communities. In the database world, entity-relationship models, inheritance, and declarative views are the basic mechanisms for expressing concepts and their relationships. An alternative approach has been adopted by the knowledge representation community, namely ontologies and description logics. In contrast to database formalisms, which focus on a subset of first-order logic that scales to large numbers of facts by running in polynomial data complexity, the description logics used to define ontologies are a **different** subset that allows richer specification of classes and relationships, but is less computationally tractable. Both approaches have a role in encoding and sharing information today, and in fact the two worlds are increasingly being interconnected: Bioinformatics data is generally represented in a variety of databases, which categorize the data entries by referencing classes in the Open Biomedical Ontologies (OBO) [25]; anatomical measurements are stored in databases that then reference the Foundational Model of Anatomy [22]; products in e-commerce reference categories in eClassOWL [15]; personal information management systems need class and relationship information such as that in FOAF [9]. Hence, ontology and database integration is already a problem today — it will be even more so if the Semantic Web is to succeed today's Web.

As we begin to integrate systems based on these disparate formalisms, a natural question arises about which architecture to use: a mediator that partitions computations across independent database and ontology-based systems, a single system that extends a description logics engine to include database query capabilities, or a database engine extended with certain description logics reasoning capabilities. Our thesis in this paper is that the third approach — a single reasoning engine that builds upon database query processing capabilities — is the most useful, as we explain and then validate.

Systems developed for description logic reasoning focus mainly on three basic capabilities:

- *subsumption reasoning*, i.e. the inference of implicit subclass-superclass relationships,
- *satisfiability and consistency checks*, i.e. whether a class can potentially have instances; if not, whether there is a contradiction with explicitly defined ones,
- *classification of objects*, i.e. the determination of class membership for objects or the retrieval of all instances of a certain class.

They are generally optimized for fairly complex reasoning about relatively few facts.

Unfortunately, with the adoption of ontologies in science, we are increasingly encountering the challenges of scale, both in terms of large amount of instance data and large numbers of concepts. For instance, the WordNet linguistic ontology includes 100,000 concepts and the NCI cancer ontology has 30,000 classes; biomedical data from different sources

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$5.00.

adopt a variety of ontologies (e.g., OBO [25], FMA [22], MGED [33]). In contrast to the assumptions of description logics reasoners, these ontologies have large, relatively simple terminological parts (i.e., concept and role definitions), and an even larger number of data instances. Ontologies such as the NCI cancer ontology exceed the capabilities of most current description logics reasoners. This inadequacy has been observed by a number of authors (e.g. [37, 30, 16]).

Some efforts have been made to employ databases to perform limited tasks to assist description logics reasoners (e.g. [8, 6]), using the reasoner to perform reasoning about classes and roles and the DBMS to reason about data. However, it has been shown that the full power of certain common description logics can be expressed using disjunctive datalog [17], and we argue that the best approach in many settings is to perform almost all of the reasoning in the DBMS, thus avoiding the overhead implicit in coupling systems. We exploit the fact that most large real-world ontologies do not use arbitrarily complex and unique representation methods for every single class definition: rather, they define many similar classes that can be reasoned about "in bulk," and the strengths of DBMS query processing are more suited for such tasks. This is especially useful because most description logics reasoners already use database technology to store data instances, and since our goal is to combine ontology and database data under a single query interface.

We make the following contributions:

- a methodology for translating $\mathcal{SHOIN}$ inference rules into relational database queries,
- a database schema and encoding scheme for high performance inferencing by means of the derived queries,
- stratification policies for the OWL inference rules, allowing for efficient execution within a fixpoint algorithm executing in middleware above a DBMS.

This paper is structured as follows. We provide an overview of the description logic $\mathcal{SHOIN}$ in Section 2. Section 3 describes the concept of description logic reasoning by means of inference rules and how suitable inference rules can be obtained. Section 4 presents our scheme for reasoning (inference) using a "thin" middleware layer over an SQL DBMS. We experimentally validate our scheme in Section 5, discuss related work in Section 6, and conclude in Section 7.

2. PRELIMINARIES

Web Ontology Language. The Web Ontology Language OWL is a semantic markup language for publishing and sharing ontologies on the World Wide Web [5]. It is an established standard and widely used in applications in science and increasingly often also in industry. OWL is developed as a vocabulary extension of RDF (the Resource Description Framework) [19]. Hence, OWL ontologies are encoded as statements adhering to the subject-predicate-object data model of RDF (see the OWL/XML example column in Table 1). OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full. While OWL Lite is not expressive enough for many applications, OWL Full (the most expressive sublanguage) provides no computational guarantees. However, OWL DL (where DL stands for Description Logic) provides high expressiveness while retaining computational tractability.

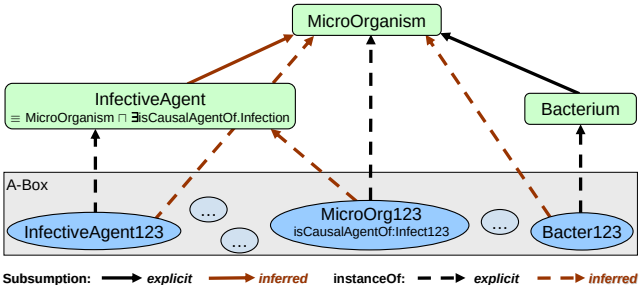

Figure 1: Subsumption and classification in a biomedical ontology.

Description logic. Description logics are decidable fragments of first order logic. They represent knowledge in terms of *objects*, *concepts*, and *roles*. Objects correspond to constants, concepts to unary predicates, and roles to binary predicates in first order logic. In Description Logic systems information is stored in a *knowledge base*, which is a set of axioms. It is divided in two parts: *TBox* and *ABox*. The ABox contains *assertions* about objects. It relates objects to concepts and roles. The TBox describes the *terminology* by relating concepts and roles.

The description logic $\mathcal{SHOIN}$. We briefly introduce the $\mathcal{SHOIN}$ description logic, which is the base language for OWL DL ontologies and refer to [3] for further background on description logics. The syntax and semantics of $\mathcal{SHOIN}$ concept constructors is shown in Table 1. As usual in logics, interpretations are used to assign a meaning to syntactic constructs. An interpretation I consists of a non-empty interpretation domain Δ^I and an interpretation function $\cdot^I$, which assigns to each object o (symbolized by ovals in the example depicted in Figure 1) an element of Δ^I, to each concept name c (symbolized by squares) a set $c^I \subseteq \Delta^I$, and to each role r (symbolized by arrows) a binary relation $r^I \subseteq \Delta^I \times \Delta^I$. Interpretations are extended from elements to concepts as shown in Table 1, and to other elements of a knowledge base in a straightforward way. An interpretation that satisfies an axiom (set of axioms) is called a model of this axiom (set of axioms).

Inference tasks. Inference algorithms extract implicit knowledge from a given knowledge base. Standard reasoning tasks include instance checks, consistency checks and subsumption. We explicitly define the last: Let c_1, c_2 be concepts and T a TBox. c_1 is subsumed by c_2, denoted by $c_1 \sqsubseteq c_2$, iff for any interpretation I we have $c_1^I \subseteq c_2^I$. c_1 is subsumed by c_2 with respect to T (denoted by $c_1 \sqsubseteq_T c_2$) iff for any model I of T we have $c_1^I \subseteq c_2^I$. c_1 is equivalent to c_2 (with respect to T), denoted by $c_1 \equiv c_2$ ($c_1 \equiv_T c_2$), iff $c_1 \sqsubseteq c_2$ ($c_1 \sqsubseteq_T c_2$) and $c_2 \sqsubseteq c_1$ ($c_2 \sqsubseteq_T c_1$).

Computing the subsumption relation between concepts is a key to render other reasoning services as well:

- *Instance checks* (used for query answering): if c_1 is subsumed by c_2, all instances of c_1 will be also instantiations of c_2.
- *Consistency*: a knowledge base is inconsistent, if a class c is known to be subsumed by the bottom concept $\perp$ and contains an instance.

