Chapter

ONTOLOGY REASONING WITH LARGE DATA REPOSITORIES

Stijn Heymans[1], Li Ma[2], Darko Anicic[1], Zhilei Ma[3], Nathalie Steinmetz[1], Yue Pan[2], Jing Mei[2], Achille Fokoue[4], Aditya Kalyanpur[4], Aaron Kershenbaum[4], Edith Schonberg[4], Kavitha Srinivas[4], Cristina Feier[1], Graham Hench[1], Branimir Wetzstein[3], Uwe Keller[1]

[1]*Digital Enterprise Research Institute (DERI), University of Innsbruck*

[2]*IBM China Research Lab*

[3]*Institute of Architecture of Application Systems (IAAS), University of Stuttgart*

[4]*IBM T.J. Watson Research Center*

Abstract: As reasoning with large amounts of data together with ontological knowledge is becoming an increasingly more pertinent issue, we will give in this chapter an overall introduction to some well-known ontology repositories, including native stores and database based stores, and highlight strengths and limitations of each store. We take Minerva as an example to analyze ontology storage in databases in depth, as well as to discuss efficient indexes for scaling up ontology repositories. We then discuss a scalable reasoning method for handling expressive ontologies, as well as summarize other similar approaches. We will subsequently delve into the details of one particular ontology language based on Description Logics called WSML-DL and we will show that reasoning with this language can be done by a transformation from WSML-DL to OWL DL and supports all main DL specific reasoning tasks. Finally, we want to make reasoning a bit more tangible by showing a reasoning example in a practical business context: we thus present the Semantic Business Process Repository (SBPR) for systemical management of semantic business process models. We first analyze the main requirements on a SBPR. After the comparison of different approaches for storage mechanisms, we conclude that a RDBMS with the IRIS inference engine integrated is, due to the expressiveness of the query language and the reasoning capability, a suitable solution.

Key words: Reasoning with large datasets, OWL DL, WSML DL, IRIS, business repository

1. INTRODUCTION

Reasoning with large amounts of data together with ontological knowledge is becoming an increasingly more pertinent issue. Especially in the case of Semantic Web Applications an important question is how to store these ontologies and how to reason with them, without losing out of sight the need for scalability: in the end the Semantic Web is envisaged to contain a huge amount of data, and reasoning with ontologies for maintaining semantical information requires scalable reasoners to extract the relevant information from these ontologies.

In order to give the reader an overview of existing solutions regarding the storage of ontologies we will start this chapter by giving an overview of existing ontology stores. Furthermore, we will explore the use of relational databases extensively as an efficient means to store ontologies.

After discussing how OWL - currently the most prominent ontology language on the Semantic Web – ontologies can be stored, we will investigate a particular language, WSML-DL and see how it can be translated to OWL-DL, thus giving the reader also insight in WSML-DL reasoning by relating it to the storing capabilities described for OWL in this chapter.

Whereas the first part of this chapter thus focuses on Description Logic based languages like OWL and WSML-DL, we will in the final part discuss a Logic Programming approach, based on WSML-Flight, and see how reasoning with ontologies can be done in that context.

An important use case for the Logic Programming approach is given by the area of Business Process Management. The globalization of the economy and the ongoing change of the market situation challenge corporations to adapt their business processes in an agile manner to satisfy the emerging requirements on the market and stay competitive against their competitors. Business Process Management (BPM) is the approach to manage the execution of IT-supported business processes from a business expert's point of view rather than from a technical perspective (Smith et al. 2003). However, currently businesses have still very incomplete knowledge of and very incomplete and delayed control over their process spaces. Semantic Business Process Management (SBPM) extends the BPM approach by adopting semantic web and semantic web service technologies to bridge the gap between business and IT worlds (Hepp et al., 2005).

In both BPM and SBPM business processes play a central role. As business processes manifest the business knowledge and logics of a corporation and normally more than one person or organization with different expertise and in different geographic locations are involved in management of business processes, it is necessary to establish a Business Process Repository (BPR) within the corporation for effective sharing of valuable business knowledge. Furthermore, business users tend to reuse existing business process artifacts during process modeling, so that they are able to adapt the business processes in a more agile manner. However, as the number of business processes increases, it is difficult for them to manage the process models by themselves and to find the required business process information effectively. A BPR helps business users by providing a systematic way to manage and obtain information on business processes.

In SBPM business process models are based on process ontologies and make use of other ontologies, such as organizational ontology or semantic web service ontology (Hepp et al. 2007). The BPR has to be able to cope with these ontological descriptions when storing and retrieving process models, and in particular support efficient querying and reasoning capabilities based on the ontology formalism used. In order to distinguish from traditional BPR technology, we call this kind of repository a Semantic Business Process Repository (SBPR).

We first analyze the functional requirements on the SBPR. We describe what kind of functionality the SBPR should offer to its clients, which is primarily a process modeling tool. We then compare different approaches for data storage and querying based on the ontological descriptions. The comparison is based on the expressiveness of the query language, the scalability of the query processing and the effort for the integration of the query processing with the underlying data storage. We then finally describe the overall architecture of the SBPR.

2. ONTOLOGY STORAGE AND REASONING IN RELATIONAL DATABASES

2.1 Overview of Ontology Repository

In the past decade, we have seen the development of ontology repositories for use in semantic web applications. In this section, we classify some well-known repositories based on their storage schemes, summarize methods to store ontologies in relational databases, and introduce reasoning methods used by these repositories briefly.

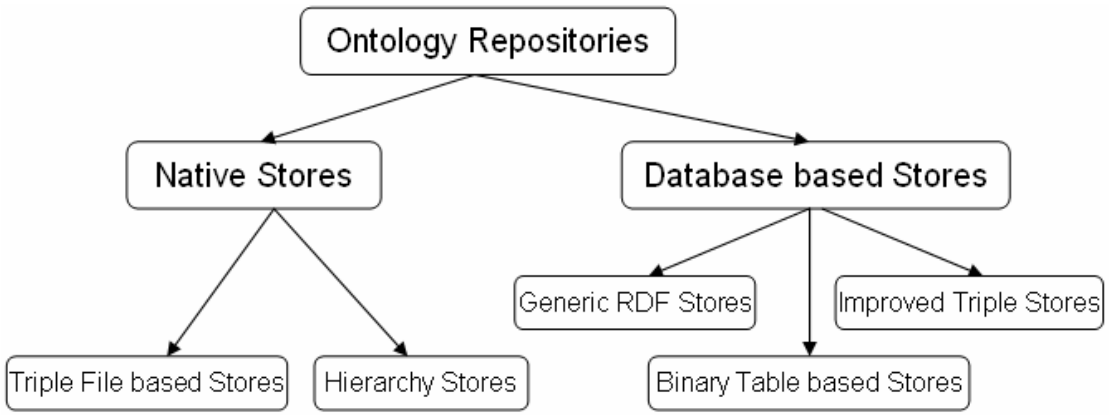

Figure -1. A Taxonomy to Classify Ontology Repositories

An ontology is in essence a directed labeled graph, which makes ontology storage highly challenging. Figure 1 shows a classification scheme for ontology repositories based on their storage models. In general, ontology repositories can be divided into two major categories, i.e., native stores and database based stores. Native stores are directly built on the file system, whereas database based repositories use relational or object relational databases as the backend store. Representative native stores include OWLIM (Kiryakov at al., 2005), HStar (Chen et al., 2006), and AllegroGraph (AllegroGraph, 2006). OWLIM and AllegroGraph adopt simple triple (N-triple) files to store all data, which results in the extremely fast speed for load and update. It is reported that AllegroGraph can load RDF data at the speed of more than 10,000 triples per second. OWLIM uses B+ trees to index triples and AllegroGraph just sorts triples in the order of (S, P, O), (P, O, S), and (O, S, P), respectively, for indexing purposes. The triple reasoning and rule entailment engine (TRREE) is utilized by OWLIM, which performs forward chaining reasoning in main memory, and inferred results are materialized for query answering. AllegroGraph can expose RDF data to Racer, a highly optimized DL reasoner (Haarslev & Moller, 2001), for inference. HStar is a hierarchy store and organizes *typeOf* triples (namely concept assertions in description logics terminology) using a class hierarchy and other non-typeOf triples (namely role assertions) using a property hierarchy. Because of its hierarchical tree models, it can leverage XML techniques to support a scalable store. Range labeling, which assigns labels to all nodes of an XML tree such that the labels encode all ancestor-descendant relationships between the nodes (Wu et al., 2004), can also largely improve query performance. Also, HStar uses B+ trees to index triples. A set of rules derived from OWL-lite is categorized into two groups, which are executed at load time using forward chaining and are evaluated at query time using backward chaining, respectively. In particular, reasoning on *SubClassOf* or *SubPropertyOf* could be easily implemented via its hierarchical trees.

Compared with database based stores, native stores greatly reduce the load and update time. However, database systems provide many query optimization features, thereby contributing positively to query response time. It is reported in (Ma et al., 2006) that a simple

exchange of the order of triples in a query may make the query time of native stores 10 times (or even more) slower. Furthermore, native stores need to re-implement the functionality of a relational database such as transaction processing, query optimization, access control, logging and recovery. One potential advantage of database based stores is that they allow users and applications to access both ontologies and other enterprise data in a more seamless way at the lower level, namely database level. For instance, The Oracle RDF store translates an RDF query into a SQL query which can be embedded into another SQL query retrieving non-RDF data. In this way, query performance can be improved by efficiently joining RDF data and other data using well-optimized database query engines. Currently, lots of research efforts are made on database based stores. So, here we focus on ontology storage and reasoning in databases.

A generic RDF store mainly uses a relational table of three columns (Subject, Property, Object) to store all triples, in addition to symbol tables for encoding URIs and literals with internal unique IDs. Both Jena and the Oracle RDF store are generic RDF stores. In Jena2 (Wilkinson et al., 2003), most of URIs and literal values are stored as strings directly in the triple table. Only the URIs and literals longer than a configurable threshold are stored in separated tables and referenced by IDs in the triple table. Such a design trades storage space for time. The property table is also proposed to store patterns of RDF statements in Jena2. An *n*-column property table stores *n-1* statements (one column per property). This is efficient in terms of storage and access, but less flexible for ontology changes. Jena2 provides by default several rule sets with different inference capability. These rule sets could be implemented in memory by forward chaining, backward chaining or a hybrid of forward and backward chaining. The Oracle RDF store (Murray et al., 2005) is the first commercial system for RDF data management on top of RDBMS. Particularly, it supports so-called rulebases and rule indexes. A rulebase is an object that contains rules which can be applied to draw inferences from RDF data. Two built-in rulebases are provided, namely RDFS and RDF (a subset of RDFS). A rule index is an object containing pre-calculated triples that can be inferred from applying a specified set of rulebases to RDF data. Materializing inferred results would definitely speed up retrieval. Different from the generic RDF store, improved triple stores, such as Minerva (Zhou et al., 2006) and Sesame on MySQL database (Broekstra et al., 2002), manage different types of triples using different tables. As we can see from the storage schema of Minerva shown in Figure 3, class and property information is separated from instances, and *typeOf* triples are isolated from other triples. The improved triple store is efficient since some self-joins on a big triple table are changed to some joins among small-sized tables. Both the generic RDF store and the improved triple store make use of a fixed database schema. That is, the schema is independent of ontologies. The schema of binary table based stores, however, changes with ontologies. These kinds of stores, such as DLDB-OWL (Pan & Heflin, 2003) and Sesame on PostgreSQL (Broekstra et al., 2002), create a table for each class (resp. each property) in an ontology. A class table stores all instances belonging to the same class and a property table stores all triples which have the same property. Such tables are called binary tables. For the subsumption of classes and properties, DLDB-OWL exploits database views to capture them, whereas Sesame leverages the sub-tables from object relational databases so as to handle them naturally. One of advantages of the binary table based store is to decrease the traversal space and improve data access for queries. That is, instances of unrelated classes or properties to a query will not be accessed. An obvious drawback is the alteration of the schema (e.g., deleting or creating tables) when ontologies change. Also, this binary table based approach is not suitable for very huge ontologies having tens of thousands of classes, such as SnoMed ontology (SnoMed, 2006). Too many tables will increase serious overhead to databases.

The above gives an overall introduction to some well-known ontology repositories, including native stores and database based stores, and highlights strengths and limitations of each store. It is reported in (Ma et al., 2006) that Minerva achieves good performance in benchmarking tests. Next, we will take Minerva as an example to analyze ontology storage in

databases in depth, as well as to discuss efficient indexes for scaling up ontology repositories. We will then discuss a scalable reasoning method for handling expressive ontologies, as well as summarize other similar approaches.

