OWL SAIQL —
An OWL DL Query Language for Ontology Extraction

Alexander Kubias[1], Simon Schenk[1], Steffen Staab[1], and Jeff Z. Pan[2]

[1] University of Koblenz-Landau, 56070 Koblenz, Germany,
(kubias|sschenk|staab)@uni-koblenz.de,
[2] The University of Aberdeen, Aberdeen AB24 3UE, UK,
jpan@csd.abdn.ac.uk

Abstract. Existing approaches for querying OWL DL do either only operate on syntactic constructs without taking into account the semantics of OWL or do only have a restricted access to the T-Box. We present SAIQL, the novel Schema And Instance Query Language for OWL DL, that is well suited for ontology extraction. We describe its syntax and explain a basic evaluation strategy. We illustrate the use of SAIQL with an example for ontology extraction and re-use.

Key words: OWL, Query Language, Schema, Ontology Extraction

1 Introduction

With the standardization of the Web Ontology Language OWL [1], the use and re-use of ontological knowledge has gained significant momentum. For using web ontologies it is crucial to be able to access web ontologies in an intuitive and versatile manner. In contrast to RDF, where SPARQL [2] is providing access to RDF data *and* RDF schema information, a corresponding query language is missing for OWL. Existing OWL querying approaches, e.g. OWL-QL [3], have only a restricted access to the T-Box so that only named classes and individuals can be retrieved. Using SPARQL [2] for querying OWL allows the user to query OWL A-Box and T-Box, but it is not aware of OWL semantics and it is very cumbersome, because of its triple semantics. In fact, we will give some examples of querying desiderata that cannot easily be fulfilled by SPARQL or OWL-QL.

The requirements for querying OWL naturally include conjunctive queries of the OWL A-Box [4], but as has also been recently argued for other ontology languages with explicitly queryable schema representations (cf. [5]), the querying of schema as well as instance information constitutes an important feature of the querying language.

We illustrate our requirements for querying OWL with an application from ontology extraction (cf., [6]) for re-using parts of an ontology. While implementations of ontology extraction algorithms are currently dominated by imperative style programming, re-using parts of an ontology would be greatly facilitated by a query language that would allow querying for schema and instance information. For instance, it is useful to ask for all individuals of classes, which are defined using a restriction with a certain property or having a specific subclass.

As one main contribution, our query language can handle class descriptions in addition to class names and individuals. The extraction of these class descriptions is of major importance as they provide the definition for a certain class name. Instead of extracting isolated class names and individuals like in OWL-QL, the class names, class descriptions and individuals are returned as OWL DL axioms. Thus, the result is a fully working OWL DL ontology.

In the following, we present our small example use case in ontology extraction and derive some requirements from it (section 2). We then discuss some of the foundations on which we build our approach in section 4. In section 5, we present the original querying language SAIQL (Schema And Instance Query Language) that is able to combine T-Box and A-Box querying in an integrated manner. Finally, we discuss some related work, before we conclude.

2 Use Case and Requirements

In our running example, we assume a large ontology, `MotorOntology`, parts of which we want to extract and re-use for a new information system about cars in order to save costs and ensure high quality (cf., [7] on the benefits of ontology re-use). Naturally, we do not want to adopt the given ontology one by one, as we are only interested in schema and instance information related to cars.

For extracting our target ontology from `MotorOntology` (see an excerpt in Figure 1), we are interested in all axioms that contain class names, which are subclasses of the cardinality restriction with the value 4 for the property `hasWheel`, and their descriptions. Additionally, we are interested in all axioms about individuals of these classes. Given the running example in Figure 1, the class names `Car`, `Convertible` and `Van` and their descriptions should be delivered as class axioms. Furthermore, the individual axioms about `c` and `v` need to be extracted.

```
EquivalentClasses(MotorBike restriction(hasWheel cardinality(2)))
EquivalentClasses(Car restriction(hasWheel cardinality(4)))

Class(Convertible partial Car
   restriction(hasConvertibleTop someValuesFrom(owl:Thing)))
Class(Van partial restriction(hasWheel cardinality(4))
   restriction(hasSlidingDoor someValuesFrom(owl:Thing)))
DisjointClasses(Convertible Van);

Individual(c type(Convertible))
Individual(v type(Van))
Individual(m type(MotorBike))
```

Fig. 1. Example `MotorOntology` given in OWL DL (in OWL abstract syntax)

From the presented use case we derive the following requirements: First, the query language should regard and use the semantics of OWL DL ontologies. By exploiting their semantics, additional knowledge can be inferred that is not explicitly stated in the ontology. Thus, in our running example the class `Van` could be returned as a subclass of `Car` without being explicitly mentioned as its subclass.