Example. The reasoning tasks subsumption and instance check and their interrelation is illustrated in Figure 1 with

Constructor name	Syntax	Semantics	OWL/RDF example		
atomic concept	a	$a^I \subseteq \Delta^I$			
role	r	$r^I \subseteq \Delta^I \times \Delta^I$			
individual	o	$o^I \in \Delta^I$			
top concept	$\top$	$\top^I = \Delta^I$	`owl:Thing`		
bottom concept	$\bot$	$\bot^I = \emptyset$	`owl:Nothing`		
inverse role	r^-	$(r^-)^I = (r^I)^-$	`r1`	`owl:inverseOf`	`r2`
conjunction	$c_1 \sqcap c_2$	$(c_1 \sqcap c_2)^I = c_1^I \cap c_2^I$	`c`	`owl:intersectionOf`	`rdflist`
disjunction	$c_1 \sqcup c_2$	$(c_1 \sqcup c_2)^I = c_1^I \cup c_2^I$	`c`	`owl:unionOf`	`rdflist`
negation	$\neg c$	$(\neg c)^I = \Delta^I \setminus c^I$	`c1`	`owl:complementOf`	`c`
oneOf	$\{o_1, \dots\}$	$(\{o_1, \dots\})^I = \{o_1^I, \dots\}$	`c`	`owl:oneOf`	`rdflist`
exists restriction	$\exists r.c$	$(\exists r.c)^I = \{x \mid \exists y : \langle x, y\rangle \in r^I \text{ and } y \in c^I\}$	`c1`	`owl:someValuesFrom`	`c`
value restriction	$\forall r.c$	$(\forall r.c)^I = \{x \mid \exists y : \langle x, y\rangle \in r^I \rightarrow y \in c^I\}$	`c1`	`owl:allValuesFrom`	`c`
atleast restriction	$\geq nr$	$(\geq nr)^I = \{x \mid \#(\{y : \langle x, y\rangle \in r^I\}) \geq n\}$	`c`	`owl:minCardinality`	`n`
atmost restriction	$\leq nr$	$(\leq nr)^I = \{x \mid \#(\{y : \langle x, y\rangle \in r^I\}) \leq n\}$	`c`	`owl:maxCardinality`	`n`
Axiom Name	**Syntax**	**Semantics**	**OWL/RDF example**		
concept inclusion	$c_1 \sqsubseteq c_2$	$c_1^I \subseteq c_2^I$	`c1`	`rdfs:subClassOf`	`c2`
role inclusion	$r_1 \sqsubseteq r_2$	$r_1^I \subseteq r_2^I$	`r1`	`rdfs:subPropertyOf`	`r2`
role transitivity	$r \equiv r^+$	$r^I = (r^I)^+$	`r`	`rdf:type`	`owl:TransitiveProperty`
role symmetry	$r \equiv r^-$	$r^I = (r^I)^-$	`r`	`rdf:type`	`owl:symmetricProperty`
individual inclusion	$o : c$	$o^I \in C^I$	`o`	`rdf:type`	`c`
individual equality	$o_1 = o_2$	$o_1^I = o_2^I$	`o1`	`owl:sameAs`	`o2`
individual inequality	$o_1 \neq o_2$	$o_1^I \neq o_2^I$	`o1`	`owl:differentFrom`	`o2`

Table 1: Syntax and semantics of $\mathcal{SHOIN}$ and RDF examples (we use the common RDF, RDF-Schema and OWL namespace prefixes and assume a default namespace defined; rdflist refers to an RDF resource representing an RDF list containing the constituents of the union, intersection or exhaustive enumeration).

an example from the Galen medical terminology ontology [31]. After inferring that InfectiveAgent is subsumed by the class MicroOrganism the instance InfectiveAgent123 will also instantiate MicroOrganism. A similar inference can be made for the instance Bacter123, however, here the subsumption relation between Bacterium and MicroOrganism is already explicit. The conclusion that MicroOrg123 is an instance of InfectiveAgent, on the other hand, can not be reduced to subsumption, but can be decided by checking the membership of MicroOrg123 with respect to every constituent of the intersection serving as a definition for InfectiveAgent.

3. DERIVING INFERENCE RULES FOR ONTOLOGIES

Description logic reasoners approach the problem of deciding subsumption between two classes by reducing the problem to a consistency check: For classes c_1 and c_2, $c_1 \sqsubseteq c_2$ can be decided by checking the consistency of $KB = \{o : c, c \equiv c_1 \sqcap \neg c_2\}$. Consistency is often decided by trying to construct a model by means of so called tableau algorithms [4]. This method is time-tested and has proven to be sound and complete for most description logics. However, in reasoning about subsumption with many classes, it requires high numbers of computation steps, and worse, large in-memory state. In order to scale to such ontologies, we instead execute inference rules for description logics by codifying them as relational database queries.

It is easy to intuitively derive *some* sound inference rules from the set based semantics of the $\mathcal{SHOIN}$ class and property constructors or axioms as shown in Table 1. For example, from the definition of a class c to contain exactly the instances within the intersection of classes c_1 and c_2 (i.e. $c = c_1 \sqcap c_2$), we can easily infer that c is a subclass of c_1 as well as of c_2 ($c \sqsubseteq c_1$ and $c \sqsubseteq c_2$). Similarly, from class definitions $Parent = (\geq 1 hasChild)$ and $LuckyParent = (\geq 2 hasChild)$ we can infer $LuckyParent \sqsubseteq Parent$.

However, in addition to deriving sound inference rules it is crucial to describe the fragment of a description logic covered by the inference rules, i.e., the fragment for which completeness of the inferences can be guaranteed. Not surprisingly, a great deal of theoretical work has been done in these areas. We base our OWLDB implementation on the theoretical foundations of [28] and the KIT report 111 [29] (both by Royer and Quantz). Royer and Quantz systematically derive a complete set of inference rules for a description logic that happens to subsume $\mathcal{SHOIN}$(the description logic behind OWL); their work is purely from a definitional perspective and does not consider an implementation. They develop sound and complete Sequent Calculi axiomatizations of First-Order Logic without (FOL) or with Equality (FOL=). A general methodology for obtaining sound and complete inference systems for any logical language L is to translate the formulae of L (L-formulae) into first-order formulae (provided L is translatable into FOL), and then to identify necessary and sufficient conditions of provability in Sequent Calculi of the FOL translations from the formulae encoding L-formulae. The extensive analysis of Sequent Calculi proofs by Royer and Quantz resulted in a set of 168 inference rules. However, it is important to note that the DL consider in the KIT report differs from $\mathcal{SHOIN}$, in the following ways:

- It supports the *complex role constructors*: $r_1 \sqcup r_2$ (role union), $r_1 \sqcap r_2$ (role intersection), $r_1 * r_2$ (role composition), $\neg r$ role complement. To close the constructors

with respect to union and intersection the DL also includes a top ($\top_r$) and bottom role ($\bot^r$).

- It supports *qualified cardinality restrictions* (notation ($\geq nr.c$) and ($\leq nr.c$)), stating that objects belonging to the restriction must have a maximum or minimum number of role values of a certain type. $\mathcal{SHOIN}$, on the other hand, supports only unqualified number restrictions (($\geq nr$) and ($\leq nr$)), where the type of the role values is not important.

- A *class restriction 'fills'* (notation $r.\{o_1, \ldots, o_n\}$) is supported, requiring all instances of the restriction to be related to all the values of the filler ($\{o_1, \ldots, o_n\}$). In the special case that the filler contains just one object (i.e. $r.\{o\}$) the restriction is equivalent to the exists restriction $\exists r.\{o\}$.

- The *domain (or range) of roles* is defined by restricting a general role r to a role ${}_c|r$ (or $r|_c$), whose domain (or range) contains exactly only instances of c. In $\mathcal{SHOIN}$ on the other hand a domain (or range) definition for a role is 'emulated' by an axiom $(\geq 1r) \sqsubseteq c$ (or $\top \sqsubseteq \forall r.c$).

Hence, the DL regarded in the KIT report clearly subsumes $\mathcal{SHOIN}$ and to obtain a subset of inference rules for $\mathcal{SHOIN}$ we can (a) omit the KIT rules dealing with complex role constructors as well as domain and range restrictions (since they are not derivable for $\mathcal{SHOIN}$), (b) derive rules with unqualified cardinality restrictions from the ones with qualified cardinality restrictions by replacing the qualification class with the top concept ($\top$) and (c) replace 'fills' restrictions in rules with exists restrictions. As a consequence of the simplification in (b) and (c) some rules (such as rule 48 in the KIT report) turn out to be redundant and can be omitted. This reduces the 168 KIT rules to 66, as shown in Figures 2 and 3. Rules 1-47 correspond to the rules with the same numbers in the KIT report. As a result of the systematic and complete derivation of inference rules in the KIT report [29] and the rule adaptation to $\mathcal{SHOIN}$ we obtain the following proposition:

Proposition 1. *The inference rules 1-55 (as shown in Figures 2 and 3) are sound and complete with respect to class and role subsumption reasoning in $\mathcal{SHOIN}$.*