2.2 Practical Methods for Ontology Storage and Index in Relational Databases

This section discusses methods to store and index ontologies in relational databases by investigating an improved triple store, namely Minerva (Zhou et al., 2006). Figure 2 shows the component diagram of Minerva, which is comprised of Import Module, Inference Module, Storage Module (viz. an RDBMS schema) and Query Module.

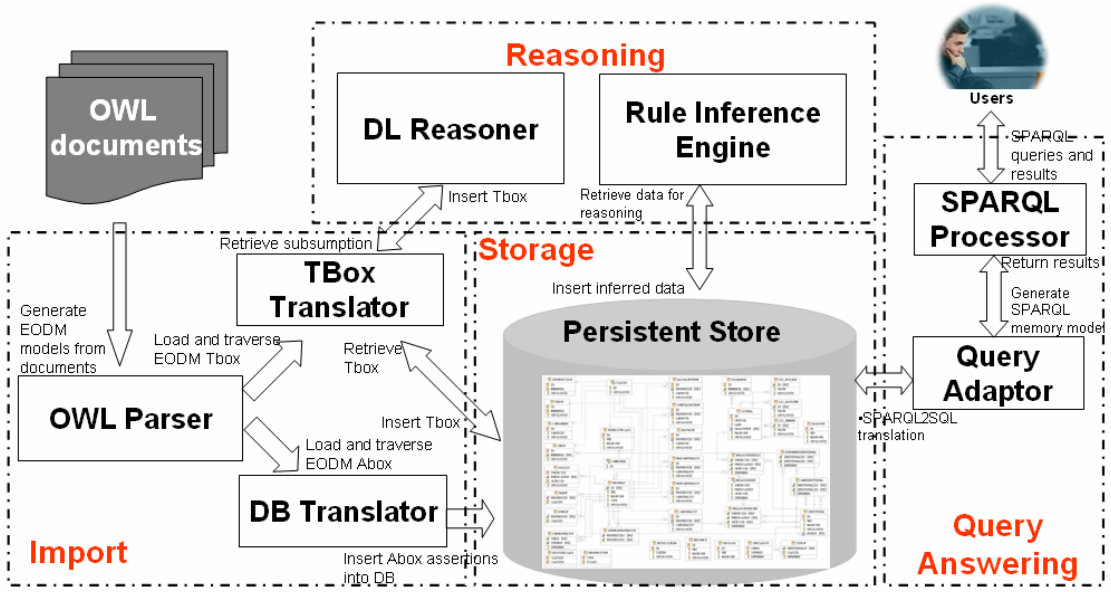

Figure -2. The Component Diagram of Minerva

The import module consists of an OWL parser and two translators. The parser parses OWL documents into an in-memory EODM model (EMF ontology definition metamodel) (IODT, 2005), and then the DB translator populates all ABox assertions into the backend database. The function of the TBox translator is twofold, one is to populate all asserted TBox axioms into a DL reasoner and the other is to obtain inferred results from the DL reasoner and insert them into the database. A DL reasoner and a rule inference engine comprise the inference module. Firstly, the DL reasoner infers complete subsumption relationships between classes and properties. Then, the rule engine conducts ABox inference based on the description logic programs (DLP) rules (Grosof et al., 2003). Currently, the inference rules are implemented using SQL statements. Minerva can use well-known Pellet (Sirin & Parsia, 2004) or a structural subsumption algorithm for TBox inference (IODT, 2005). The storage module is intended to store both original and inferred assertions by the DL reasoner and the rule inference engine. But, there is a way to distinguish original assertions from inferred assertions by a specific flag. Since inference and storage are considered as an inseparable component in a complete storage and query system for ontologies, a specific RDBMS schema is designed to effectively support ontology reasoning. Currently, Minerva can take IBM DB2, Derby, MySQL and Oracle as the back-end database. The query language supported by Minerva is SPARQL (Prud'hommeaux & Seaborne, 2006). SPARQL queries are answered by directly retrieving inferred results from the database using SQL statements. There is no

inference during the query answering stage because the inference has already been done at the loading stage. Such processing is expected to improve the query response time.

In summary, Minerva combines a DL reasoner and a rule engine for ontology inference, followed by materializing all inferred results into a database. The database schema is well designed to effectively support inference and SPARQL queries are answered by direct retrieval from the database. Jena and Sesame have provided support for ontology persistence in relational databases. They persist OWL ontologies as a set of RDF triples and do not consider specific processing for complex class descriptions generated by class constructors (boolean combinators, various kinds of restrictions, etc). The highlight of Minerva's database schema is that all predicates in the DLP rules have corresponding tables in the database. Therefore, these rules can be easily translated into sequences of relational algebra operations. For example, Rule *Type(x,C) :- Rel(x,R, y).Type(y,D).SomeValuesFrom(C,R,D)* has four terms in the head and body, resulting in three tables: *RelationshipInd*, *TypeOf* and *SomeValuesFrom*. It is straightforward to use SQL statements to execute this rule. We just need to use simple SQL *select* and *join* operations among these three tables. Leveraging well-optimized database engines for rule inference is expected to significantly improve the efficiency. Figure 3 shows the relational storage model of Minerva.

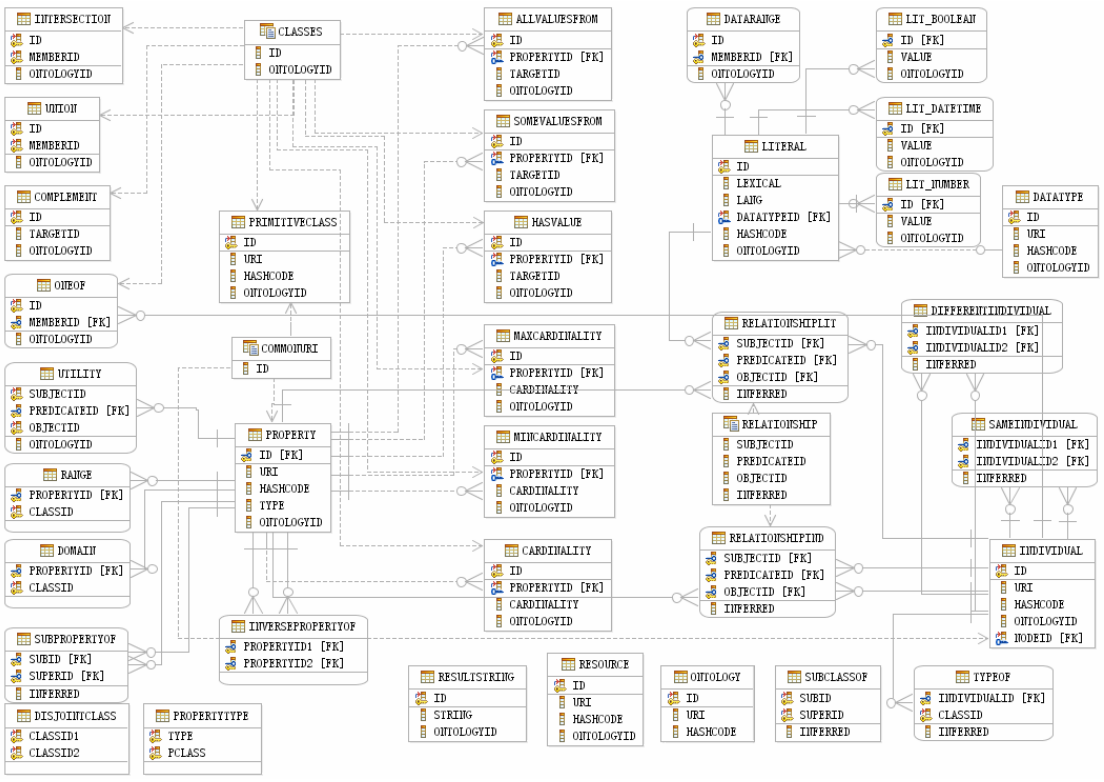

Figure -3. Database Schema of Minerva

We categorize tables of the database schema of Minerva into 4 types: atomic tables, TBox axiom tables, ABox fact tables and class constructor tables. The atomic tables include: Ontology, PrimitiveClass, Property, Datatype, Individual, Literal and Resource. These tables encode the URI with an integer (the ID column), which reduces the overhead caused by the long URI to a minimum. The hashcode column is used to speed up search on URIs and the ontologyID column denotes which ontology the URI comes from. The Property table stores characteristics (symmetric, transitive, etc.) of properties as well. To leverage built-in value comparison operations of databases, boolean, date time and numeric literals are separately represented using the corresponding data types provided by databases. There are three

important kinds of ABox assertions involved in reasoning: TypeOf triples, object property triples and datatype property triples. They are stored in three different tables, namely tables TypeOf, RelationshipInd and RelationshipLit. A view named *Relationship* is constructed as an entry point to object property triples and datatype property triples. Triples irrelevant to reasoning, such as those with RDFS:comment as the property, are stored in Table Utility. Tables SubClassOf, SubPropertyOf, Domain, Range, DisjointClass, InversePropertyOf are used to keep TBox axioms. The class constructor tables are used to store class expressions. Minerva decomposes the complex class descriptions into instantiations of OWL class constructors, assigns a new ID to each instantiation and stores it in the corresponding class constructor table. Taking the axiom Mother ≡ Woman ⊓∃hasChild.Person as an example, we first define S1 for ∃hasChild.Person in Table SomeValuesFrom. Then I1, standing for the intersection of Woman and S1, will be defined in Table IntersectionClass. Finally, {Mother ⊑ I1, I1 ⊑ Mother} will be added to the SubClassOf table. Such a design is motivated by making the semantics of complex class description explicit. In this way, all class nodes in the OWL subsumption tree are materialized in database tables, and rule inference can thus be easier to implement and faster to execute via SQL statements. Also, a view named *Classes* is defined to provide an overall view of both named and anonymous classes in OWL ontologies.

The triple table of three columns (Subject, Property, Object) is also called a vertical table in data management. In (Agrawal et al., 2001), Agrawal et al. discussed the advantages of vertical tables over binary tables in terms of manageability and flexibility. Improved triple stores, including Minerva, generally adopt vertical tables to store ABox facts. The vertical table is efficient in space, but its retrieval often requires a 3-way join. This becomes a bottleneck in the case of complex queries or a large number of records involved, although using some indexes. Wang et al. (Wang et al., 2002) gives an insight into why the vertical table sometimes results in long query response time. Most relational databases transform a user query into a physical query plan which represents the operations, the method of performing the operations, and the order of processing the different operations (Garcia-Molina et al., 2000). A query optimizer of the database considers multiple physical plans and estimates their costs, and then selects a plan with the least estimated cost and passes it to the execution engine. So, the accuracy of the cost estimation seriously affects the efficiency of a query execution. Usually statistics collected from the base data are used to estimate the cost of a query plan. The query optimizer builds a histogram for each column. The histogram contains information about the distribution of the corresponding column and is stored in a database catalog (Wang et al., 2002, Poosala et al., 1996, Matias et al., 1998). Apparently, if the statistical information represented by the histogram is inaccurate, the query optimizer may make a wrong selection among different physical query plans. Since values of different properties are stored in the same column of the vertical table, the corresponding histogram can not accurately reflect the value distribution of each property. This may affect the query plan selection and execution of a query which needs to access information in the vertical table. Wang et al. proposed to build external histograms for values of different attributes and rewrite the physical query plan based on these external histograms. That is, with the external histograms, the DBMS query engine could generate an optimal query plan. Therefore, we can adopt this optimization method for the performance of triple stores. Sometimes, it is impossible to apply this method since one needs to access the core engine of the database. So, it is desirable to leverage indexes as much as possible to improve ontology repositories.

Currently, most commercial database systems provide primary clustering indexes. In this design, an index containing one or more keyparts could be identified as the basis for data clustering. All records are organized on the basis of their attribute values for these index keyparts by which the data is ordered on the disk. More precisely, two records are placed physically close to each other if the attributes defining the clustering index keyparts have similar values or are in the same range. Clustering indexes could be faster than normal indexes since they usually store the actual records within the index structure and the access on

the ordered data needs less IO costs. In practice, it is not suitable to create an index on a column with few distinct values because the index does not narrow the search too much. But, a clustering index on such a column is a good choice because similar values are grouped together on the data pages. Considering that real ontologies have a limited number of properties, the property column of triple tables, such as the RelationshipInd table of Minerva, could be a good candidate for clustering. So, it is valuable to use clustering indexes on triple tables for performance purpose.