As a further requirement, the query language should not only retrieve class names and individuals, but also class descriptions. As shown in our use case, it is not sufficient to return the class `Convertible` without knowing its definition. We also want to retrieve the class description of the class `Convertible`, namely that it is a subclass of the class `Car` and that it has an existential restriction for the property `hasConvertibleTop`.

Instead of extracting isolated class names and individuals, we want to return the class names, class descriptions and individuals as OWL DL axioms so that the result is a fully working OWL DL ontology. Thus, it should be possible to use the result of a SAIQL query as an input for another SAIQL query.

As models of OWL ontologies are infinite in general, query answering might cause infinite results. In order to ensure finite answers to queries, we need to define privileged sets of class names, class descriptions and individuals that are used in query results. These sets should be determined by the concrete syntactic notation of the queried ontology, which is always finite.

Finally, we want to state join-like conditions on selected classes and individuals by using identical names for variables. Thus, it should be possible to select all classes $?X$ so that an individual $?i$ belongs to this class $?X$ and so that the same class $?X$ must be a subclass of another class $?Y$.

3 SAIQL in a Nutshell

In our running example, we want to extract the axioms for our target ontology from `MotorOntology`. As we are interested in all axioms about class names, which are subclasses of the cardinality restriction with value 4 for the property `hasWheel`, their descriptions and their individuals, the SAIQL query in figure 2 is formulated. Within this query, three variables `?i`, `?X` and `?Z` are used. The variable `?i` is treated as a placeholder for an individual, whereas the variable `?X` is a placeholder for a class name and `?Z` for a class description. Furthermore, the SAIQL query consists of four clauses: The `CONSTRUCT` clause determines the format of the extracted axioms, the `FROM` clause determines from which ontology axioms are extracted, the `LET` clause associates variables with value ranges and the `WHERE` clause constitutes the conditions under which axioms are extracted.

```
CONSTRUCT Class(?X partial ?Z); Individual(?i type(?X))
FROM MotorOntology
LET IndividualName ?i; ClassName ?X; ClassDescription ?Z
WHERE Class(?X partial restriction(hasWheel cardinality(4)))
      AND Individual(?i type(?X)) AND Class(?X partial ?Z)
```

Fig. 2. SAIQL query for our example

4 Abstract Syntax and Semantics for OWL DL

SAIQL is a query language for OWL DL. In this section, we repeat some of the foundations of OWL DL that we need for defining SAIQL.

4.1 Abstract Syntax for OWL DL

In the following, we present the abstract syntax for OWL DL by means of an extended BNF. The syntax is adopted from [8]. For the sake of simplicity and consistency, the syntax is slightly simplified leaving out annotation properties and import commands. Additionally, some terms are renamed. For example, the term `fact` is called `individualAxiom` in order to be consistent with the rest of the paper. Furthermore, we omit datatypes and datatype properties. Based on this syntax, we will define the syntax of SAIQL. Below you find an excerpt of the OWL abstract syntax in EBNF:

```
ontology ::= 'Ontology(' [ ontologyID ] { axiom } ')'
axiom ::= classAxiom | propertyAxiom | individualAxiom
...

classAxiom ::= 'Class(' classID modality { description } ')'
   | 'EnumeratedClass(' classID { individualID } ')'
   | 'DisjointClasses(' description description { description } ')'
   | 'EquivalentClasses(' description { description } ')'
   | 'SubClassOf(' description description ')'

description ::= classID | restriction | 'unionOf(' { description } ')'
   | 'intersectionOf(' { description } ')' | 'complementOf(' description ')'
   | 'oneOf(' { individualID } ')'
...

individualAxiom ::= individual
   | 'SameIndividual(' individualID individualID {individualID} ')'
   | 'DifferentIndividuals(' individualID individualID {individualID} ')'
...
```

4.2 Excerpt of OWL DL semantics

In the following, we describe parts of the model-theoretic semantics for OWL DL. The presented semantics is taken from [9]. We have slightly modified its presentation given here in order to use it more easily for the definition of SAIQL in the remainder of the paper.