Proof. The KIT rules are derived by translating all possible subsumption and instance recognition formulae of the KIT description logic into first-order formulae, and then to identify necessary and sufficient conditions of provability in Sequent Calculi of the FOL translations. This method is clearly monotonic, i.e. a subsumed, less expressive description logic will result in a subset of possible subsumption and instance recognition formulae and consequently a subset of derivable inference rules. Hence, it is safe to omit KIT rules containing language constructs which are not available in $\mathcal{SHOIN}$ and reduce the more general language constructs (qualified number restrictions and fills) to the specialized equivalents in $\mathcal{SHOIN}$:

Rules 1-28 are the same as in the KIT report. Rules 29-47 are derived by replacing qualified cardinality restrictions with unqualified cardinality restrictions. After doing so in KIT rule 48, this rule turns out to be equivalent to rule 38. KIT rules 51, 52 and 55, 56, 101, 103, 117 are the same

Conjunction and Disjunction

$$\rightarrow c \sqsubseteq c \quad (1)$$
$$\rightarrow c_1 \sqcap c_2 \sqsubseteq c_1 \sqcup c_3 \quad (2)$$
$$c_1 \sqsubseteq c_2 \rightarrow c_1 \sqcap c_3 \sqsubseteq c_2 \quad (3)$$
$$c_1 \sqsubseteq c_2 \rightarrow c_1 \sqsubseteq c_2 \sqcup c_3 \quad (4)$$
$$c_1 \sqsubseteq c_2, c_1 \sqsubseteq c_3 \leftrightarrow c_1 \sqsubseteq c_2 \sqcap c_3 \quad (5)$$
$$c_2 \sqsubseteq c_1, c_3 \sqsubseteq c_1 \leftrightarrow c_2 \sqcup c_3 \sqsubseteq c_1 \quad (6)$$
$$c_1 \sqsubseteq c_2, c_2 \sqcap c_3 \sqsubseteq c_4 \rightarrow c_1 \sqcap c_3 \sqsubseteq c_4 \quad (7)$$
$$\rightarrow \bot \sqsubseteq c \quad (8)$$
$$\rightarrow c \sqsubseteq \top \quad (9)$$

Quantification

$$\top \sqsubseteq c \leftrightarrow \top \sqsubseteq \forall r.c \quad (10)$$
$$c \sqsubseteq \bot \leftrightarrow \exists r.c \sqsubseteq \bot \quad (11)$$
$$c_1 \sqsubseteq c_2, r_2 \sqsubseteq r_1 \rightarrow \forall r_1.c_1 \sqsubseteq \forall r_2.c_2 \quad (12)$$
$$c_1 \sqsubseteq c_2, r_1 \sqsubseteq r_2 \rightarrow \exists r_1.c_1 \sqsubseteq \exists r_2.c_2 \quad (13)$$
$$c_2 \sqcap c_1 \sqsubseteq \bot, r_1 \sqsubseteq r_2 \rightarrow \forall r_2.c_2 \sqcap \exists r_1.c_1 \sqsubseteq \bot \quad (14)$$
$$\top \sqsubseteq c_2 \sqcup c_1, r_1 \sqsubseteq r_2 \rightarrow \top \sqsubseteq \exists r_2.c_2 \sqcup \forall r_1.c_1 \quad (15)$$
$$c_1 \sqcap c_2 \sqsubseteq c_3, r_3 \sqsubseteq r_1, r_3 \sqsubseteq r_2 \rightarrow \forall r_1.c_1 \sqcap \forall r_2.c_2 \sqsubseteq \forall r_3.c_3 \quad (16)$$
$$c_3 \sqsubseteq c_1 \sqcup c_2, r_3 \sqsubseteq r_1, r_3 \sqsubseteq r_2 \rightarrow \exists r_3.c_3 \sqsubseteq \exists r_1.c_1 \sqcup \exists r_2.c_2 \quad (17)$$
$$c_1 \sqcap c_2 \sqsubseteq c_3, r_2 \sqsubseteq r_1, r_2 \sqsubseteq r_3 \rightarrow \forall r_1.c_1 \sqcap \exists r_2.c_2 \sqsubseteq \exists r_3.c_3 \quad (18)$$
$$c_3 \sqsubseteq c_1 \sqcup c_2, r_2 \sqsubseteq r_1, r_2 \sqsubseteq r_3 \rightarrow \forall r_3.c_3 \sqsubseteq \exists r_1.c_1 \sqcup \forall r_2.c_2 \quad (19)$$

Negation

$$\rightarrow c \sqcap \neg c \sqsubseteq \bot \quad (20)$$
$$\rightarrow \top \sqsubseteq c \sqcup \neg c \quad (21)$$
$$c_1 \sqcap c_2 \sqsubseteq c_3 \leftrightarrow c_1 \sqsubseteq c_3 \sqcup \neg c_2 \quad (22)$$
$$c_1 \sqsubseteq c_2 \sqcup c_3 \leftrightarrow c_1 \sqcap \neg c_2 \sqsubseteq c_3 \quad (23)$$
$$\rightarrow \neg\neg c \equiv c \quad (24)$$
$$\rightarrow \neg(c_1 \sqcap c_2) \equiv \neg c_1 \sqcup \neg c_2 \quad (25)$$
$$\rightarrow \neg(c_1 \sqcup c_2) \equiv \neg c_1 \sqcap \neg c_2 \quad (26)$$
$$\rightarrow \neg\exists r.c \equiv \forall r.\neg c \quad (27)$$
$$\rightarrow \neg\forall r.c \equiv \exists r.\neg c \quad (28)$$

Cardinality Restrictions

$$c \sqsubseteq \bot \rightarrow \top \sqsubseteq (\leq nr) \quad (29)$$
$$c \sqsubseteq \bot \rightarrow (\geq nr) \sqsubseteq \bot \quad (30)$$
$$\rightarrow (\geq 1r) \equiv \exists r.\top \quad (31)$$
$$\rightarrow (\geq 1r) \equiv \neg\forall r.\bot \quad (32)$$
$$\rightarrow (\leq 0r) \equiv \forall r.\bot \quad (33)$$
$$\rightarrow (\leq 0r) \equiv \neg\exists r.\top \quad (34)$$
$$m = n - 1 \rightarrow \neg(\geq nr) \equiv (\leq mr) \quad (35)$$
$$m = n + 1 \rightarrow \neg(\leq nr) \equiv (\geq mr) \quad (36)$$
$$q \leq p, r_1 \sqsubseteq r_2 \rightarrow (\geq pr_1) \sqsubseteq (\geq qr_2) \quad (37)$$
$$q < p, r_1 \sqsubseteq r_2 \rightarrow (\leq qr_2) \sqcap (\geq pr_1) \sqsubseteq \bot \quad (38)$$
$$q < p, r_1 \sqsubseteq r_2 \rightarrow \top \sqsubseteq (\geq qr_2) \sqcup (\leq pr_1) \quad (39)$$
$$p \leq q, r_2 \sqsubseteq r_1 \rightarrow (\leq pr_1) \sqsubseteq (\leq qr_2) \quad (40)$$
$$r_1 \sqsubseteq r_2 \rightarrow (\geq nr_1) \sqcap \forall r_2.\bot \sqsubseteq \bot \quad (41)$$
$$q < p, r_1 \sqsubseteq r_2, r_1 \sqsubseteq r_3 \rightarrow (\leq qr_2) \sqcap (\geq pr_1) \sqcap \forall r_3.c \sqsubseteq \bot \quad (42)$$
$$q \leq p, r_1 \sqsubseteq r_2, r_1 \sqsubseteq r_3 \rightarrow (\geq pr_1) \sqcap \forall r_3.c \sqsubseteq (\geq qr_2) \quad (43)$$
$$q \leq p, r_1 \sqsubseteq r_2, r_1 \sqsubseteq r_3 \rightarrow (\leq qr_2) \sqcap \forall r_3.c \sqsubseteq (\leq pr_1) \quad (44)$$
$$r_1 \sqsubseteq r_2 \rightarrow (\geq nr_1) \sqcap (\leq nr_2) \sqsubseteq \top \quad (45)$$
$$\sum_{i=1}^{i=k}(p_i) < n, r \sqsubseteq r_i \rightarrow (\geq nr) \sqcap (\leq p_1 r_1) \sqcap \ldots \sqcap (\leq p_k r_k) \sqsubseteq \bot \quad (46)$$
$$r \sqsubseteq r_1, r \sqsubseteq r_2 \rightarrow (\leq pr_1) \sqcap (\leq qr_2) \sqsubseteq (\leq p{+}qr) \quad (47)$$

OneOf

$$S \subseteq T \rightarrow S \sqsubseteq T \quad (48)$$
$$\rightarrow S \sqcap T \equiv \{S \cap T\} \quad (49)$$
$$n \geq |S|, r_2 \sqsubseteq r_1 \rightarrow \forall r_1.S \sqsubseteq (\leq nr_2) \quad (50)$$

Figure 2: $\mathcal{SHOIN}$ class subsumption inference rules.