Similar to normal unclustered indexes, the clustering index typically contains one entry for each record as well. More recently, Multi-Dimensional Clustering (MDC) (Bhatt et al., 2003) is developed to support block indexes which is more efficient than normal clustering indexes. Unlike the primary clustering index, an MDC index (also called MDC table) can include multiple clustering dimensions. Moreover, the MDC supports a new physical layout which mimics a multi-dimensional cube by using a physical region for each unique combination of dimension attribute values. A physical block contains only records which have the same unique values for dimension attributes and could be addressed by block indexes, a higher granularity indexing scheme. Block indexes identify multiple records using one entry and are thus quite compact and efficient. Queries using block indexes could benefit from faster block index scan, optimized prefetching of blocks, as well as lower path length overheads while processing the records. Evaluation results from (Brunner et al., 2007) showed that the MDC indexes could dramatically improve query performance (20 times faster and even more) and the set of indexes P*, (P,O), (S,P,O) on the triple table gives the best result for most queries on Minerva using DB2, where P* means an MDC index, other two represent composites unclustered indexes. Additionally, the MDC index could be built on the table defining *typeOf* information, grouping the records by classes.

Currently, the MDC index is a unique feature of DB2. But other popular databases provide advanced index functionalities as well. Oracle supports range partitioning which is a single dimension clustering of the data into partitions. It allows tables, indexes, and index-organized tables to be subdivided into smaller pieces, enabling these objects to be managed and accessed at a finer level of granularity. SQL Server and Teradata Non StopSQL support B+ tree tables. In this scheme, one can define the entire table as a B+ tree itself clustered on one or more columns. These features are helpful for the performance of triple stores.

2.3 A Scalable Ontology Reasoning Method by Summarization and Refinement

Reasoning algorithms that could be scaled to realistic databases are a key enabling technology for the use of ontologies in practice. Unfortunately, OWL-DL ontology reasoning using the tableau algorithm is intractable in the worst case. As we discussed previously, rule inference is adopted for OWL reasoning by some ontology repositories, and sometimes, inferred results are materialized for retrieval. But, rule inference cannot realize complete and sound reasoning of OWL-DL ontologies and maintaining the update of materialized results is also a non-trivial problem. Here, we introduce a novel method that allows for efficient querying of SHIN ontologies with large ABoxes stored in databases. Currently, this method focuses on instance retrieval that queries all individuals of a given class in the ABox. This summarization and refinement based method can also be treated as an optimization that any tableau reasoner can employ to achieve scalable ABox reasoning.

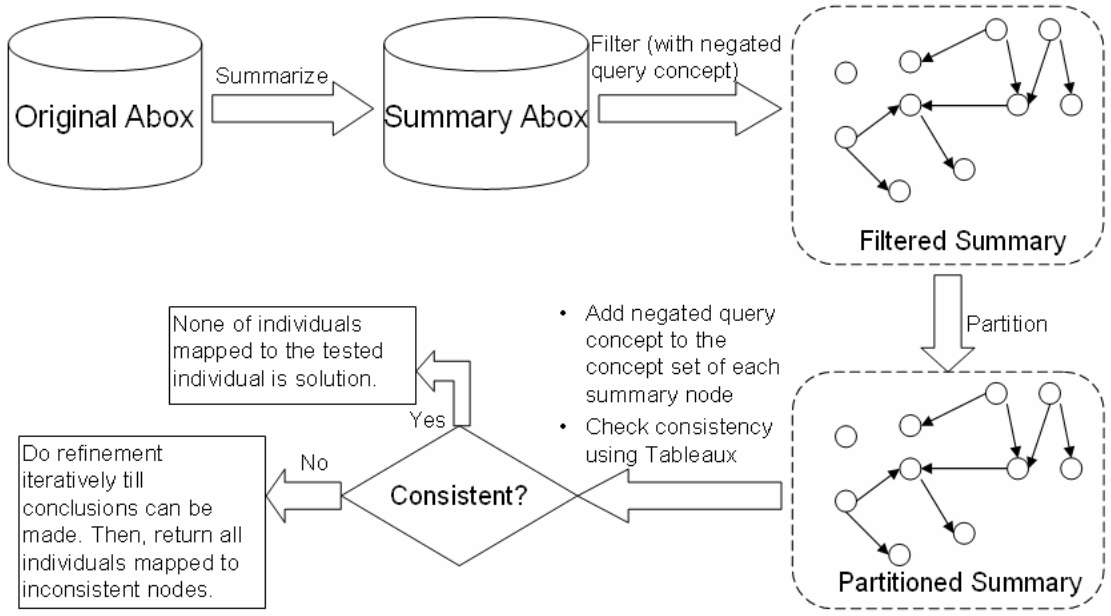

Figure -4. The Workflow of SHER Reasoner

It is well known that all queries over DL ontologies can be reduced to consistency check (Horrocks & Tessaris, 2002), which is usually checked by a tableau algorithm. As an example, an instance retrieval algorithm can be realized by testing if the addition of an assertion $a : \neg C$ for a given individual a results in an inconsistency. If the resulting ABox is inconsistent, then a is an instance of C. But, it is impractical to apply such a simple approach to every individual. Motivated by the fact that in most real ontologies: 1) individuals of the same class tend to have the same assertions with other individuals; 2) most assertions are in fact irrelevant for purposes of consistency check, Fokoue et al. (Fokoue et al., 2006) proposed to group individuals which are instances of the same class into a single individual to generate a summary ABox of a small size. Then, consistency check can be done on the dramatically simplified summary ABox, instead of the original ABox. By testing an individual in the summary ABox, all real individuals mapped to it are effectively tested at the same time. Figure 4 shows the workflow of the algorithm.

The first step is to construct a summary ABox *A'* corresponding to the original ABox *A*. This can be done by two steps: 1) Generate a single individual in the summary to represent all individuals which have the same concept set (the concept set consists of all classes of an individual). But individuals involved in *differentFrom* assertions are preserved in the summary. 2) Add a relation *R* to a pair of summary individuals *(C, D)* if there is an *R* edge between an individual mapped to *C* and another individual mapped to *D* in the ABox. Formally, we can use a mapping function f to describe the correspondence between *A* and *A'*. It satisfies the following constraints:

(1) if $a : C \in A$, then $f(a) : C \in A'$

(2) if $R(a, b) \in A$, then $R(f(a), f(b)) \in A'$

(3) if $a \neq b \in A$, then $f(a) \neq f(b) \in A'$

It is proven that if the summary ABox *A'* of ABox *A* is consistent w.r.t. an ontology's TBox, then *A* is consistent as well. However, the converse does not hold. This is because the summarization may cause inconsistencies.

Next is to filter out role assertions that cannot be responsible for the detection of an inconsistency in the ABox, either because they cannot be used to propagate a concept assertion, or because they cannot be involved in the detection of an inconsistency due to a merger of ABox individuals. For the SHIN sub-language of DL, these are role assertions where the roles are not specified in any universal restriction or a maximum cardinality

restriction in *clos(A)*. Note that the *clos(A)* includes the negated query, because the queried concept is effectively a part of the summary ABox. This filtering step further reduces the size of the summary ABox that is used as a starting point for instance retrieval.

When the summary ABox contains disconnected sub-graphs, each isolated sub-graph can be processed separately. Since the algorithm currently works for the description logic SHIN, which does not contain nominals, it is safe to partition the ABox without affecting soundness and completeness of the instance retrieval algorithm. Note that individuals in disconnected partitions can only interact via axioms in the TBox by using nominals. The partitioning strategy works well in a lot of realistic large ontologies where the class hierarchy is spread out, typically observed when dealing with separate domains or specializing in numerous areas. In such cases, there exist a lot of disconnections between sub-ABoxes that are tied into separate class hierarchies. In addition, filtering out irrelevant role assertions that connect instances together in practice produces a large number of disconnected partitions. Partitioning presents a great opportunity for parallelization since consistency check can be executed on each separate partition simultaneously with the results being combined at the end.

Each individual s in the summary ABox A' is tested by adding an assertion $s : \neg$QueriedConcept to A', and checking for consistency using a tableau reasoner, such as Pellet reasoner. To achieve scalability, one can test multiple individuals in the summary graph at the same time.

Definition A. Let A be an ABox. Let Q be a concept expression. Let S be a subset of individuals in A such that for all $s \in S$, $s : \neg Q \notin A$. 1 We define the tested ABox w.r.t. A, Q and S, denoted *tested(A,Q, S)*, to be the ABox obtained from A by adding the assertion $s : \neg Q$ for each $s \in S$. Formally, $tested(A,Q, S) = A \cup \{s : \neg Q | s \in S\}$.

If the result of testing a single individual s is consistent, then we know that none of real individuals mapped to s is a query solution. However, if the result is inconsistent, then we cannot conclude anything about individuals mapped to s. This situation arises because individuals are aggregated based only on the similarity of their concepts, not relationships. One approach for resolving summary ABox inconsistencies is to iteratively refine the summary. Refinement partitions the set of individuals mapped to a single summary individual and remaps each partition to a new summary individual. Obviously, refinement increases the size and precision of the summary, and preserves three properties defined by the above summary mapping function. Here, the strategy is to refine only individuals that are part of a summary ABox justification, where a justification is a minimal set of assertions which, when taken together, imply a logical contradiction, thus making the entire ABox inconsistent. In addition to guiding refinement, justifications are helpful for users to understand query results. Since justifications are at a summarized level, the information is more useful than detailed information about each individual in an ABox. In some cases, inconsistencies disappear through refinement. Otherwise, when a justification J is precise, we typically know that we have converged on a solution. That is, there is a tested individual s in J, such that all real individuals mapped to s are instances of the query. We say that a tested individual s is tested in J for query Q if $s : \neg Q$ is an assertion in J .

Definition B. Let A' be a summary ABox of an ABox A obtained through the summary mapping f . Let Q be a queried concept, S be a subset of individuals in A' such that for all $x \in S$, $x : \neg Q \notin A'$ and let H be a subset of $tested(A', Q, S)$. We say that an individual $s \in H$ is precise w.r.t. H iff the following conditions are satisfied:

1. for all individuals $t \in H$ and for all roles R, $R(s, t) \in H$ (resp. $R(t, s) \in H$) implies that, for all individuals $a \in A$ such that $f(a) = s$, there is an individual $b \in A$ such that $f(b) = t$ and $R(a, b) \in A$ (resp. $R(b, a) \in A$); and

2. for all individuals s, $t \in H$, $s \neq t \in H$ (resp. $t \neq s \in H$) implies that, for all individuals $a \in A$ such that $f(a) = s$, there is an individual $b \in A$ such that $f(b) = t$ and $a \neq b \in A$ (resp. $b \neq a \in A$); and

3. There is an individual $a \in A$ such that $f(a) = s$; and

4. $s : C \in H - \{x : \neg Q | x \in S\}$ implies that, for all individuals $a \in A$ such that $f(a) = s$, $a : C \in A$

We say that H is precise iff all its individuals are precise w.r.t. H.

In summary, a high level outline of the algorithm is shown below.

```
S←{x|x ∈ individuals in A ′and x : ¬Q∉ A ″};
R←A ‴;
Results←∅;
while S ≠ ∅ do
    RT←tested(R,Q, S) (see Definition A) ;
    if consistent(RT ) then
        return Results;
    end
    Find Justifications in RT ;
    T←individuals tested in precise Justifications;
    Results←Results ∪ Image(T);
    S←S − T;
    Execute refinement strategy on R ;
end
return Results
```

The SHER reasoner (Dolby et al., 2007) implemented this reasoning approach on top of Minerva's storage component (Zhou et al., 2006) and proved its effectiveness and efficiency on the UOBM benchmark ontology. It is reported that SHER can process ABox queries with up to 7.4 million assertions efficiently, whereas the state of the art reasoners could not scale to this size.

2.4 Other approaches to scaling reasoning over large knowledge bases

The issue of scaling reasoning over large ABoxes has recently received a lot of attention from the Semantic Web and Description Logics communities. Two main approaches have been proposed to tackle it. The first approach consists in building new algorithms, heuristics and systems that exhibit acceptable performance on realistic large and expressive knowledge bases. Proponents of the second approach, on the other hand, advocate reducing the expressiveness of TBoxes describing large ABoxes so that the worst-case data complexity[1] of reasoning becomes tractable. The summarization and refinement technique to scale reasoning over large and expressive ABoxes presented in the previous section is an illustration of research work guided by the first approach. In this section, we present other important recent work on reasoning over large and expressive knowledge bases as well as Description Logics that have been defined with a tractable worst-case data complexity.