Definition 1. *Let N_C, N_{IP}, N_{DP}, N_I be the sets of URI references that can be used to denote classes, individual-valued properties, data-valued properties and individuals. An OWL DL interpretation is a tuple $I = (\Delta^I, \Delta^D, .^I, .^D)$ where*

- *the individual domain Δ^I is a nonempty set of individuals,*
- *the datatype domain Δ^D is a nonempty set of data values,*
- *$.^I$ is a individual interpretation function, and*
- *$.^D$ is a datatype interpretation function.*

Definition 2. *An individual interpretation function $.^I$ is a function that maps*

- *each individual name $a \in N_I$ to an element $a^I \in \Delta^I$,*
- *each class name $C \in N_C$ to a subset $C^I \subseteq \Delta^I$,*
- *each individual-valued property name $R \in N_{IP}$ to a binary relation $R^I \subseteq \Delta^I \times \Delta^I$, and*
- *each data-valued property name $T \in N_{DP}$ to a binary relation $T^I \subseteq \Delta^I \times \Delta^D$.*

Definition 3. *A* class *is a group of individuals. If a class is only defined by naming it, we call it an* atomic class*. A* class description *is a declaration of a class using its name (for atomic classes) or OWL class constructors (for non-atomic classes). Thus, each atomic class is also a class description.*

OWL class constructors are e.g. union, intersection or complement of classes, restrictions or enumerations. As mentioned in section 4.1, datatypes and datatype properties are not considered in detail in this paper. More details of the semantics of OWL DL can be found in [8].

As models of OWL ontologies are infinite in general, query answering could cause infinite answers. In oder to ensure finite answers to queries, we have defined the four finite sets N_C, N_{IP}, N_{DP} and N_I. Additionally, we must define the finite set of class descriptions used in the OWL DL ontology O.

Definition 4. *Given an OWL DL ontology O, the finite set of* class descriptions N_{CD} *consists of all elements that appear in O and that are produced by the* `description` *rule in the OWL DL EBNF.*

Example. Given the knowledge base in Figure 1, we have $N_C = \{$Car, Convertible, MotorBike, Van$\}$, $N_{IP} = \{$hasWheel, hasConvertibleTop, hasSlidingDoor$\}$, $N_{DP} = \emptyset$, $N_I = \{$c,v,m$\}$ and, finally, $N_{CD} = N_C \cup \{$ restriction(hasWheel cardinality(2)), restriction(hasWheel cardinality(4)), intersectionOf(Car restriction(hasConvertibleTop someValuesFrom(owl:Thing))), intersectionOf(restriction(hasWheel cardinality(4)) restriction(hasSlidingDoor someValuesFrom(owl:Thing))), restriction(hasConvertibleTop someValuesFrom(owl:Thing)), restriction(hasSlidingDoor someValuesFrom(owl:Thing)) $\}$.

Furthermore, we need to define what an axiom is. Axioms are the central elements of OWL DL ontologies and play an important role for SAIQL.

Definition 5. *Given an OWL DL ontology O, an* axiom *is a statement that is produced by the* `axiom` *rule in the OWL DL EBNF and that appears in O relating classes, properties or individuals. The whole set of axioms forms the ontology O.*

As proposed in [9], we distinguish class axioms, individual axioms and property axioms.

5 OWL SAIQL

In this section the syntax of SAIQL is described. Additionally, a basic evaluation strategy for SAIQL queries is proposed.

5.1 Syntax

As mentioned before, axioms are the central elements of OWL DL ontologies and they are also important for SAIQL queries. Their definition is extended by allowing variables in them.

Definition 6. *An* axiom pattern p *is defined analogously to an axiom, but allows variables at positions of class descriptions and individual names. The range of these variables can be either* N_C*,* N_I *or* N_{CD}*. The name of a variable must start with a "?".*

In contrast to [3], we do not support non-distinguished variables. As proposed for axioms without any variables, axiom patterns can be either class axiom patterns, individual axiom patterns or property axiom patterns.

Example. The SAIQL query in Figure 2 contains two individual axiom patterns and three class axiom patterns in the `CONSTRUCT` clause and in the `WHERE` clause. For instance, `Class(?X partial ?Z)` is a class axiom pattern and `Individual(?i type(?X))` is an individual axiom pattern.

The range of a variable is specified in the `LET` clause. In our running example in Figure 2, the variable `?X` is specified as a class name, the variable `?Z` is specified as a class description and the variable `?i` is specified as an individual name.