Role Subsumption and Inversion

$$\rightarrow r \sqsubseteq r \quad (51)$$
$$r_1^- \sqsubseteq r_2 \rightarrow r_1 \sqsubseteq r_2^- \quad (52)$$
$$\rightarrow r^{--} \equiv r \quad (53)$$
$$r_1 \sqsubseteq r_2^- \rightarrow r_1^- \sqsubseteq r_2 \quad (54)$$
$$r_1 \sqsubseteq r_2 \rightarrow r_1^- \sqsubseteq r_2^- \quad (55)$$

Instance recognition

$$a : c_1, c_1 \sqsubseteq c_2 \rightarrow a : c_2 \quad (56)$$
$$\rightarrow a : \top \quad (57)$$
$$a : c_1, a : c_2 \leftrightarrow a : c_1 \sqcap c_2 \quad (58)$$
$$a \in S \rightarrow a : S \quad (59)$$
$$a \notin S \rightarrow a : \neg S \quad (60)$$
$$a_1 : \forall r.c, a_1 : r.\{a_2\} \rightarrow a_2 : c \quad (61)$$
$$a_1 : \forall r.c, a_2 : \neg c \rightarrow a_1 : \neg(r.\{a_2\}) \quad (62)$$
$$a_1 : r.\{a_2\}, a_2 : c \rightarrow a_1 : \exists r.c \quad (63)$$
$$a_1 : r.\{a_2\}, \top \sqsubseteq \forall r.c \rightarrow a_2 : c \quad (64)$$
$$a_1 : r.\{a_2\}, (\geq 1r) \sqsubseteq c \rightarrow a_1 : c \quad (65)$$
$$a_1 : r.\{a_2\} \rightarrow a_2 : (r^- : a_1) \quad (66)$$

Figure 3: $\mathcal{SHOIN}$ role subsumption, inversion and instance recognition inference rules.

as 48, 49 and 50, 51, 52, 54, 55. KIT rules 102, 104 are equivalent to rule 53. The following KIT rules will not be derivable for $\mathcal{SHOIN}$: 49, 57-100, 131-156 (due to usage of complex roles constructors), 50, 53, 54 (qualified cardinality restriction which can not be replaced by top concept, since the qualification class is required to be an oneOf class). □

However, the rules are incomplete with respect to instance recognition, and consequently also not sufficient to detect inconsistencies with respect to objects being instances of unsatisfiable classes. The incompleteness with respect to instance recognition is due to missing rules for case reasoning. Case reasoning would be required to handle rules with disjunctions in the consequent, such as:

$$o : \forall r.\{o_1, \ldots, o_n\}, o : \exists r.c \rightarrow o_1 : c \vee \cdots \vee o_n : c$$

Because by the notorious complexity of deriving case reasoning rules and standard databases being ill-suited to handle such disjunctions efficiently we decided not to extend our approach in that direction.

4. ONTOLOGY REASONING WITH A RELATIONAL DATABASE

In order to exploit the indexing and query optimization techniques available in relational databases for the execution of the inference rules we present in this section a storage schema for OWL ontologies optimized for inference rule execution, we demonstrate how the inference rules can be translated into queries on this schema and exhibit an algorithm for efficiently executing these queries.

4.1 Ontology Storage in Relational Databases

Due to the standardized representation of OWL ontologies in RDF, which consists of subject-object-predicate tuples, it is common to store OWL ontologies within a ternary relational DBMS table. (See Table 1 to see how $\mathcal{SHOIN}$ axioms can be represented as RDF statements). A variety of different relational schema to store such RDF statements has

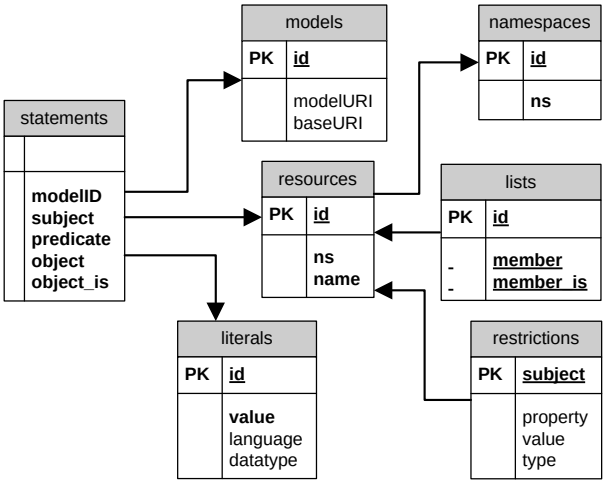

Figure 4: Database schema for RDF storage, optimized for the execution of inference rules.

been proposed. The central element of most of these is a table capturing the subject-predicate-object statements and optionally additional tables for normalization, replication, caching purposes. During our extensive experiments with OWLDB, we developed a scheme that sped up queries and insertions by de-normalizing several relations, with 30-50% increased storage space requirements. OWLDB makes inferences by executing queries and making insertions into tables; hence insertion performance is the key bottleneck. Insertions over better-normalized schemes are comparably slow due to checks for existence of identifiers in the normalization tables.

Although the presented approach does not require a specific relational schema (whether normalized or unnormalized), the one depicted in Figure 4 performed best on our tests. In this database schema, each RDF triple corresponds to a row in the table `statements` referencing RDF resources and literals in the `subject`, `predicate` and `object` columns from respective tables (the binary values of the column `objects_is` indicates whether the object column references a resource or literal). The `modeID` enables the storage of multible ontologies (models) within one relational schema. For simplicity we omit constraints on this column in the example queries below. The two helper tables `lists` and `restrictions` replicate information with respect to RDF lists and OWL restrictions from the `statements` table to speed-up query execution. As a consequence, the `restrictions` table contains one row for each restriction with either one of the types `minCardinality`, `maxCardinality`, `cardinality`, `hasValue`, `someValuesFrom`, `allValuesFrom`.

4.2 Translating Inference Rules into Database Queries

After we derived inference rules for the description logic $\mathcal{SHOIN}$ and we are able to store $\mathcal{SHOIN}$ ontologies in a relational database, we would like to employ these rules for computing appropriate inferences. Each one of the rules can be translated into one or multiple corresponding SQL queries on the ontology storage schema. The general method is to translate all premises of a rule, as well as requirements on the structure of the subsumed and subsuming class in the consequent into joins and join conditions on tables of the

RDF storage schema. Query optimization techniques developed for database systems will lead to the identification of those conditions which restrict the result set most and hence can be evaluated fastest; data indexing makes sure matching classes are retrieved fast. The generated SQL queries select exactly two classes (two roles or an instance and a class) for which a subclass-superclass (subrole-superrole or type) relationship can be inferred. The result of these queries can be added back to the database in the form of triples or stored in separate tables for faster querying or deletion in case of ontology changes.

We demonstrate the method on the example of rule 37:

$$q \leq p, r_1 \sqsubseteq r_2 \rightarrow (\geq pr_1) \sqsubseteq (\geq qr_2)$$

An equivalent denotation is:

$$c_1 = (\geq pr_1), c_2 = (\geq qr_2), q \leq p, r_1 \sqsubseteq r_2 \rightarrow c_1 \sqsubseteq c_2$$

Each of the conjuncts in the premise of this rule can be translated into either joins or join conditions on the storage schema. The resulting SQL query looks as follows:

```
1 SELECT r1.subject c1,r2.subject c2
2 FROM restrictions r1 INNER JOIN restrictions r2
3     ON(r1.type=r2.type OR r1.type="cardinality")
4   INNER JOIN ‘subPropertyOf‘ p
5     ON(r1.property=p.subject AND
6       r2.property=p.object)
7 WHERE r2.type="minCardinality"
8   AND r1.value>=r2.value
```

To obtain the subsumption relation for c_1 and c_2, we select in the query first those cardinality restrictions from the restrictions helper table (lines 2-3), where the role of the first cardinality restriction is a subrole of the role of the second cardinality restriction (lines 4-6) and the cardinality value of the first restriction is greater or equal than the cardinality value of the second restriction (line 8).