Since state-of-the-art in-memory reasoners, such as Pellet (Sirin & Parsia, 2004) and Racer (Haarslev & Moller, 2001), offer good performance on realistic expressive but small knowledge bases, Guo et al. have recently proposed to decompose large and expressive ABoxes into possibly overlapping small components that could be separately fed to state-of-the-art in-memory reasoners. The decomposition is such that the answer to a conjunctive query over the original ABox is the union of the answers of the same conjunctive query over each component of the decomposition. Conservative analyses of the inference rules of the considered DL provide the understanding of interdependency between ABox assertions. Two ABox assertions depend on each other if they might be used together to infer new assertions. The decomposition is such that two assertions that depend on each other always appear together in a component. Results of initial experimental evaluation presented in (Guo & Heflin, 2006) are very promising. Another approach (Hustadt et al., 2004) to

[1] Data complexity refers to the complexity of reasoning over the ABox only assuming that the TBox is fixed. It measures the complexity of reasoning as a function of the ABox size only.

efficiently answer conjunctive queries over large and expressive knowledge bases consists in transforming any SHIN(D)[2] knowledge base into a disjunctive datalog program. The advantages of this approach are twofold. First, it leverages decades of research on optimizations of disjunctive datalog programs (e.g. magic set transformation). Second, it naturally supports DL-safe rules (Motik et al., 2004), which can straightforwardly be translated into datalog rules.

Other researchers have advocated reducing the expressive power in order to obtain tractable reasoning over large ABoxes. Calvanese et al. have introduced a family of inexpressive Description Logics, the DL-Lite family, with data complexity varying from LogSpace to co-NP-hard (Calvanese et al., 2006). DL-Lite$_{core}$, the least expressive language in the DL-Lite family, consists of existential restriction and a restricted form of negation (Calvanese et al., 2005). The language for DL-Lite$_{core}$ concepts and roles is defined as follows:

$$C_l \rightarrow A \mid \exists R; C_r \rightarrow A \mid \exists R \mid \neg A \mid \neg \exists R$$
$$R \rightarrow P \mid P^-$$

where C_l (resp. C_r) denotes a concept that may appear in the left (resp. right) hand side of a concept inclusion axiom in the TBox. Two simple extensions of DL-Lite$_{core}$, DL-Lite$_{F,6}$ and DL-Lite$_{R,6}$, have been defined and shown to be FOL-reducible: i.e. answering a conjunctive query in DL-Lite$_{core}$ or in one of these extensions can be reduced to evaluating a SQL query over the database corresponding to the ABox. The advantages of these FOL-reducible languages are straightforward for applications with very limited expressiveness needs. DL-Lite$_{F,6}$ extends DL-Lite$_{core}$ by allowing intersections on the left hand side of concept inclusion axioms and functional roles; while DL-Lite$_{R,6}$ extends DL-Lite$_{core}$ by allowing inclusion axioms between roles, intersections on the left hand side of concept inclusion axioms, and qualified existential restrictions on the right hand side of concept inclusion axioms. All the other extensions[3] to DL-Lite$_{core}$ are not FOL-reducible, but, for the most part, they remain tractable. Other Description Logics with polynomial data complexity include Horn-SHIQ (Hustadt et al., 2005, Krotzsch et al., 2006), a fragment of SHIQ analogous to the Horn fragment of first-order logic, and description logic programs (Grosof et al., 2003).

2.5 Bridging Discrepancies Between OWL ontology and Database

Recently, Semantic Web and ontologies are receiving extensive attention from data management area. Ontologies are used as semantic models which are believed to be able to represent more semantics of the underlying data and are easy to understand. OWL provides numerous constructs to define complex and expressive models. However, it is gradually recognized that there are remarkable discrepancies between description logics (the logical foundation of OWL) and databases. As is well-known, DL is based on an open world assumption (OWA) permitting incomplete information in an ABox, while DB adopts a closed world assumption (CWA) requiring information always understood as complete. The unique name assumption (UNA) is often emphasized in DB but not in DL. OWL Flight (Bruijn et al., 2005), furthermore, clarifies restrictions in DL and constraints in DB, of which the former is to infer and the latter to check. When negation comes, DBs prefer to "non-monotonic negation", while DLs rely on "monotonic negation". The following simple example gives us an intuitive understanding of such discrepancies. In a relational database, if "each employee must be known to be either male or female" is specified as an integrity constraint, the database system would check whether the gender of a person is given and set to be male or female during database updates. If the gender is not specified as male or female, the update would fail. In an ontology, the same requirement would naturally be represented by an axiom that Employee is subsumed by the union of Male and Female. Adding an employee without

[2] SHIN(D) is the subset of OWL DL without nominal.
[3] We are not considering extension allowing n-ary predicate with n>2.

expressing he/she is an instance of Male or Female to the ontology would not result in any errors, and just imply that the employee could be either Male or Female.

Some research work on extending DLs with integrity constraints are mainly based on autoepistemic extensions of DLs, such as the description logics of minimal knowledge and negation-as-failure (MKNF) (Donini et al., 2002) and some nonmonotonic rule extensions of DLs (Motik et al., 2007). This may be inspired by Reiter's observation that integrity constraints describe the state of the database and have an epistemic nature (Reiter, 1992). Motivated by representing integrity constraints in MKNF, Mei et al. imposed epistemic operators on union and existential restrictions and explained them using integrity constraints in an ontology (Mei et al., 2006). Given the ABox of an SHI ontology is satisfiable with regard to its epistemic TBox, reasoning on such an ontology could be done by a datalog program.

More recently, Boris et al. proposes an extension of OWL that attempts to mimic the intuition behind integrity constraints in relational databases (Motik et al., 2006). Integrity constraints, introduced in (Mei et al., 2006), are used for conveying semantic aspects of OWL that are not covered by deductive databases, while (Motik et al., 2006) extends standard TBox axioms with constraint TBox axioms, s.t., for TBox reasoning, constraints behave like normal TBox axioms; for ABox reasoning, however, they are interpreted in the spirit of relational databases. Acting as checks, constraints are thrown away, if satisfied, without losing relevant consequences. Algorithms for checking constraint satisfaction are also discussed in (Motik et al., 2006), and the complexity of constraint checking is primarily determined by the complexity of the standard TBox. As a result, answering queries under constraints may be computationally easier due to a smaller input of the standard TBox concerning. Currently, (Motik et al., 2006) plans to implement such an approach in the OWL reasoner KAON2 and tests its usefulness on practical problems.

Technically, (Motik et al., 2006) defines constraints in the same way as subsumptions, having the form of $C \sqsubseteq D$ where C and D are DL concepts. Keeping the semantics of DLs unchanged, constraints rely on Herbrand models for checking satisfiability. Query answering is another reasoning service, provided the constraints are satisfied, and again uses the standard semantics of DLs after throwing those constraints away. That is, authors define TBox axioms, of which some are for inferring (namely, standard TBox axioms) and some for checking (namely, constraint TBox axioms). The extended DL system will provide support for DL reasoning as usual, in addition to checking constraint satisfiability using the well-known methods of logic programming.

By definition, an extended DL knowledge base is a triple $K=(S, C, A)$ such that S is a finite set of standard TBox axioms, C is a finite set of constraint TBox axioms, and A is a finite set of ABox assertions, $D(a)$, $\neg D(a)$, $R(a,b)$, $a \approx b$, $a \neq b$, for D an atomic concept, R a role, and a, b individuals. Checking C in the minimal models of $A \cup S$, the algorithm is sketched as follows (Motik et al., 2006).

1. The standard TBox S is translated into a first-order formula $\pi(S)$ according to the standard DL semantics, and the results are further translated into a (possibly disjunctive) logic program $LP(S) = LP(\pi(S))$ which can be exponentially larger than S. For each rule in $LP(S)$ in which a variable x occurs in the head but not in the body, the atom $HU(x)$ is added to the rule body. Additionally, for each individual a occurring in $A \cup S$, an assertion $HU(a)$ is introduced.
2. The constraint TBox C is translated into a first-order formula $\pi(C)$, and $CN(C) = CN(\pi(C))$ is constructed as a stratified datalog program. For each formula φ, a unique predicate E_φ is associated, also $\mu(\varphi)$ and $sub(\varphi)$ are defined, where $\mu(\varphi)$ is a translation rule for φ and $sub(\varphi)$ is the set of sub-formulae of φ, s.t. the following logic program is computed: $CN(\varphi) = \mu(\varphi) \cup \bigcup_{\phi \in sub(\varphi)} CN(\phi)$.

As a consequence, $K=(S, C, A)$ satisfies the constraint TBox C if and only if $A \cup LP(S) \cup CN(C) \models_c E_C$, where $\models_c$ denotes the well-known entailment in stratified (possibly disjunctive) logic programs, and $E_C = E_{\pi(C)}$.

Intuitively, *CN(C)* simply evaluates *C* and ensures that E_C holds in a model if and only if *C* is true in the model. Thus, E_C is derived if and only if *C* is satisfied in all minimal models. Finally, suppose *K=(S, C, A)* be an extended DL knowledge base that satisfies *C*. For any union of conjunctive queries $\gamma(v)$ over *K=(S, C, A)* and any tuple of constants *u*, it holds that $A \cup S \cup C \models \gamma(u)$ if and only if $A \cup S \models \gamma(u)$.

Not surprising, in query answering, constraints are thrown away, if they are satisfied. All other reasoning problems look like before, and the existing DL algorithms can be applied to solve them.

3. REASONING WITH WSML-DL

In this section, we take the approach of looking at another practical language for ontology reasoning. We focus on reasoning with the Description Logic-based Ontology language WSML-DL. We use WSML-DL as a more intuitive surface syntax for an expressive Description Logic (DL) in the WSML family of knowledge representation languages. Its syntax is inspired by First-order Logic modelling style.

WSML-DL is less expressive than OWL DL, given that WSML-DL does not support nominals. This reduces the complexity of WSML-DL, which is important for reasoning. In fact, until recently many state-of-the-art DL reasoners did not support reasoning with nominals, since no good optimization techniques were known.

To enable the use of existing DL reasoning engines for WSML, we transform WSML-DL to OWL DL. This is because OWL DL is the appropriate syntax for DL reasoners as e.g. Pellet or KAON2. Then we integrate the reasoners into a flexible WSML reasoner framework.

In the following, we first point out the particularities of DL reasoning. Next we describe the WSML-DL syntax and its correspondence to DLs. We show the translation from WSML-DL to OWL DL abstract syntax and explain the architecture and implementation of the WSML2Reasoner framework.

3.1 Reasoning with Description Logics

Description Logics can be seen as particularly restricted subset of Predicate Logic and constitute a family of logic-based knowledge representation formalisms. They have become a cornerstone of the Semantic Web for its use in the design of ontologies.

DL knowledge bases are separated into two components: TBoxes, containing the terminological knowledge of a knowledge base (e.g. concept definitions), and ABoxes, containing the assertional knowledge (knowledge about the individuals of a domain).

In DLs, there are different basic reasoning tasks for reasoning with TBoxes or ABoxes. As described in Baader et al. (2003), the main inference procedures with TBoxes are concept subsumption and concept satisfiability. With ABoxes, the main reasoning tasks are ABox consistency and instance checking.

The OWL community focuses on entailment and query answering as the key inference services. Entailment can be reduced to satisfiability, while query answering amounts to compute the result of a query for instances with specific properties over a database, or an ABox respectively.

Descriptions of the main standard DL reasoning tasks, as well as of some main non-standard inference tasks can be found in Baader et al. (2003).

3.2 WSML-DL

The Web Service Modeling Language WSML (de Bruijn et al., 2005) is a family of formal Web languages based on the conceptual model of WSMO (Roman et al., 2004). Conforming to different influences, as e.g. Description Logics (Baader et al., 2003), Logic Programming (Lloyd, 1987) and First-order Logic (Fitting, 1996), there exist five variants of WSML: WSML-Core, WSML-DL, WSML-Flight, WSML-Rule and WSML-Full.

The WSML-DL variant captures the expressive Description Logic SHIQ(D). The following sections will introduce the WSML-DL syntax and its correspondence to Description Logics.