Definition 7. *An* OWL SAIQL *query has the form*

```
'CONSTRUCT' constructClause
'FROM' fromClause
'LET' letClause
'WHERE' whereClause
```

where `constructClause` *contains a sequence of axiom patterns,* `fromClause` *contains an URI reference of an ontology,* `letClause` *specifies the range of the variables and* `whereClause` *is a sequence of axiom patterns.*

In this paper, we restrict ourselves to a single ontology, from which axioms can be extracted. The complete syntax for SAIQL is as follows:

```
SAIQL-query ::= 'CONSTRUCT' constructClause 'FROM' fromClause
   'LET' letClause 'WHERE' whereClause

constructClause ::= axiomPattern {';' axiomPattern}
fromClause ::= ontologyID
letClause ::= variableBinding {';' variableBinding}
whereClause ::= axiomPattern {'AND' axiomPattern}

axiomPattern ::= classAxiomPattern | propertyAxiomPattern | individualAxiomPattern

classID ::= URIreference
individualID ::= URIreference
ontologyID ::= URIreference
individualvaluedPropertyID ::= URIreference

variableBinding ::= classNameBinding | individualNameBinding | classDescriptionBinding
classNameBinding ::= 'ClassName' classNameVar {',' classNameVar}
individualNameBinding ::= 'IndividualName' individualNameVar {',' individualNameVar}
classDescriptionBinding ::= 'ClassDescription' classDescriptionVar {',' classDescriptionVar}

lexicalForm ::= a unicode string in normal form C
classNameVar ::= '?'lexicalForm
individualNameVar ::= '?'lexicalForm
classDescriptionVar ::= '?'lexicalForm

className ::= classNameVar | classID
individualName ::= individualNameVar | individualID
classDescription ::= classDescriptionVar | description

classAxiomPattern ::= 'Class(' className modality { classDescription } ')'
   | 'EnumeratedClass(' className { individualName } ')'
   | 'DisjointClasses(' classDescription classDescription { classDescription } ')'
   | 'EquivalentClasses(' classDescription { classDescription } ')'
   | 'SubClassOf(' classDescription classDescription ')'
modality ::= 'complete' | 'partial'

description ::= className | restriction | 'unionOf(' { classDescription } ')'
   | 'intersectionOf(' { classDescription } ')'
   | 'complementOf(' classDescription ')' | 'oneOf(' { individualName } ')'

restriction ::= restriction( individualvaluedPropertyID individualRestrictionComponent
     { individualRestrictionComponent } )
individualRestrictionComponent ::= allValuesFrom( classDescription )
   | someValuesFrom( classDescription ) | value( individualName ) | cardinality

cardinality ::= 'minCardinality(' non-negative-integer ')'
   | 'maxCardinality(' non-negative-integer ')'
   | 'cardinality(' non-negative-integer ')'

propertyAxiomPattern ::= 'ObjectProperty(' individualvaluedPropertyID
     { 'super(' individualvaluedPropertyID ')' }
     [ 'inverseOf(' individualvaluedPropertyID ')' ] [ 'Symmetric' ]
     [ 'Functional' | 'InverseFunctional' | 'Functional' 'InverseFunctional' | 'Transitive' ]
     { 'domain(' classDescription ')' } { 'range(' classDescription ')' } ')'
   | 'EquivalentProperties(' individualvaluedPropertyID individualvaluedPropertyID
       { individualvaluedPropertyID } ')'
   | 'SubPropertyOf(' individualvaluedPropertyID  individualvaluedPropertyID ')'
```

```
individualAxiomPattern ::= individual
   | ’SameIndividual(’ individualName individualName {individualName} ’)’
   | ’DifferentIndividuals(’ individualName individualName {individualName} ’)’
individual ::= ’Individual(’ [ individualName ] { ’type(’ classDescription ’)’ } { value } ’)’
value ::= ’value(’ individualvaluedPropertyID individualName ’)’
   | ’value(’ individualvaluedPropertyID  individual ’)’
```

5.2 Canonical Process for the Query Evaluation

The query evaluation consists of three steps. In the first step the `LET` clause is evaluated: From the ontology O declared in the `FROM` clause we retrieve three sets of ontology elements, namely the finite set of class names N_C, the finite set of class descriptions N_{CD} and the finite set of individual names N_I. The finite set of class names N_C contains the names of all atomic classes and the names of all named complex classes. Additionally, the finite set of class descriptions N_{CD} contains all class descriptions that appear in the concrete syntactic notation of O (note that this includes all class names). Thus, $N_C \subseteq N_{CD}$. The range of every variable is bound to one of these sets:

Definition 8. *A* binding *is a substitution* $[?x_1/a_1]$ *of a variable* $?x_1$ *by a value* a_1 *from the range as defined in the* `LET` *clause, that is from* N_C*,* N_{CD} *or* N_I*. A* solution $s = [\boldsymbol{?x}/\boldsymbol{a}] = [?x_1/a_1][?x_2/a_2]\ldots[?x_n/a_n]$ *of a SAIQL query is a composition of substitutions*[3]*, one for every variable declared in the* `LET` *clause. The set of all possible solutions is called* S_{all}*.*