Unfortunately, due to the systematic derivation of the inference rules from sequent calculus proofs, the KIT rules are not minimal. The rules regarding only conjunctions and disjunctions, for example, can be significantly simplified as the following proposition shows:

PROPOSITION 2. *The subsumption inference rules 2-5,7 are equivalent to:*

$$\rightarrow \quad c_1 \sqcap c_2 \sqsubseteq c_1 \tag{67}$$
$$\rightarrow \quad c_1 \sqsubseteq c_1 \sqcup c_2 \tag{68}$$
$$c_1 \sqsubseteq c_3, c_2 \sqsubseteq c_4 \quad \rightarrow \quad c_1 \sqcap c_2 \sqsubseteq c_3 \sqcap c_4 \tag{69}$$

PROOF. We proof the equivalence by deriving rules 67-69 from rules 2-5,7 and conversely:

$\Rightarrow$ Rule 67 can be derived from rule 2 by setting $c_3 = \bot$. Rule 68 can be derived from rule 2 by setting $c_2 = \top$. Rule 69 can be derived from applying rule 3 twice (to $c_1 \sqsubseteq c_3$ and $c_2 \sqsubseteq c_4$) and on the conclusions of both rule 5.

$\Leftarrow$ Rule 2 is a combination of rules 67 and 68. Rule 3 can be derived from rule 69 by setting $c_4 = \top$. Rule 4 can be derived from rule 68, by adding a subsumption condition to the premise. The forward direction of rule 5 can be derived from rule 69 by setting $c_1 = c_2$, the backward direction from rule 67. Rule 7 can be derived from rule 69 by setting $c_2 = c_4$ and adding a subsumption condition to the premise. $\square$

In addition to discovering eliminating redundancies in the rule set (which we consider to still be an ongoing effort), we faced the following, rather technical challenges during the translation:

- The inference rules contain only binary intersections and unions. For OWL ontologies, however, the union or intersection operator is applied to an arbitrarily long list of objects (as stored in the `lists` table). Hence, the resulting SQL queries must handle unions and intersections with arbitrary numbers of constituents.

- Some rules are bi-directional (i.e. support inferences in both directions), some state the equivalence of two objects (i.e. $c_1 \equiv c_2 \Leftrightarrow c_1 \sqsubseteq c_2, c_2 \sqsubseteq c_1$). We had to split each of those into two separate rules.

- The OWL embedding in the RDF subject-predicate-object data model, which served as a basis for the storage schema, defines a number of shortcuts for expressing certain axioms, i.e. `<c1;owl:disjointWith;c2>` for $c_1 \sqcap c_2 \sqsubseteq \bot$, `<r;rdfs:domain;c>` for $(\geq 1r) \sqsubseteq c$ or `<r;rdfs:range;c>` for $\top \sqsubseteq \forall r.c$. Hence, we had to make sure the OWL shortcuts are used when checking for (or adding of) corresponding $\mathcal{SHOIN}$ axioms is required.

For a complete list of the generated SQL queries we refer to the source code of our implementation (which is presented in Section 5). Theoretically, the rules could also be translated into an RDF query language (such as SPARQL [27]), which would turn each RDF store into a reasoner. Practically however, we are not aware of an RDF query language, which is sufficiently expressive (especially with regard to required counting and aggregation functionality).

4.3 A Fixpoint Algorithm for Stratified Inferences

After we obtained a set of SQL queries, we wanted to find an efficient way for their execution. Due to implications between the inference rules and consequently between queries, we have to execute the queries until no further inferences are possible. However, doing that in an arbitrary order might turn out to be fairly inefficient. In order to overcome this obstacle, we derived a stratification of the inference rules based on observations regarding the dependencies between inference rules:

- Rules 8, 9 and 57 can be safely ignored, since they give no exploitable information.

- Only 13 out of the 50 inference rules for class subsumption depend on the subsumption relation excluding tautology (i.e. subsumption of $\top$) and unsatisfiability (i.e. subsumption by $\bot$), namely rules: 3-7, 12, 13, 16-19, 22, 23.

- The following rules depend with respect to concept subsumption solely on unsatisfiability: 11, 14, 29, 30.

- The following rules depend with respect to concept subsumption solely on tautologies: 10,15

- Role subsumption and inversion inferences (i.e. rules 51-55) are independent of class subsumption and only the role subsumption rules 52, 54, 55 depend on the role hierarchy.

- Role subsumption and inversion as well as class subsumption rules are up to nominals independent of instance recognition rules.

As a result of these observations, it is safe to first compute role subsumption and inversion inferences (lines 2-5 in Algorithm 1), proceed with concept subsumption inferences (lines 6-15) and finally compute class membership (lines 16-19).

Since the algorithm can be implemented in just a few lines of code with a modest number of additional SQL queries it can be easily integrated as a small middleware layer on top of a standard relational database. Once such a layer is available, the revealed implicit information from OWL-DL ontologies will be fully accessible for standard database applications. The integration of ontologies and relational databases simply boils down to joining the `statements` table in the RDF storage schema (now also containing inferred relationships) with other database tables containing instance data.

To illustrate the impact for the integration of ontologies and databases, we look back at our introductory example from Section 2. Suppose in addition to the Galen ontology, there is a database table `organisms` containing experimental data about a number of organisms examined by a biomedical research group. A list of all microorganisms can be simply retrieved by the following query, once the subsumption inference results are added back to the RDF storage schema:

```
SELECT o.* FROM statements s
  INNER JOIN organisms o ON(o.type=s.subject)
WHERE s.predicate="rdfs:subClassOf"
  AND s.object="MicroOrganism"
```

Since there is no overhead in serializing an ontology, transferring it to a separate reasoner, serializing the inference results, transferring them back and adding them to the database the performance and scalability depends solely on the inference algorithm (and database query performance), whose evaluation in real application scenarios is the aim of the following section.

5. EVALUATION

Our approach aims at supporting the most advanced applications of ontologies. These can be found in biology, medicine, commerce and – central for all domains – so called upper-level ontologies, which define crucial concepts for reference in multiple domains. The experimental methodology aims at comparing the performance of our implementation OWLDB with both a state-of-the-art OWL reasoner and a specialized RDF instance store. Consequently, the evaluation criteria are:

- *soundness and completeness* of the subsumption and instance reasoning with respect to the given data set,
- *scalability* of subsumption and instance reasoning reasoning,
- *query performance* for queries of various complexity.

The employed ontologies and instance data sets comprise central ontologies from the mentioned domains as well as synthetic benchmark ontologies and instance data sets.

Input: $\mathcal{SHOIN}$ ontology stored in a relational database
Output: inferred concept and role subsumption hierarchies and classification of instances

1 **begin**
2 **Execute** inference rules for role subsumption which do not depend on the role subsumption relation (51,53);
3 **repeat**
4 **Execute** inference rules for role subsumption which depend on the role subsumption relation (52,54,55);
5 **until** *no new role subsumption inferences* ;
6 **Execute** inference rules for concept subsumption which do not depend on the concept subsumption relation;
7 **repeat**
8 **Execute** inference rules for concept subsumption which dependent on the concept subsumption relation excluding rules depending solely on tautology and unsatisfiability;
9 **if** *first loop* **or** *new unsatisfiability inferred* **then**
10 **Execute** inference rules for concept subsumption which dependent with respect to concept subsumption solely on unsatisfiability (11, 14, 29, 30);
11 **end**
12 **if** *first loop* **or** *new tautology inferred* **then**
13 Execute inference rules for concept subsumption which dependent with respect to concept subsumption solely on tautologies (10, 15);
14 **end**
15 **until** *no new concept subsumption inferences* ;
16 **foreach** *instance recognition rules (58-66)* **do**
17 **Execute** instance recognition rule;
18 **end**
19 **Execute** rule 56
20 **end**

Algorithm 1: Fixpoint algorithm for stratified $\mathcal{SHOIN}$ reasoning.

5.1 Implementation

The presented approach has been implemented on the basis of the Semantic Web application development framework Powl [1]. Hence, OWLDBs architecture consists of the following components:

- *Parser/Serializer* for loading and exporting ontologies from various formats. This component is provided by Powl.
- *RDBMS.* The execution of the generated SQL queries has been tested with the database systems MySQL and DB2.
- *The subsumption reasoner and instance classifier* is the core OWLDB component. It can be configured to utilize different storage engines (e.g. on-disk or in-memory tables) or schemata (e.g. inferred triples in

statements table, separate property tables) to store inferred results.

- *API and SPARQL interface.* All components are accessible by an Application Programming Interface implemented in PHP. To query OWLDB Powl provides RDQL and SPARQL query interfaces.
- *Frontends.* OWLDB is further prototypically integrated into Powl's Web frontend for ontology authoring and into the social semantic collaboration tool OntoWiki [2].

Powl as well as OWLDB are Open-Source and available under GNU Public License (GPL). A web-interface where OWLDB can be tested with custom ontologies is available at http://powl.sf.net/owldb.