3.2.1 WSML-DL Syntax

WSML makes a clear distinction between the modelling of conceptual elements (Ontologies, Web Services, Goals and Mediators) and the specification of logical definitions. Therefore the WSML syntax is split in two parts: the conceptual syntax and the logical expression syntax. The following sections will provide an overview of the WSML-DL conceptual and the logical expression syntax. A more detailed description can be found in de Bruijn et al. (2005).

3.2.1.1 WSML-DL Conceptual Syntax

A WSML ontology specification may contain concepts, relations, instances, relation instances and axioms. Concepts form the basic terminology of the domain of discourse and may have instances and associated attributes. A concept can be defined as subconcept of another concept, and in this case, a concept inherits all attribute definitions of its superconcept.

A concept may have an arbitrary number of instances associated to it. The instance definition can be followed by the attribute values associated with the instance. Instead of being explicitly defined in the ontology, instances can exist outside the ontology in an external database.

There are two sorts of attribute definitions that a concept may contain: inferring definitions with the keyword impliesType and constraining definitions with the keyword ofType. The constraining definitions may only be used for datatype ranges. Inferring attribute definitions are similar to range restrictions on properties in RDFS (Brickley and Guha, 2004) and OWL (Bechhofer et al., 2004).

In WSML-DL only binary relations are allowed. They correspond to the definition of attributes. The usage of inferring and constraining definitions in relations corresponds to their usage in attribute definitions. A relation can be defined as a subrelation of another relation.

A relation may contain relation instances with parameter values associated to it.

Axioms can be used to refine the definitions already given in the conceptual syntax, e.g. the subconcept and attribute definitions of concepts. By defining respective axioms one can define cardinality restrictions and global transitivity, symmetricity and inversity of attributes, just like in DLs or OWL. The logical expression syntax is explained in the following section.

3.2.1.2 WSML-DL Logical Expression Syntax

The form of WSML-DL logical expressions and their expressiveness is based on the Description Logic SHIQ(D). The WSML-DL logical expression syntax has constants, variables, predicates and logical connectives, which all are based on First-order Logic modelling style.

An atom in WSML-DL is a predicate symbol with one or two terms as arguments. WSML has a special kind of atoms, called molecules. There are two types of molecules that are used to capture information about concepts, instances, attributes and attribute values: “isa

molecules", that are used to express concept membership or subconcept definitions, and "object molecules", that are used to define attribute and attribute value expressions.

These molecules build the set of atomic formulae in WSML-DL. Using First-order connectives, one can combine the atomic formulae to descriptions and formulae. How exactly the molecules can be combined to build descriptions and formulae, can be seen in detail in de Bruijn et al. (2005).

3.2.2 WSML-DL vs. SHIQ(D)

Table 1 illustrates the relationship between the WSML-DL semantics, the Description Logics syntax and the OWL DL syntax. The table follows de Bruijn et al (2005), Volz (2004) and Borgida (1996).

In the table, "id" can be any identifier, "dt" is a datatype identifier, "X" can be either a variable or an identifier and "Y" is a variable.

Table -1. WSML-DL logical expressions - DL syntax

WSML-DL	DL Syntax	OWL DL
τ(lexpr **impliedBy** rexpr)	$rexpr \subseteq lexpr$	subClassOf
τ(lexpr **or** rexpr)	$lexpr \cup rexpr$	unionOf
τ(lexpr **and** rexpr)	$lexpr \cap rexpr$	intersectionOf
τ(**neg** expr)	$\neg expr$	complementOf
τ(**forall** Y expr)	$\forall R.expr$	allValuesFrom
τ(**exists** Y expr)	$\exists R.expr$	someValuesFrom
τ(X **memberOf** id)	$X : id$	Type
τ(id1 **subConceptOf** id2)	$id1 \subseteq id2$	subClassOf
τ(X1[id **hasValue** X2])	$< X1, X2 > : id$	Property
τ(id1[id2 **impliesType** id3])	$id1 \subseteq \forall id2.id3$	subPropertyOf
τ(id1[id2 **ofType** dt])	$id1 \subseteq \forall id2.dt$	subPropertyOf
τ(p(X_1,…,X_n))	$< X_1,...X_n > : p$	Type
τ(X1 :=: X2)	$X1 \equiv X2$	equivalentClass

3.3 Translation of WSML-DL to OWL DL

The following sections show the translation from WSML-DL to OWL DL abstract syntax (Steinmetz, 2006). The mapping is based on a mapping from WSML-Core to OWL DL, which can be found in de Bruijn et al. (2005), and can be applied to WSML ontologies and logical expressions.

3.3.1 Transformation Steps

The transformation of a WSML-DL ontology to an OWL DL ontology is done in a line of single transformation steps that are executed subsequently.

- Relations, subrelations and relation instances are replaced by attributes and axioms, according to the preprocessing steps described in Steinmetz (2006).
- All conceptual elements are converted into appropriate axioms specified by logical expressions. The resulting set of logical expressions is semantically equivalent to the original WSML ontology.
- Equivalences and right implications in logical expressions are replaced by left implications.
- Conjunctions on the left side and disjunctions on the right side of inverse implications are replaced by left implications.
- Complex molecules inside of logical expressions are replaced by conjunctions of simple ones.

As last step, the resulting axioms and logical expressions are transformed one by one into OWL Descriptions according to the mapping presented in the following section.

3.3.2 Mapping Tables

Table 2 and Table 3 contain the mapping between the WSML-DL syntax and the OWL DL abstract syntax. The mapping is described through the mapping function τ. In Table 3 we will introduce the functions α and ε, which are needed for the correct translation of WSML-DL descriptions.

Boldfaced words in the tables refer to keywords in the WSML language. "X" and "Y" are meta-variables and are replaced with actual identifiers and variables during the translation, while "DES" stands for WSML-DL descriptions. IRIs[4] are abbreviated by qualified names. The prefix 'wsml' stands for 'http://wsmo. org/wsml/wsml-syntax#' and 'owl' stands for 'http://www.w3.org/2002/07/owl#'.

Table -2. Mapping WSML-DL ontologies and axioms to OWL DL

WSML-DL	OWL-DL	Remarks
Mapping for ontologies		
τ(**ontology** id $header_1$ … $header_n$ $ontology\_element_1$ … $ontology\_element_n$)	Ontology(id $\tau(header_1)$ … $\tau(header_n)$ $\tau(ontology\_element_1)$ … $\tau(ontology\_element_n)$)	A header can contain *nonFunctionalProperties*, *usesMediator* and *importsOntology* statements. An *ontology_element* can be a *concept*, a *relation*, an *instance*, a *relation instance* or an *axiom*.
τ(**nonFunctionalProperties** id_1 **hasValue** $value_1$ … id_n **hasValue** $value_n$ **endNonFunctionalProperties**)	$Annotation(id_1\ \tau(value_1))$ … $Annotation(id_n\ \tau(value_n))$	For non functional properties on the ontology level "Annotation" instead of "annotation" has to be written.
τ(**importsOntology** id)	Annotation(owl#import id)	"id" stands for the identifier of a WSML file.
τ(**usesMediator** id)	Annotation(wsml#usesMediator id)	As OWL doesn't have the concept of a mediator, a wsml#usesMediator annotation is used.
$\tau(datatype\_id(x_1,\ldots,x_n))$	$datatype\_id(x_1,\ldots,x_n)\text{^^}$ $\tau_{datatypes}(datatype\_id)$	$\tau_{datatypes}$ maps WSML datatypes to XML Schema datatypes, according to de Bruijn et al. (2005).
τ(id)	id	In WSML an IRI is enclosed by _" and ", which are omitted in OWL abstract syntax.
Mapping for axioms		
τ(**axiom** id log_expr **nfp**)	τ(log_expr)	A log_expr can be a logical expression like the following. The axiom does not keep its non functional properties.
τ(id[att_id **impliesType** range_id])	Class(id restriction (att_id allValuesFrom range_id)) ObjectProperty (att_id)	

[4] http://www.ietf.org/rfc/rfc3987.txt

WSML-DL	OWL-DL	Remarks
τ(id[att_id **ofType** range_id])	Class(id restriction (att_id allValuesFrom range_id)) DatatypeProperty (att_id)	
τ(id1 **subConceptOf** id2)	Class(id1 partial id2)	
τ(id[att_id **hasValue** value])	Individual (id value (att_id τ(value)))	
τ(id1 **memberOf** id2)	Individual(id1 type(id2))	
τ(?x[att_id2 **hasValue** ?y] **impliedBy** ?x[att_id **hasValue** ?y])	SubProperty(att_id att_id2)	A left implication with attribute values as left-hand and right-hand sides is mapped to an OWL subProperty.
τ(?x[att_id **hasValue** ?y] **impliedBy** ?x[att_id **hasValue** ?z] and ?y[att_id **hasValue** ?z])	ObjectProperty(att_id Transitive)	Transitive Property
τ(?x[att_id **hasValue** ?y] **impliedBy** ?y[att_id **hasValue** ?x])	ObjectProperty(att_id Symmetric)	Symmetric Property
τ(?x[att_id **hasValue** ?y] **impliedBy** ?y[att_id2 **hasValue** ?x])	ObjectProperty(att_id inverseOf(att_id2))	Inverse Property
τ(?x **memberOf** concept_id2 **impliedBy** ?x **memberOf** concept_id)	Class(concept_id partial concept_id2)	Equivalence of concepts can be expressed as follows, with A and B being membership molecules: “A equivalent B” :=: “A impliedBy B and B impliedBy A”.
τ(?x **memberOf** concept_id **impliedBy** ?x[att_id **hasValue** ?y])	ObjectProperty(att_id domain(concept_id))	
τ(?y **memberOf** concept_id **impliedBy** ?x[att_id **hasValue** ?y])	ObjectProperty(att_id range(concept_id))	
τ(DES1 **impliedBy** DES2)	α(DES1) α(DES2) subClassOf(ε(DES2) ε(DES1))	“A impliedBy B” can be written as “subClassOf(B,A)”.
τ()		If τ is applied for a non-occurring production no translation has to be made

Table 3 shows the mapping of WSML-DL descriptions that are used inside of axioms, as can be seen in Table 2. The descriptions are translated to concept expressions and to axioms. Concept expressions are again used within other expressions, while the axioms are added as such to the OWL ontology. The mapping τ is translated into a tuple of concept expressions and axioms as follows: τ (DES) = (ε (DES), α (DES)).

The table also indicates a mapping for Qualified Cardinality Restrictions (QCRs). In WSML-DL the QCRs are represented by a combination of WSML-DL descriptions. The mapping to OWL DL is done according a workaround with OWL subproperties, described in Rector (2003).

Table -3. Mapping WSML-DL descriptions to OWL DL

WSML-DL	OWL-DL – concept expression ε	OWL-DL – axiom α	Remarks
Mapping for descriptions (DES)			

WSML-DL	OWL-DL – concept expression ε	OWL-DL – axiom α	Remarks
τ(?x **memberOf** id)	id	Class(id)	Membership molecule.
τ(?x[att_id **hasValue** ?y])	restriction(att_id allValuesFrom(owl:Thing))	ObjectProperty(att_id)	Attribute value molecule with ?y being an unbound variable within the logical expression.
τ(?x[att_id **hasValue** ?y] **and** ?y **memberOf** id)	restriction (att_id someValuesFrom(id))	Class(id) ObjectProperty(att_id)	Attribute value molecule with ?y being a bound variable.
τ(DES_1 **and** … **and** DES_n)	intersectionOf(ε(DES_1),…,ε(DES_n))	α(DES_1) … α(DES_n)	Conjunction.
τ(DES_1 **or** … **or** DES_n)	unionOf(ε(DES_1),…,ε(DES_n))	α(DES_1) … α(DES_n)	Disjunction.
τ(**neg** DES)	complementOf(ε(DES))	α(DES)	Negation.
τ(**exists** ?x (?y[att_id **hasValue** ?x] **and** DES))	restriction(att_id someValuesFrom(ε(DES)))	α(DES) ObjectProperty(att_id)	Existential quantification.
τ(**exists** ?x (?x[att_id **hasValue** ?y] **and** DES))	restriction(inverseOf(att_id) someValuesFrom(ε(DES)))	α(DES) ObjectProperty(att_id)	Existential quantification with inverse role.
τ(**forall** ?x (DES **impliedBy** ?y[att_id **hasValue** ?x]))	restriction(att_id allValuesFrom(ε(DES)))	α(DES) ObjectProperty(att_id)	Universal quantification.
τ(**forall** ?x (DES **impliedBy** ?x[att_id **hasValue** ?y]))	restriction(inverseOf(att_id) allValuesFrom(ε(DES)))	α(DES) ObjectProperty(att_id)	Universal quantification with inverse role.
τ(**exists** ?y1,…,?yn (?x [att_id **hasValue** ?y1] **and** … **and** ?x[att_id **hasValue** ?yn] **and** DES **and** **neg**(?y1 :=: ?y2) **and** … **and neg**(?yn-1 :=: ?yn)))	restriction(att_id' minCardinality(n))	α(DES) ObjectProperty(att_id) ObjectProperty(att_id' range(ε(DES))) SubPropertyOf(att_id' att_id)	(Qualified) minCardinality restriction.
τ(**forall** ?y1,…,?yn+1 (?y1 :=: ?y2 **or** … **or** ?yn :=: ?yn+1 **impliedBy** ?x[att_id **hasValue** ?y1] **and** … **and** ?x[att_id **hasValue**	restriction(att_id' maxCardinality(n))	α(DES) ObjectProperty(att_id) ObjectProperty(att_id' range(ε(DES))) SubPropertyOf(att_id' att_id)	(Qualified) maxCardinality restriction.