5.2 Experimental Methodology

Since the OWLDB approach comprises TBox (i.e. subsumption) as well as ABox (i.e. instance recognition) reasoning, we evaluate OWLDBs performance by comparing it to a prominent representative from both worlds — the Pellet DL reasoner (in Version 1.3) for subsumption reasoning and the RDF instance store Sesame/OWLIM performing ABox reasoning. OWLDB was running with Powl 0.94 on PHP 5.1.6. A MySQL database (version 5.1.12) was used as underlying DBMS with the MyISAM on disk storage engine and configuration defaults. All systems were run on a notebook computer with a 2 GHz, Pentium M processor and 2 GB main memory. The Java Virtual Machine running Pellet and Sesame/OWLIM was configured with a 1.5 GB heap size. The benchmark results represent averages of 5 runs for each test. The following ontologies and instance data sets were used within the evaluation:

Ontology	Expressiveness	Class.	Prop.	Inst.	Size (MB)
Wine		137	17	206	0.12
Wine5	SHOIF(D)	685	85	1030	0.61
Wine10		1370	170	2060	1.23
Pathway	EL	486	3	0	0.20
MolFunc	ALR+	7957	3	0	2.89
BioProc	ALR+	11K	3	0	5.42
MGED	ALCOF(D)	234	162	709	0.41
Galen	SHF	2749	413	0	2.33
SUMO	ALH(D)	630	246	435	0.46
eClassOWL	ALH(D)	76976	5529	4544	25.4
UOBDL1				0.05M	23
UOBDL5	SHOIN(D)	69	47	0.2M	100
UOBDL10				1.2M	196

Table 2: Expressiveness and sizes of benchmark ontologies.

Wine Ontology is the example ontology accompanying the official W3C OWL standard. To exhibit the OWL features, it *exemplary demonstrates* their usage in the domain of wine and food. Due to the comprehensive and balanced usage of different OWL expressions, the Wine Ontology qualifies as an indicator for the grade of OWL support. Further, when used as prototype for ontologies of larger size (by cloning resources), these can be expected to give meaningful insights into the scalability properties of a reasoning algorithm.

Open Bio-Medical Ontologies (OBO) is a collection of well-structured controlled vocabularies for shared use across different biological and medical domains. The original OBO format is a representation based on directed acyclic graphs, which can be straightforwardly translated into OWL. Currently the OBO project comprises more than 60 individual ontologies with individual sizes ranging from thousands to millions of triples. From those, we selected the following ontologies for the evaluation: Biological Process (BioProc), Pathway, Molecular Function (MolFunc), Microarray experimental conditions (MGED).

EClassOWL [15] is an OWL-Lite representation of eClass – a products and services categorization standard. EClass includes products and services concepts, product properties, values for enumerated data types, a hierarchy of the product concepts reflecting the perspective of a buying organization, recommendations which properties should be used for which type of products, and recommendations which values are allowed for which property. The eClassOWL ontology was built to capture as much of the original semantics as possible while being well within the limits of OWL-DL.

In addition to these, we evaluated OWLDB with the OWL version of **Galen** [31] and the OWL representation of the **SUMO** - Suggested Upper Merged Ontology [34]. The ABox reasoning and querying was tested using the dataset and queries of the **UOB** - University Ontology Benchmark [21], an extension of the well known Lehigh University Benchmark (LUBM) [12].

5.3 Comparison with DL-Reasoners

To evaluate OWLDB's subsumption reasoning, we compared OWLDB and Pellet [32] with regard to the execution time for the calculation of the overall subsumption relation in various ontologies. For all tested ontologies for which Pellet was able to compute the subsumption relation, OWLDB's subsumption results were the same as Pellet's and hence *sound and complete*.

To evaluate the scalability of OWLDB's subsumption algorithm, we processed the wine ontology in different sizes. These were derived by simply copying all axioms and descriptions and renaming all classes, properties and instances therein. The results are visualized in Figure 5.

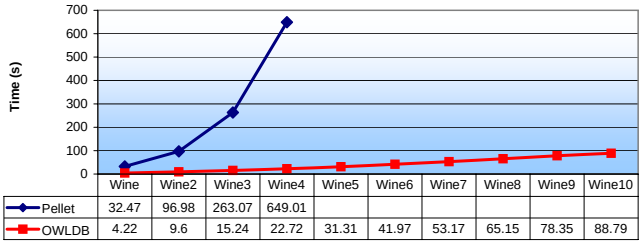

Figure 5: Scalability of subsumption for Pellet and OWLDB for the Wine ontology in different sizes.

The measured results show that OWLDB's execution time grows significantly slower than Pellet's for increasing ontology sizes. Furthermore, Pellet's desire for main memory exceeds the available resources starting from Wine5. This need for main memory limits the amount of potentially processable OWL ontologies for Pellet (and, we conjecture, most tableau algorithm based reasoners) dramatically. However, the MySQL process executing OWLDB's queries

uses a small (less than 50MB) and constant amount of main memory independent of the ontology size. Consequently, OWLDB is able to process potentially all OWL ontologies as long as execution time is not limited.

In Figure 6, the subsumption calculation time of OWLDB and Pellet is presented for various ontologies of different sizes, using different OWL features.

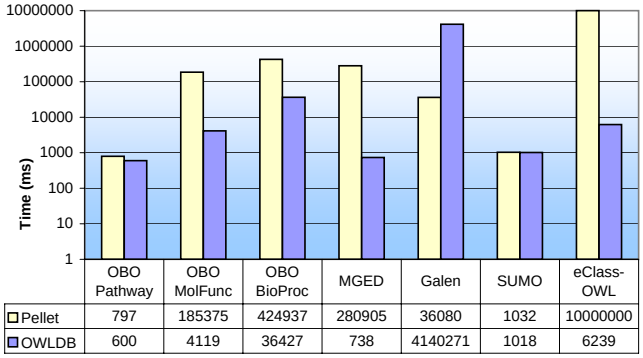

Figure 6: Calculation of the subsumption relation for various ontologies (Note the logarithmic scale).

OWLDB computes the subsumption relation significantly faster for 4 out of the 7 ontologies. Pellet's and OWLDB's performance are comparable for the two small ontologies OBO Pathway and SUMO. For the Galen ontology OWLDB is significantly slower. Profiling OWLDB for Galen showed that 90% of the computation was spend on computing the transitive closure of the subsumption relation (rule 1) and on the detection of intersection classes where each member is a super-class of a member of a second intersection class (rule 69). The former is very probable due to the - with 10 - high average depth of the class tree, the latter due to the large number of intersection classes (2024).

5.4 Comparison with Triple Stores

To evaluate OWLDB's ABox reasoning and querying performance, we used the University Ontology Benchmark for OWL DL (UOB [21]) of the UOB ontology with instance data for 1, 5, and 10 universities (later referred to as UOB-DLx). Sesame [10] (version 1.2.6) together with its OWLIM storage and inference layer [18] (version 2.8.4) served as a reference system for performance comparison.

Storing ontologies in relational databases gives developers more flexibility to fine-tune table layouts and index structures in order to facilitate reasoning, inferencing, and querying in specific scenarios. For this evaluation, OWLDB was configured to store inferred facts together with explicit information from the UOB ontologies in the statements table additionally in separate property tables (cf. [35] for a discussion about property tables). This facilitates high-performance querying, but it may, however, result in slightly longer load and update times.

5.4.1 *Loading*

We first compared the load times (see Figure 7). OWLDB is roughly 3 times slower than OWLIM when parsing the RDF files, load them into the data structures and perform ABox inferences. We identified the following reasons:

- Parsing of RDF and loading into the data structures is better integrated in OWLIM. Furthermore, OWLIM works solely in main memory which at this point significantly contributes to better performance.
- OWLDB evaluates more OWL expressions (such as arbitrary cardinality restrictions) than OWLIM and hence needs more time to perform the required inferences.
- The currently used DBMS MySQL does not provide optimizations for computing transitive closures. In UOB the calculation of the property extent for the transitive property `uob:hasSameHomeTownWith` alone accounts for 30-60% of OWLDB's overall ABox inference time (cf. also Section 5.5). Hence, OWLDB will profit significantly when the SQL99 features for recursive queries are available and data access is optimized accordingly.

In general, load times have much less importance for the usage of OWLDB than for OWLIM, since (due to the persistent on-disk storage in database tables) loading of ontologies occurs just once. In OWLIM, on the contrary, ontologies are loaded into main memory each time the system is started.