WSML-DL	OWL-DL – concept expression ε	OWL-DL – axiom α	Remarks
?yn+1] **and** DES)			

3.3.3 Restrictions to the Transformation

The transformation is not complete, i.e. WSML-DL supports features that cannot be expressed in OWL DL and that can thus not be translated. Concretely, OWL DL does not support datatype predicates. They are lost during the transformation.

3.3.4 Translation Example

Table 4 shows two simple translation examples of both WSML-DL conceptual syntax and logical expression syntax. More examples can be found in Steinmetz (2006).

Table -4. Translation Example

WSML-DL	OWL DL
concept Human hasChild **impliesType** Human hasBirthday **ofType** date	ObjectProperty(hasChild domain(Human) range(Human)) DatatypeProperty(hasBirthday domain(Human) range(xsd:date)) Class(Human partial)
axiom definedBy ?x **memberOf** Man **implies neg**(?x **memberOf** Woman).	Class(Man partial) Class(Woman partial) SubClassOf(Man complementOf(Woman))

3.3.5 Architecture and Implementation

In the following we will discuss the architecture and the implementation of a reasoner prototype that allows us to perform reasoning with WSML-DL ontologies using state-of-the-art reasoning engines by means of a wrapper component.

The WSML2Reasoner framework[5] is a flexible and highly modular architecture for easy integration of external reasoning components. It has been implemented in Java and is based on the WSMO4J[6] project, which provides an API for the programmatic access to WSML documents. Instead of implementing new reasoners, existing reasoner implementations can be used for WSML through a wrapper that translates WSML expressions into the appropriate syntax for the reasoner.

As already said above, the appropriate syntax for many DL Reasoners is OWL DL. We have implemented the transformation from WSML-DL to OWL DL using the Wonderweb OWL API (Bechhofer et al., 2003). The OWL API allows a programmatic access to OWL ontologies. It offers a high-level abstraction from the Description Logics underlying OWL DL, what increases the usage of DL knowledge bases in the Semantic Web area.

The WSML2Reasoner framework infrastructure offers an interface that represents a façade to various DL reasoning engines. The façade provides a set of DL usual reasoning task methods and mediates between the OWL DL ontologies produced by the transformation and the reasoner-specific internal representations. For each new DL reasoning engine that is integrated into the framework, a specific adapter façade has to be implemented.

The framework currently comes with façades for two OWL DL reasoners: Pellet[7] and KAON2[8]:

[5] http://tools.deri.org/wsml2reasoner/
[6] http://wsmo4j.sourceforge.net/
[7] http://pellet.owldl.com/

- **Pellet** – Pellet is an open-source Java based OWL DL reasoner. It can be used directly in conjunction with the OWL API.
- **KAON2** – KAON2 is an infrastructure to manage, amongst others, OWL DL ontologies. It provides a hybrid reasoner that allows datalog-style rules to interact with structural Description Logics knowledge bases.

4. SEMANTIC BUSINESS PROCESS REPOSITORY

In the final section of this chapter, we take a look at a practical use of ontological reasoning with large instance data.

4.1 Requirements Analysis

In general, a repository is a shared database of information about engineered artifacts produced or used by an enterprise (Bernstein et al. 1994). In SBPM, these artifacts are semantic business process models (process models for short).

Process models are modeled by business users with help of a process modeling tool. To support process modeling, the SBPR has to provide standard functionality of a Database Management System, such as storage of new process models, update, retrieval or deletion of existing process models, transaction support for manipulation of process models and query capability. The query capability enables business users or client applications to search process models in the SBPR based on the criteria specified. We classify the queries into two categories. The first category of queries can be answered based on the artifacts explicitly stored in the SBPR. This kind of queries is of the same kind as the queries that traditional database systems can process. The second category of queries is "semantic queries", which can only be processed, when the ontological knowledge of the process models is taken into account.

The modeling of process models can be a time-consuming task. It may take days or even months for business users to finish modeling a given business process. Therefore, treating the entire modeling activity related to a process model as a single transaction is impractical. The SBPR has to provide check-in and check-out operations, that support long running interactions, enable disconnected mode of interaction with the SBPR, and are executed as separate short transactions. In this case the modeling tool could work in a disconnected mode regarding the SBPR. The process model in the SBPR can be locked when the modeling tool obtains it (check-out), so that no other users can modify the process model in the SBPR in the meantime. After the modeling work has been done the process model is updated in the SBPR and any locks that have been held for the process model are released (check-in). Please note that the locking mechanism refers only to the locking of the process models in the SBPR. The process ontologies, that are stored separately in an ontology store and have been referenced by the process models, are not locked simultaneously. Furthermore, in a distributed modeling environment several business users may work on the same process model simultaneously. A fine-grained locking of elements in a process model enables different business users to lock only the part of the process model they are working on, thus avoiding producing inconsistent process models.

Process models may undergo a series of modifications undertaken by business users. The series of modification is called change history of the process model. The SBPR represents the change history as versions. A version is a snapshot of a process model at a certain point in its change history (Bernstein et al. 1994). In certain industry sectors corporations must record all the change histories of their process models for government auditing or for some legal requirements. From the modeling perspective it is meaningful to keep process models in

[8] http://kaon2.semanticweb.org/

different versions, so that business users can simply go back to an old version and develop the process model from the old version further. Due to these reasons the SBPR has to provide also versioning functionality, so that the change history of process models can be documented.

4.2 Comparison of Storage Mechanisms

As storing and querying process models stored are the main requirements for the SBPR, we evaluate in this section several options for storage mechanism and their query capabilities.

A process model is an instance of a process ontology. Process ontologies which are developed in the SUPER project (SUPER, Hepp et al. 2007) include the Business Process Modeling Ontology (BPMO); the semantic Business Process Modeling Notation ontology (sBPMN), which is an ontological version of Business Process Modeling Notation (BPMN); the semantic Event Process Chain ontology (sEPC), which is an ontological version of Event Process Chain (EPC) (Keller 1992); the semantic Business Process Execution Language ontology (sBPEL), which is a ontological version of Business Process Execution Language (BPEL) (Andrews 2003). These ontologies are described using the ontology-formalism Web Service Modeling Language (WSML) (de Bruijn et al. 2005). There are 5 variants of WSML: WSML-Core, WSML-DL, WSML-Flight, WSML-Rule, and WSML-Full, differing in logical expressiveness and underlying language paradigm. The ontologies considered in this paper are formalized using WSML-Flight, which is a compromise between the allowed expressiveness and the reasoning capability of the ontology language. In the following, we assume thus that a process model is an instance of a process ontology, which is specified in WSML-Flight.

For each option we take into account the expressiveness of the query language, the scalability of the query processing and the effort for the integration of the query processing with the underlying data storage. Scalability is a rather fuzzy term. In general, one would understand that in the context of reasoning. Reasoning is used to infer conclusions that are not explicitly stated but are required by or consistent with a known set of data (cf. (Passin 2004)). A system or a framework is scalable if enlarging the data-set, which is in our context the set of actual process models that described using ontologies, leads to a performance loss that is tolerable. More formal, one could say that reasoning is scalable if augmenting the input size of the problem, which in this case refers to the ontologies plus the instance data of the ontologies, leads at most to a polynomial increase of the time in which reasoning can be performed. With regards to the reasoning capability we consider two options, namely the storage mechanism with or without reasoning capability.

4.2.1 Option 1: Without Reasoning Capability

For storage mechanisms without reasoning capability we considered Relational Database Management System (RDBMS) and RDF store, which have been widely adopted at the time of writing this paper.

Queries against RDBMS are normally formalized using the Structured Query Language (SQL). SQL is quite powerful and bases on both the relational algebra and the tuple relational calculus (Siberschatz 2006). However, it has still some limitations. For example, a simple query such as:

Find all supervisors of the employee John Smith

requires computation of transitive closures on the personnel hierarchies. It is known that transitive closure can not be expressed using relational algebra (Libkin 2001,Abiteboul 1995). In SQL one can express transitive closures using *WITH RECURSIVE* to create recursive views, which could be very expensive. Furthermore the “supervisor” relationship must be stored explicitly in the database system. Because SQL can express queries aim at the explicitly stored data, it has no capability to take into account of the implicit data, which can

be derived from the instances of the ontologies based on the axioms specified there. This is not sufficient for the requirements on query processing of the SBPR.

(de Bruijn 2006) defined a RDF representation of WSML, which allows storing WSML data in a RDF store. RDF (RDF 2004) store is a framework providing support for the RDF Schema (RDFS 2004) inference and querying, which uses a relational database system as the underlying storage for the RDF data. In this section we only consider RDF stores without third-party inference engine or reasoner integrated. The inference here refers to the RDFS entailments supported by the RDFS semantics. There are already several reference implementations of RDF stores like Sesame[9]. The inference in such RDF stores is normally based on the RDF schema, which provides only restricted number of constructs to describe the relationships between the resources, as well as these between the properties, such as rdfs:subClassOf, rdfs:subPropertyOf. The query processing of RDF stores is based on special query languages for RDF data like Simple Protocol and RDF Query Language (SPARQL) or Sesame RDF Query Language (SeRQL). Using these query languages one cannot express transitivity or transitive closure. Furthermore, these query languages take only into account explicitly stored data. The implicit data can be derived by the inference capability. However, the inference capability is very limited in RDF stores.

4.2.2 Option 2: With Reasoning Capability

Jena 2 (JENA) is another RDF store, which support not only native entailment of RDFS semantics but also third-party inference engines or reasoner. The primary use of plug-in such inference engine or reasoner is to support the use of languages such as RDFS and OWL which allow additional facts to be inferred from instance data and class descriptions. The default OWL reasoner in Jena can only perform reasoning on a subset of OWL semantics. To provide complete support of OWL DL reasoning one can use external OWL DL reasoner such as Pellet[10], Racer[11] or FaCT[12]. Jena can handle OWL DL, but there is only a partial bi-directional mapping defined between WSML-Core and OWL DL, which is not sufficient to fulfill the requirements of SBPR.

Besides Jena OWLIM (OWLIM 2006) is another implementation, which enables RDF store with reasoning capability. OWLIM is a high performance Storage and Inference Layer (SAIL) for Sesame, which performs OWL Description Logic Programs (DLP) (Grosof 2003) reasoning, based on forward-chaining of entailment rules (Kiryakov 2005). As argued in (Kiryakov 2005), OWLIM can query the Knowledge Base (KB) of 10 million statements with an upload and storage speed of about 3000 statements per second. In more detail [OWLIM], querying is done by materializing the KB, i.e., for every update to the KB, the inference closure of the program is computed: all conclusions that can be recursively obtained by applying Process Ontology rules, given certain instance data (process models), are computed. This approach has the advantage that querying or other reasoning tasks are performed fast because the reasoning was done beforehand. Moreover, one could store the inference closure in the persistent storage, effectively using optimization methods for storage. The approach taken in OWLIM shows that taking into account ontologies does not need to lead to a significant performance loss per se. Nonetheless, the approach has some disadvantages.

OWLIM provides support for a fraction of OWL, close to OWL DLP and OWL-Horst (ter Horst 2005), which can be mapped to WSML and vice versa. However, the expressiveness of OWL DLP corresponds to WSML-Core. OWL-Horst is more powerful than WSML-Core, but it is still not as powerful as WSML-Flight. Therefore, the expressiveness is not adequate. As we already discussed, the reasoning in OWLIM takes the forward-chaining approach. Forward-chaining means that the reasoner starts from the facts that are already known and

[9] http://www.openrdf.org/index.jsp
[10] http://pellet.owldl.com/
[11] http://www.racer-systems.com/
[12] http://www.cs.man.ac.uk/~horrocks/FaCT/

infers new knowledge in an inductive fashion. The result of forward-chaining can be stored for reuse. This enables efficient query answering, because all facts needed for the query processing are already available in the data storage. But in the meanwhile this introduces also the expensive time and space consuming operations of data manipulation such as update or delete. Newly added or updated data leads to computing the inference closure in the SBPR again. Removal of process models is even more problematic, as facts from the inference closure that were introduced by this removed process models have also to be removed from the SBPR, which could lead to more removal operations. In the worst case this could lead to a recalculation of a large part of the inference closure. However, the removal of process models from the SBPR seems to be an action that is less common. The OWLIM approach also relies heavily on the fact that the semantics of OWL DLP and extensions towards OWL Lite are monotonic. The monotonic semantics allows for incremental additions to the Process Library, i.e. one can extend the current inference closure with new inferences. In the presence of non-monotonism, e.g., negation as failure as for example in WSML-Flight (de Bruijn 2006), such an incremental approach no longer works, as adding knowledge may prohibit previously made deductions.

IRIS (Integrated Rule Inference System)[13] is an inference engine, which together with the WSML2Reasoner framework [14], supports query answering for WSML-Core and WSML-Flight. In essence, it is a Datalog engine extended with stratified negation[15]. The system implements different deductive database algorithms and evaluation techniques. IRIS allows different data types to be used in semantic descriptions according the XML Schema specification and offers a number of built-in predicates. Functionality for constructing complex data types using primitive ones is also provided.

The translation from a WSML ontology description to Datalog is conducted using the WSML2Reasoner component. This framework combines various validation, normalization and transformation functionalities which are essential to the translation of WSML ontology descriptions to set of predicates and rules. Further on, rules are translated to expressions of relational algebra and computed using the set of operations of relational algebra (i.e., union, set difference, selection, Cartesian product, projection etc.). The motivation for this translation lies in the fact that the relational model is the underlying mathematical model of data for Datalog and there are a number of database optimization techniques applicable for the relational model. Finally optimized relational expressions serve as an input for computing the meaning of recursive Datalog programs.

The core of the IRIS architecture, see Figure 5, is defined as a layered approach consisting of:

- Knowledgebase API;
- Invocation API;
- Storage API.

The knowledgebase API is a top API layer encapsulating central abstractions of the underlying system (e.g., rule, query, atom, tuple, fact, program, knowledge base, context etc.). The purpose of this layer is to define the basic concepts of data model used in IRIS as well as to define the functionality for the knowledge base and program manipulation.

The invocation API characterizes a particular evaluation strategy (e.g., bottom-up, top-down or mixture of these two strategies) and evaluation methods for a given strategy which are used with respect to a particular logic program.

IRIS implements the following evaluation methods[16]:

- Naive evaluation;

[13] http://sourceforge.net/projects/iris-reasoner/

[14] WSML2Reasoner framework: http://tools.deri.org/wsml2reasoner/

[15] IRIS is continuously being developed and the support for non-stratified negation and unsafe rules is envisioned in coming releases.

[16] More evaluation techniques are under development.

- Semi-naive evaluation;
- Query-subquery (QSQ) evaluation.

The storage layer defines the basic API for accessing data and relation indexing. A central abstraction in this layer is a relation which contains a set of tuples and serves as an argument in each operation of relation algebra. The implementation of IRIS relation is based on Collection and SortedSet Java interfaces where red-black binary search trees are utilized for indexing.

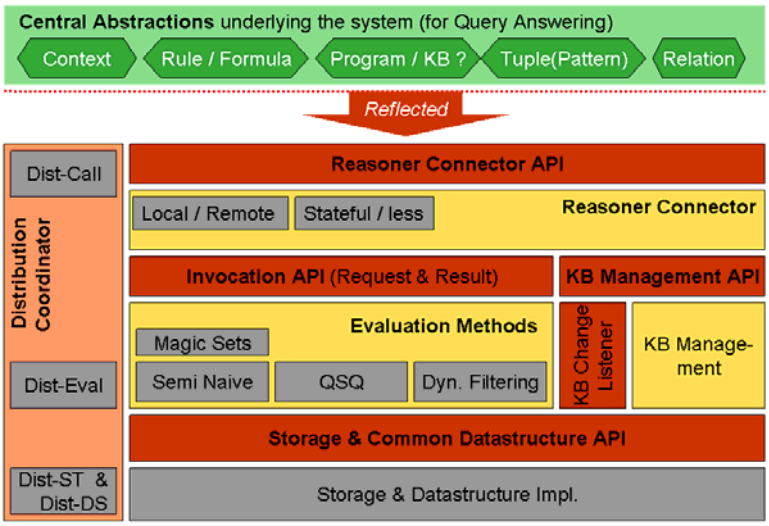

Figure 5: IRIS Architecture

Current inference systems exploit reasoner methods developed rather for small knowledge bases. Such systems either process data in main memory or use a Relational Database Management System (RDBMS) to efficiently access and do relational operations on disk persistent relations. Main memory reasoners cannot handle datasets larger than their memory. On the other side, systems based on RDBMSs feature great performance improvement comparing with main memory systems, but efficient database techniques (e.g., cost-based query planning, caching, buffering) they utilize are suited only for EDB relations and not fully deployable on derived relations.

IRIS is designed to meet requirements for large scale reasoning. Apart from the state-of-the-art deductive methods, the system utilizes database techniques and extends them for implicit knowledge in order to effectively process large datasets. We are building an integrated query optimizer. The estimation of the size and evaluation cost of the intentional predicates will be based on the adaptive sampling method (Liption 1990, Ruckhaus 2006), while the extensional data will be estimated using a graph-based synopses of data sets similarly as (Spiegel 2006). Further on, for large scale reasoning (i.e., during the derivation of large relations which exceeds main memory), run time memory overflow may occur.

Therefore in IRIS we are developing novel techniques for a selective pushing of currently processed tuples to disk. Such techniques aim to temporarily lessen the burden of main memory, and hence to make the entire system capable of handling large relations.

The comparison shows that a RDBMS with integrated IRIS inference engine is the only suitable solution to fulfill the requirements of the SBPR.

4.3 Overall Architecture

In this section we present the overall architecture of the SBPR. The BPL has been designed in a layered architecture style consisting of

Semantic Business Process Repository API
Service Layer
Persistence Layer

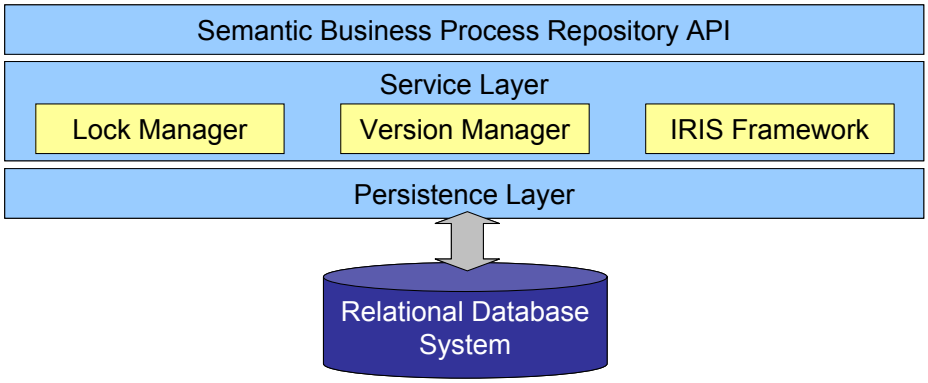

Figure 6: SBPR Architecture

Semantic Business Process Repository API

The Semantic Business Process Repository API provides the programmatic access to the SBPR. It includes the API designed after the CRUD pattern, which represents the four basic functions of persistent storage, namely create, retrieve, update and delete. Besides the CRUD API the SBPR API also provides check-in and check-out functions for long-running process modeling. The query API rounds off the SBPR API by providing programmatic access to the IRIS Framework for query answering.

Service Layer

The Service Layer implements the SBPR API and processing logic of the SBPR. The Service Layer contains three modules: Lock Manger, Version Manager and the IRIS Framework. The Lock Manager take charge of requests on locking and unlocking for the process models in the SBPR. A locking request can only be granted when the process model is yet not locked. The Version Manager takes care of the management of the versions of process models. To record the modeling history every new process model or changed process model is stored as a new version in the SBPR. IRIS Framework takes the responsibility for the query processing in SBPR.

Persistence Layer

The Persistence Layer manages the data access to the underlying relational database system and provides an abstraction for data access operations. It provides persistent solutions for persistent objects by adopting Object Relational Mapping (ORM) middleware such as Hibernate and Data Access Object (DAO) pattern.

5. CONCLUSIONS AND DIRECTIONS FOR FURTHER RESEARCH

We gave an overall introduction to some well-known ontology repositories, including native stores and database based stores, and highlights strengths and limitations of each store. It is reported in (Ma et al., 2006) that Minerva achieves good performance in benchmarking tests. We took Minerva as an example to analyze ontology storage in databases in depth, as well as to discussed efficient indexes for scaling up ontology repositories. We then discussed a scalable reasoning method for handling expressive ontologies, as well as summarized other similar approaches.

We have presented a framework for reasoning with Description Logic based WSML. It builds on top of a transformation from WSML-DL to OWL-DL and supports all main DL specific reasoning tasks. We thus linked the work for storing OWL ontologies, to the work on WSML-DL, providing the reader with an insight in storing and reasoning with both OWL-DL and WSML-DL ontologies.

As a practical use case of storing ontologies and reasoning with them, we presented Semantic Business Process Repository (SBPR) for systemically management of semantic business process models. We first analyzed the main requirements on SBPR. After the comparison of different approaches for storage mechanisms we concluded that a RDBMS with IRIS inference engine integrated is, due to the expressiveness of the query language and the reasoning capability, the most suitable solution.

Currently IRIS is a WSML-Flight reasoner. The system is extensively being developed to support reasoning with WSML-Rule (i.e., support for function symbols, unsafe rules and non-stratified negation). Further on, IRIS will tightly integrate a permanent storage system designed for distributed scalable reasoning. One of our major objectives is the implementation of Rule Interchange Format (RIF)[17] in IRIS. Implementing RIF, IRIS will be capable of handling rules from diverse rule systems and will make WSML rule sets interchangeable with rule sets written in other languages that are also supported by RIF.

Finally, IRIS will implement novel techniques for reasoning with integrating frameworks based on classical first-order logic and nonmonotonic logic programming as well as techniques for Description Logics reasoning.

6. REFERENCES

AllegroGraph, http://www.franz.com/products/allegrograph/index.lhtml, 2006

SnoMed Ontology, http://www.snomed.org/snomedct/index.html, 2006

IODT, IBM's Integrate Ontology Development Toolkit, http://www.alphaworks.ibm.com/tech/semanticstk, 2005

Abiteboul, Serge; Hull, Richard; Vianu, Victor: Foundations of Databases. Addison-Wesley, 1995

Agrawal, R., Somani, A., and Xu, Y., 2001, Storage and Querying of E-Commerce Data. In Proceedings of the 27th International Conference on Very Large DataBases, pages 149–158, Morgan Kaufmann.

Andrews, Tony; Curbera, Francisco; Dholakia, Hitesh; et al.: Business Process Execution Language for Web Services Version 1.1. 5 May 2003

[17] Rule Interchange Format-W3C Working Group: http://www.w3.org/2005/rules/

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D. and Patel-Schneider, P. F., 2003, *The Description Logic Handbook*. Cambridge University Press.

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A., 2004, Owl web ontology language reference. Technical report. Available from: http://www.w3.org/TR/owl-ref/.

Bechhofer, S., Volz R. and Lord P.W., 2003, Cooking the Semantic Web with the OWL API, in: *International Semantic Web Conference*, pp. 659-675.

Bernstein, Philip A.; Dayal, Umeshwar: An Overview of Repository Technology. In VLDB 1994.