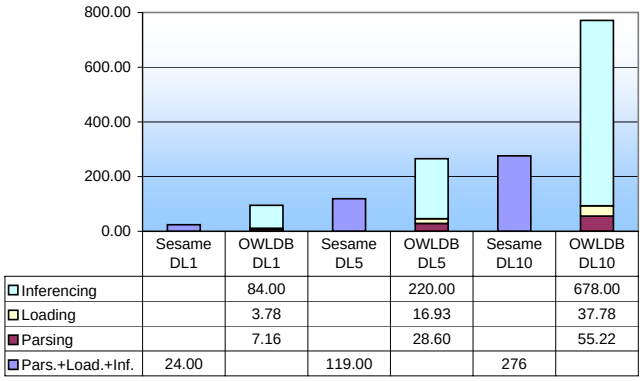

Figure 7: Loading of UOB ontologies into OWLIM and OWLDB. For OWLDB numbers are broken down into inferencing, loading and parsing.

5.4.2 *Querying*

OWLDB is capable to answer all UOB queries sound and complete. OWLIM has problems answering the following UOB queries :

- No results are returned for:

Version	Query	Reference
UOB-DL5	5	200
UOB-DL5	9	1041
UOB-DL5	14	6913

- An incomplete result set is returned for:

Version	Query	OWLIMs	Reference
UOB-DL1	14	6643	6893
UOB-DL5	11	6210	6230
UOB-DL5	14	6696	6913
UOB-DL10	14	6712	7088

- The answers for query 15 are not sound:
  OWLIM returns 379 (416, 408) results instead of 62 (76, 79) for DL1 (DL5, DL10).

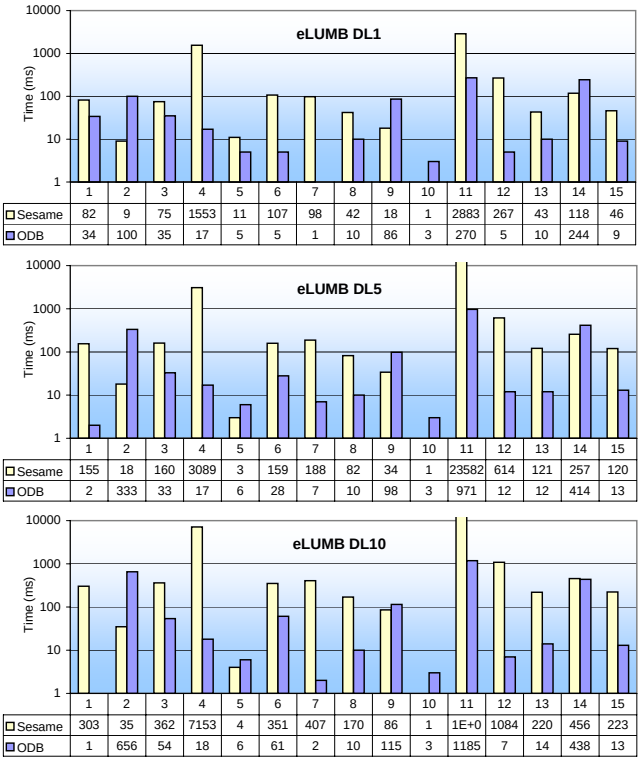

Figure 8: OWLDB and OWLIM querying performance results for the University Ontology Benchmark with instance data for 1, 5 and 10 universities.

Table 8 summarizes OWLDB's and OWLIM's query performance. Please note the logarithmic scale. OWLDB is able to answer 10 out of 15 UOB queries significantly (2 to over 100 times) faster than OWLIM. OWLIM outperforms OWLDB in 4 queries (i.e. 2, 5, 9, 10) by approximately 2 to 15 times. In addition, OWLDB is capable to support OWL features not included in the University Ontology Benchmark (e.g. more complex subsumption reasoning).

5.5 Profiling OWLDB reasoning

In order to gain insights which OWL expressions are the most expensive ones with respect to reasoning in OWLDB, we profiled the timings of SQL query execution for each of the evaluation ontologies. Surprisingly, more than 50% of the overall query execution time was caused by only two query types:

- reasoning and inferences involving transitive properties with large property extents (including the RDF-S and OWL vocabulary properties, `owl:sameAs`, `rdfs:subClassOf`)
- subsumption inferences on the basis of the inference rule 69 (i.e. the detection of intersection classes where each member is a super-class of a member of a second intersection class).

Hence, further significant performance improvements can be expected from optimizing these queries either at the database or application level.

6. RELATED WORK

Traditionally, approaches for reasoning and inferencing for Description Logics are either optimized for TBox reasoning in form of tableau based description logic reasoners or implemented as proprietary or database backed instance stores with ABox reasoning capabilities. In addition there exist efforts to integrate both strategies as-is into larger systems. The problem of scalability is furthermore tackled by approaches to modularize ontologies. Other approaches and developments in addition to the ones presented in this section can be found in the slightly outdated SWAD deliverable [7].

6.1 Description Logic Reasoners

One of the few native OWL reasoners is Pellet [32] — an open-source OWL-DL reasoner written in Java. Pellet is based on tableau algorithms developed for expressive description logics. It supports full OWL-DL including reasoning about nominals. Pellet is becoming increasingly popular due to its easy deployment and the ability to directly load any OWL ontology. RACER [13] is a very well optimized and maybe the fastest tableau-based reasoner. However, both Pellet and RACER perform in-memory reasoning and are due to the memory requirements of tableau reasoning limited with regard to scalability and do not offer to selectively disable computationally expensive reasoning steps in order to trade completeness for speed as OWLDB does. A system which apparently has many similarities to the presented approach is KAON2 [23]. It realizes on-disk reasoning for a subset of OWL with disjunctive datalog, which is close to other database query languages such as SQL. Due to KAON2's missing support for nominals we were not able to compare both systems directly.

6.2 Instance Stores

Instance stores such as Jena [36], Sesame [10], OWLIM [18], Mulgara[1] and AllegroGraph[2] mainly focus on the efficient processing of ABox queries and support rule-based TBox inferences often only in a very limited way. The Jena toolkit for example implements a Java API including support for different database backends and light-weight forward- and backward-chaining RDF-Schema reasoning together with support for the declarative RDF query languages RDQL and SPARQL. A system primarily focused on efficient parsing, storing and querying is Sesame. In addition to accessing relational databases for storing RDF data, Sesame provides a highly optimized in-memory RDF storage layer, known to be very fast in computing and materializing entailed statements. Its modular structure allows the implementation of customized Storage and Inference Layers (SAIL). The reference system of our evaluation OWLIM for example is such a Sesame SAIL. OWLIM though aims at balancing between a scalable repository and light-weight reasoner and in-memory, forward chaining rule inferences based on Description Logic Programs.

6.3 Combined Approaches

One of the earliest approaches to coupling a DL reasoner with a database is described in [8]. It uses SQL to classify instances stored in a database and loading them into

9[1]http://www.mulgara.org
9[2]http://www.franz.com/products/allegrograph

the CLASSIC system. In the context of the Semantic Web, the DLDB [26] system was an early approach to use an RDBMS for RDF and limited RDF-S entailment and connecting it with a reasoner for DAML+OIL subsumption. More recently, InstanceStore [6] and Minerva [38] combined a tableau based DL reasoner for the TBox inference with rules for the ABox inference. In Minerva rules are translated into SQL queries on a relational triple store schema. However, the selected rule language in Minerva is relatively weak and focuses mainly on instance retrieval.

6.4 Modularization of Ontologies

Another way to cope with the growing size of ontologies in addition to more efficient indexing and better optimized algorithms is to segment ontologies into reasonably self-contained parts for separate processing. Work with regard to the modularization of ontologies can be categorized along three main categories:

- *Query-based methods* aiming at generating views of ontologies by means of database inspired query languages such as SPARQL [27],
- *Network partitioning*, which views ontologies as networks, which can be clustered into different segments with regard to characteristics of its nodes,
- *Extraction by traversal* sees the ontology as an network of nodes, which is partitioned by starting at a certain node and following links to other nodes, a prominent representative is the PROMPT algorithm [24].

The dependency analysis as part of the OWLDB approach is closely related to extraction by traversal. An overview about further approaches from each of these categories can be found in [31]. A method to drastically reduce the reasoning relevant ABox part of an ontology by eliminating redundant data is described in [11] and yields good results especially with the homogeneous bio-medical ontologies.

7. CONCLUSIONS AND FUTURE WORK

Using today's tools, reasoning about large ontologies (either in terms of many classes or many instances) is not feasible: full-blown description logic reasoners do not scale adequately. Strategies to push as many inference tasks as possible into standard relational database systems are a way to overcome these difficulties, as DBMS query processing is designed to handle large data volumes.