Bhattacharjee, B., Padmanabhan, S., and Malkemus, T., 2003, Efficient Query Processing for Multi-Dimensionally Clustered Tables in DB2, In Proceedings of the 29th Conference on Very Large Data Bases, pages 963–974, Morgan Kaufmann.

Borgida, A., 1996, On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence* 82(1-2):353–367. Available from: http://citeseer.ist.psu.edu/borgida96relative.html.

BPMN, Business Process Modeling Notation Specification. OMG Final Adopted Specification, February 6, 2006

Brickley, D. and Guha, R. V., 2004, Rdf vocabulary description language 1.0: Rdf schema. Technical report. Available from: http://www.w3.org/TR/rdf-schema/.

Broekstra, J., Kampman, A., and Harmelen, van F., 2002, Sesame: A generic architecture for storing and querying RDF and RDF schema. In Proceedings of the 1st International Semantic Web Conference, volume 2342 of Lecture Notes in Computer Science, pages 54–68, Springer.

Bruijn, Jos de; Kopecký, Jacek; Krummenacher, Reto: RDF Representation of WSML. 20 December 2006

Bruijn, de J., Lausen, H., Krummenacher, R., Polleres, A., Predoiu, L., Kifer, M., and Fensel, D., 2005, The web service modeling language WSML. WSML Final Draft D16.1v0.21, WSML. Available from: http://www.wsmo.org/TR/d16/d16.1/v0.21/.

Bruijn, de J., Polleres, A., Lara, R., and Fensel, D., 2005, OWL DL vs. OWL Flight: Conceptual Modeling and Reasoning on the Semantic Web. In Proceedings of the 14th International Conference on the World Wide Web.

Brunner, J., Ma, L., Wang, C., Zhang, L., Wolfson, D. C., Pan, Y., and Srinivas, K., 2007, Explorations in the Use of Semantic Web Technologies for Product Information Management. In Proceedings of the 16th International Conference on the World Wide Web. To appear.

Calvanese, D., Giacomo, De G., Lembo, D., Lenzerini, M., and Rosati, R., 2005, DL-Lite: Tractable Description Logics for Ontologies. In Proceedings of the 12th National Conference on Artificial Intelligence, pages 602-607.

Calvanese, D., Giacomo, De G., Lembo, D., Lenzerini, M., and Rosati, R., 2006, Data Complexity of Query Answering in Description Logics. In Proceedings of the 10th International Conference on the Principles of Knowledge Representation and Reasoning, pages 260-270, AAAI Press.

Chen, Y., Ou, J., Jiang, Y., and Meng, X., 2006, HStar-a Semantic Repository for Large Scale OWL Documents. In Proceedings of the 1st Asian Semantic Web Conference, volume 4185 of Lecture Notes in Computer Science, pages 415-428, Springer.

Das, S., Chong, E.I., Eadon, G., and Srinivasan, J., 2004, Supporting Ontology-Based Semantic matching in RDBMS. In Proceedings of the 30th International Conference on Very Large Data Bases, pages 1054-1065.

Dolby, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Ma, L., Schonberg, E., and Srinivas, K., 2007, Scalable semantic retrieval through summarization and refinement. IBM Technical report, 2007.

Donini, M. F., Nardi, D., and Rosati, R., 2002, Description Logics of Minimal Knowledge and Negation as Failure. ACM Transactions on Computational Logic, 3(2):177–225.

Fitting, M., 1996, *First-Order Logic and Automated Theorem Proving*. 2nd ed., Springer-Verlag, New York.

Fokoue, A., Kershenbaum, A., Ma, L., Schonberg, E., and Srinivas, K., 2006b, The summary abox: Cutting ontologies down to size. In Proceedings of the 5th International Semantic Web Conference, volume 4273 of Lecture Notes in Computer Science, pages 343–356, Springer.

Garcia-Molina, H., Ullman, J., and Widom, J., 2000, Database System Implementation. Prentice-Hall.

Grosof, B., Horrocks, I., Volz, R., and Decker, S., 2003, Description logic programs: combining logic programs with description logic. In Proceddings of the 12th International Conference on the World Wide Web, pages 48-57.

Guo, Y., and Heflin, J., 2006, A Scalable Approach for Partitioning OWL Knowledge Bases. In Proceedings of the 2nd International Workshop on Scalable Semantic Web Knowledge Base Systems.

Haarslev, V., and Moller, R., 2001, RACER System Description. In Proceedings of Automated Reasoning, the 1st International Joint Conference.

Hepp, Martin; Leymann, Frank; Domingue, John; Wahler, Alexander; Fensel, Dieter: Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. Proceedings of the IEEE ICEBE 2005, October 18-20, Beijing, China, pp. 535-540.

Hepp, Martin; Roman, Dumitru: An Ontology Framework for Semantic Business Process Management, Proceedings of Wirtschaftsinformatik 2007, February 28 - March 2, 2007, Karlsruhe (forthcoming).

Horrocks I., Patel-Schneider P.F., van Harmelen F., 2003, From SHIQ and RDF to OWL: The making of a Web Ontology Language, *J. of Web Semantics*, 1570-8268, pp. 7-26, Available from: http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/HoPH03a.pdf

Horrocks, I., and Tessaris, S., 2002, Querying the semantic web: a formal approach. In Proceedings of the 1st International Semantic Web Conference, volume 2342 of Lecture Notes in Computer Science, pages 177–191, Springer.

Hustadt, U., Motik, B., and Sattler, U., 2004, Reducing SHIQ Descrption Logic to Disjunctive Datalog Programs. In Proceedings of the 9th International Conference on Knowledge Representation and Reasoning, pages 152-162.

Hustadt, U., Motik, B., and Sattler, U., 2005, Data Complexity of Reasoning in Very Expressive Description Logics. In Proceedings of the 19th International Joint Conference on Artificial Intelligence, pages 466-471.

JENA, http://jena.sourceforge.net/index.html

Keller, G.; Nüttgens, M.; Scheer, A.-W.: Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)", in: Scheer, A.-W. (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89, Saarbrücken 1992.

Kiryakov, A., Ognyanov, D., and Manov, D, 2005, OWLIM - a pragmatic semantic repository for OWL. In Proceedings of the 2005 International Workshop on Scalable Semantic Web Knowledge Base Systems.

Kiryakov, Atanas; Ognyanov, Damyan; Manov, Dimitar: OWLIM – a Pragmatic Semantic Repository for OWL. In Proc. of Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2005), WISE 2005, 20 Nov, New York City, USA.

Krotzsch, M.., Rudolph, S., and Hitzler, P., 2006, On the complexity of Horn description logics. In Proceedings of the 2nd Workshop OWL Experiences and Directions.

Libkin, Leonid: Expressive Power of SQL. The 8th International Conference on Database Theory. London, United Kingdom, 2001

Lipton, Richard and Naughton, Jeffrey. Query size estimation by adaptive sampling (extended abstract). In PODS '90: Proceedings of the ninth ACM SIGACTSIGMOD-SIGART symposium on Principles of database systems, pages 40–46, New York, NY, USA, 1990. ACM Press.

Lloyd, J. W., 1987, *Foundations of Logic Programming*. 2nd ed., Springer-Verlag, New York.

Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., and Liu. S., 2006, Towards a complete owl ontology benchmark. In Proceedings of the 3rd Europe Semantic Web Conference, volume 4011 of Lecture Notes in Computer Science, pages 125–139, Springer.

Matias, Y., Vitter, J. S., and Wang, M., 1998, Wavelet-based histograms for selectivity estimation. In Proceedings of the ACM SIGMOD International Conference on Management of Data.

Mei, J., Ma, L., and Pan, Y., 2006, Ontology Query Answering on Databases. In Proceedings of the 5th International Semantic Web Conference, volume 4273 of Lecture Notes in Computer Science, pages 445-458, Springer.

Motik, B., Sattler, U., and Studer, R., 2004, Query Answering for OWL-DL with Rules. In Proceedings of the 3th International Semantic Web Conference, volume 3298 of Lecture Notes in Computer Science, pages 549-563, Springer.

Motik, B., Horrocks, I., and Sattler, U., 2006, Integrating Description Logics and Relational Databases. Technical Report, University of Manchester, UK.

Motik, B., and Rosati, R., 2007, A Faithful Integration of Description Logics with Logic Programming. In Proceedings of the 20th International Joint Conference on Artificial Intelligence.

Murray C., Alexander N., Das S., Eadon G., Ravada S., 2005, Oracle Spatial Resource Description Framework (RDF), 10g Release 2 (10.2).

OWLIM – OWL semantics repository. 2006. http://www.ontotext.com/owlim/

Pan, Z., and Heflin, J., 2003, DLDB: Extending relational databases to support semantic web queries. In Proceddings of Workshop on Practical and Scaleable Semantic Web Systems.

Passin, Thomas B.: Explorer's Guide to the Semantic Web. Manning, 2004.

Prud'hommeaux, E., Seaborne, A., eds., 2005, SPARQL Query Language for RDF.W3C Working Draft.

Poosala, V., Ioannidis, Y. E., Haas, P. J., and Shekita, E., 1996, Improved histograms for selectivity estimation of range predicates. In Proceedings of the ACM SIGMOD International Conference on Management of Data.

RDF Primer, W3C Recommendation 10 February 2004. http://www.w3.org/TR/rdf-primer

RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation 10 February 2004

Rector, A., 2003, Message to public-webont-comments@w3.org: "case for reinstatement of qualified cardinality restrictions". Available from: http://lists.w3.org/Archives/Public/public-webontcomments/2003Apr/0040.html.

Reiter, R., 1992, What Should a Database Know? Journal of Logic Programming, 14(1–2):127–153.

Roman, D., Lausen, H., and Keller, U., 2004, Web service modeling ontology (WSMO). WSMO final draft d2v1.2. Available from: http://www.wsmo.org/TR/d2/v1.2/.

Rosati, R., 2006, DL + log: A Tight Integration of Description Logics and Disjunctive Datalog. In Proceedings of the 10th International Conference on the Principles of Knowledge Representation and Reasoning, pages 68–78, AAAI Press.

Ruckhaus, Edna and Ruiz, Eduardo. Query evaluation and optimization in the semantic web. In Proceedings of the ICLP'06 Workshop on Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALPSWS2006), Washington, USA, August 16 2006.

Siberschatz, Abraham; Korth, Henry F.; Sudarshan, S.: Database System Concepts. Fifth Edition, McGraw-Hill, 2006.

Sirin, E., and Parsia, B., 2004, Pellet: An OWL DL Reasoner. In Proceedings of Workshop on Description Logic.

Smith, Howard; Fingar, Peter: Business Process Management. The Third Wave. Meghan-Kiffer,US 2003.

Spiegel, J. and Polyzotis, N. Graph-based synopses for relational selectivity estimation. In SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, pages 205–216, New York, NY, USA, 2006. ACM Press.

Steinmetz, N., 2006, WSML-DL Reasoner. Bachelor thesis, Leopold-Franzens University Innsbruck. Available from: http://www.deri.at/fileadmin/documents/thesis/dlreasoner.pdf

SUPER, The European Integrated Project – Semantics Utilised for Process Management within and between Enterprises. http://www.ip-super.org/

ter Horst, Herman J.: Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity. In Proc. of ISWC 2005, Galway, Ireland, November 6-10, 2005. LNCS 3729, pp. 668-684.

Volz, R., 2004, Web Ontology Reasoning with Logic Databases. PhD thesis, Fridericiana University Karlsruhe.

Wang, M., Chang, Y., and Padmanabhan, S., 2002, Supporting Efficient Parametric Search of E-Commerce Data: A Loosely-Coupled Solution. In Proceedings of the 8th International Conference on Extending Database Technology, pages 409-426.

Wilkinson, K., Sayers, C., Kuno, H. A., and Reynolds, D., 2003, Efficient RDF storage and retrieval in Jena2. In Proceedings of VLDB Workshop on Semantic Web and Databases, pages 131-150.

Wu, XD, Lee, ML, Hsu, W., 2004, A prime number labeling scheme for dynamic ordered XML trees. In Proceedings of the 20th Int'l Conf. on Database Engineering (ICDE). pages 66-78, IEEE Computer Society.

Zhou, J., Ma, L., Liu, Q., Zhang, L., Yu, Y., and Pan, Y., 2006, Minerva: A Scalable OWL Ontology Storage and Inference System. In Proceedings of the 1st Asian Semantic Web Conference, volume 4185 of Lecture Notes in Computer Science, pages 429-443, Springer.