In this paper, we have developed such an approach, presenting a set of techniques for mapping subsumption inferences into SQL queries, which are then recomputed until a fixpoint is reached. We evaluated our resulting OWLDB system across a variety of ABox and TBox reasoning tasks from real-world and synthetic ontologies. Our implementation typically ran faster than both the Pellet OWL reasoner and the highly specialized Sesame/OWLIM instance store; meanwhile, OWLDB provides greater scalability than Pellet and better expressiveness than OWLIM.

We see this work as the initial stage in a longer-term agenda. Our initial OWLDB engine represents the first step towards scalable reasoning for OWL ontologies; we would like to improve its support for incremental ontology updates and for distributed computation across multiple database systems. Taking our lessons to a related but higher-level problem, we would also like to explore how data integration techniques [20, 14], which provide view-based data translation capabilities, might be extended to incorporate OWL reasoning capabilities in a similar way to OWLDB. Such an architecture would provide an excellent basis for semantic data sharing across ontologies when the classes do not precisely align: for instance, database schema mappings can convert between prices in different currencies, or from one type of local ID to another.

8. ACKNOWLEDGMENTS

We would like to thank Joe Kopena, Frank Loebe, Jens Lehmann and Robert Hoehndorf for fruitful discussions and valuable suggestions.

9. REFERENCES

[1] S. Auer. Powl: A web based platform for collaborative semantic web development. In S. Auer, C. Bizer, and L. Miller, editors, *Proceedings of the Workshop Scripting for the Semantic Web*, number 135 in CEUR Workshop Proceedings, Heraklion, Greece, 05 2005.

[2] S. Auer, S. Dietzold, and T. Riechert. Ontowiki - a tool for social, semantic collaboration. In I. C. et al., editor, *Proc. of 5th International Semantic Web Conference, Athens, GA, USA, Nov 5th-9th*, number 4273 in LNCS, pages 736–749. Springer-Verlag Berlin Heidelberg, 2006.

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

[4] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.

[5] S. Bechhofer. Web Ontology Language (OWL) reference version 1.0. W3C. Technical report, W3C, 2004.

[6] S. Bechhofer, I. Horrocks, and D. Turi. The OWL instance store: System description. In R. Nieuwenhuis, editor, *CADE*, volume 3632 of *Lecture Notes in Computer Science*, pages 177–181. Springer, 2005.

[7] D. Beckett and J. Grant. Swad-europe deliverable 10.2: Mapping semantic web data with rdbmses. Technical report, W3C Semantic Web Advanced Development for Europe (SWAD-Europe), 2003.

[8] A. Borgida and R. J. Brachman. Loading data into description reasoners. In P. Buneman and S. Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 217–226, Washington, D.C., 26–28 May 1993.

[9] D. Brickley and L. Miller. FOAF vocabulary specification. http://xmlns.com/foaf/0.1/, 2005.

[10] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: An architecture for storing and querying RDF data and schema information, 2003.

[11] A. Fokoue, A. Kershenbaum, L. Ma, E. Schonberg, and K. Srinivas. The summary abox: Cutting ontologies down to size. In I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, *International*

Semantic Web Conference, volume 4273 of *Lecture Notes in Computer Science*, pages 343–356. Springer, 2006.

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

[13] V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In B. Nebel, editor, *Proceedings of the seventeenth International Conference on Artificial Intelligence (IJCAI-01)*, pages 161–168, San Francisco, CA, Aug. 4–10 2001. Morgan Kaufmann Publishers, Inc.

[14] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *ICDE*, March 2003.

[15] M. Hepp. Products and services ontologies: A methodology for deriving owl ontologies from industrial categorization standards. *International Journal on Semantic Web & Information Systems (IJSWIS)*, 2:77–99, 2006.

[16] R. Hoehndorf, K. Prüfer, M. Backhaus, H. Herre, J. Kelso, F. Loebe, and J. Visagie. A proposal for a gene functions wiki. In R. Meersman, Z. Tari, and P. Herrero, editors, *OTM Workshops (1)*, volume 4277 of *Lecture Notes in Computer Science*, pages 669–678. Springer, 2006.

[17] U. Hustadt, B. Motik, and U. Sattler. Reducing shiq description logic to disjunctive datalog programs. In D. Dubois, C. A. Welty, and M.-A. Williams, editors, *KR2004: Principles of Knowledge Representation and Reasoning*, pages 152–162, Menlo Park, California, 2004. AAAI Press.

[18] A. Kiryakov, D. Ognyanov, and D. Manov. OWLIM - A pragmatic semantic repository for OWL. In M. Dean, Y. Guo, W. Jun, R. Kaschek, S. Krishnaswamy, Z. Pan, and Q. Z. Sheng, editors, *WISE Workshops*, volume 3807 of *Lecture Notes in Computer Science*, pages 182–192. Springer, 2005.

[19] G. Klyne, J. J. Caroll, and B. McBride. Resource description framework (rdf): Concepts and abstract syntax. Technical report, W3C, 2002.

[20] M. Lenzerini. Tutorial - data integration: A theoretical perspective. In *PODS*, 2002.

[21] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. Towards a complete OWL ontology benchmark. In Y. Sure and J. Domingue, editors, *ESWC*, volume 4011 of *Lecture Notes in Computer Science*, pages 125–139. Springer, 2006.

[22] J. L. V. Mejino, N. F. Noy, M. A. Musen, J. F. Brinkley, and C. Rosse. Representation of structural relationships in the foundational model of anatomy. In *Proceedings, American Medical Informatics Association Fall Symposium*, 2001.

[23] B. Motik and U. Sattler. Practical DL reasoning over large a-boxes with KAON2.

[24] Noy, N. F., Musen, and M. A. The PROMPT suite: interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6):983–1024, 2003.

[25] Open biomedical ontologies. Available from `http://obo.sourceforge.net/`, 2004.

[26] Z. Pan and J. Heflin. Dldb: Extending relational databases to support semantic web queries. Technical report, Dept. of Computer Science and Engineering, Lehigh University, 2004.

[27] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Working Draft (http://www.w3.org/TR/rdf-sparql-query/), 2005.

[28] V. Royer and J. Quantz. Deriving inference rules for terminological logics. In D. Pearce and G. Wagner, editors, *JELIA*, volume 633 of *Lecture Notes in Computer Science*, pages 84–105. Springer, 1992.

[29] V. Royer and J. J. Quantz. Deriving inference rules for description logics: a rewriting approach into sequent calculi. Technical Report TUB-FB13-KIT-111, KIT Project Group Publications, Dec. 1 1993. Tue, 07 Nov 1995 19:31:17 GMT.

[30] S. Schulz, E. Beisswanger, U. Hahn, J. Wermter, H. Stenzhorn, and A. Kumar. From GENIA to BioTop – Towards a top-level ontology for biology. In *International Conference on Formal Ontology in Information Systems (FOIS 2006), Baltimore, Maryland (USA), November 9-11, 2006*, 2006.

[31] J. Seidenberg and A. L. Rector. Web ontology segmentation: analysis, classification and use. In L. Carr, D. D. Roure, A. Iyengar, C. A. Goble, and M. Dahlin, editors, *WWW*, pages 13–22. ACM, 2006.

[32] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Submitted for Publication to Journal of Web Semantics*.

[33] C. Stoeckert, H. Parkinson, T. Whetzel, P. Spellman, C. A. Ball, J. White, J. Matese, L. Fan, G. Fragoso, M. Heiskanen, S. Sansone, H. Causton, L. Game, and C. Taylor. An ontology for microarray experiments in support of MAGE v.1. Available from `http://mged.sourceforge.net/ontologies/MGEDontology.php`, March 2005.

[34] SUMO. The suggested upper merged ontology. Teknowledge, 2003. Version 1.60.

[35] K. Wilkinson. Jena property table implementation. In *Proceedings of Second International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2006) 5th International Semantic Web Conference, Athens, Georgia, USA 2006*.

[36] K. Wilkinson, C. Sayers, H. A. Kuno, and D. Reynolds. Efficient RDF storage and retrieval in Jena2. In I. F. Cruz, V. Kashyap, S. Decker, and R. Eckstein, editors, *SWDB*, pages 131–150, 2003.

[37] S. Zhang, O. Bodenreider, and C. Golbreich. Experience in reasoning with the foundational model of anatomy in OWL DL. In R. B. Altman, T. Murray, T. E. Klein, A. K. Dunker, and L. Hunter, editors, *Pacific Symposium on Biocomputing*, pages 200–211. World Scientific, 2006.

[38] J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, and Y. Pan. Minerva: A scalable OWL ontology storage and inference system. In R. Mizoguchi, Z. Shi, and F. Giunchiglia, editors, *Asian Semantic Web Conference*, volume 4185 of *Lecture Notes in Computer Science*, pages 429–443. Springer, 2006.