In the second step the `WHERE` clause is evaluated. The conjunction of the axiom patterns in the `WHERE` clause is split into single axiom patterns. Afterwards, each single axiom pattern is instantiated:

Definition 9. *An axiom pattern* p instantiated *with a solution* $s = [\boldsymbol{?x}/\boldsymbol{a}]$ *is an axiom pattern* $p_{[\boldsymbol{?x}/\boldsymbol{a}]}$ *where every occurrence of a variable is replaced by the value of its binding in the solution.*

By replacing each variable of a single axiom pattern with a constant value, we can decide for the resulting (instantiated) axiom if it is implied by O.

Definition 10. *Let* O *be the ontology that is specified in the* `FROM` *clause. A solution* $s = [\boldsymbol{?x}/\boldsymbol{a}]$ *is* valid *w.r.t. an axiom pattern* p*, if* $O \models p_{[\boldsymbol{?x}/\boldsymbol{a}]}$.

Definition 11. *A solution is* valid *w.r.t. the* `WHERE` *clause if it is valid w.r.t. every axiom pattern in the* `WHERE` *clause. The set of valid solutions* $S_v \subseteq S_{all}$ *of a SAIQL query is the set of solutions of this query, which are valid w.r.t. the* `WHERE` *clause.*

In the third and last step, the `CONSTRUCT` clause is evaluated and the result of the query is generated.

Definition 12. *The set of* resulting axioms *of a SAIQL query is the set of axiom patterns in the* `CONSTRUCT` *clause instantiated with every* $s \in S_v$.

The result of the query evaluation is a new set of axioms, i.e. a new ontology.

[3] Note that the composition of substitutions here is commutative. Hence, we can easily write a particular order of substitutions to denote an equivalence class of composed substitutions and call this one solution.

5.3 Example for Query Evaluation

As mentioned in our use case in section 2, an information system about a company's cars has to be developed. Because of the benefits of re-using ontologies, an appropriate part of the company's ontology, called `MotorOntology`, should be re-used. In order to extract the correct part of the original ontology, the following query is formulated: "Retrieve all class names that are subclasses of the cardinality restriction with the value 4 for the property `hasWheel`, and their descriptions and individuals which exist in the ontology `MotorOntology`!"

Given the ontology `MotorOntology` in Figure 1, the SAIQL query in Figure 2 is formulated. In the first step of the query evaluation, the `LET` clause is evaluated. Thereby, we extract $N_C = \{$Car, Convertible, MotorBike, Van$\}$, $N_I = \{$c, m, v$\}$ and, finally, $N_{CD} = N_C \cup \{$ restriction(hasWheel cardinality(2)), restriction(hasWheel cardinality(4)), intersectionOf(Car restriction(hasConvertibleTop someValuesFrom(owl:Thing))), intersectionOf(restriction(hasWheel cardinality(4)) restriction(hasSlidingDoor someValuesFrom(owl:Thing))), restriction(hasConvertibleTop someValuesFrom(owl:Thing)), restriction(hasSlidingDoor someValuesFrom(owl:Thing))$\}$.

Afterwards, the set of all possible solutions S_{all} is created. As the `LET` clause contains a variable `?i` representing individual names, a variable `?X` representing class names and a variable `?Z` representing class descriptions, $|S_{all}| = |N_I| \times |N_C| \times |N_{CD}|$ and $S_{all} = \{$[?i / c][?X / Convertible][?Z / restriction(hasWheel cardinality(2))], [?i / c][?X / MotorBike][?Z / restriction(hasWheel cardinality(4))], ..., [?i / v][?X / Van][?Z / restriction(hasSlidingDoor someValuesFrom(owl:Thing))] $\}$.

In the second step, the conjunction of the axiom patterns in the `WHERE` clause is split into single axiom patterns. In our example, the first and the third single axiom pattern is a class axiom pattern and the second single axiom pattern is an individual axiom pattern. After checking the first single axiom pattern, all solutions are removed in which ?X = MotorBike because MotorBike is not a subclass of restriction(hasWheel cardinality(4)).

After checking the second and the third single axiom pattern, $S_v \subseteq S_{all}$ is retrieved.

```
Class(Car partial Car)
Class(Car partial restriction(hasWheel cardinality(4)))

Class(Convertible partial Car)
Class(Convertible partial restriction(hasWheel cardinality(4)))
Class(Convertible partial Convertible)
Class(Convertible partial restriction(hasConvertibleTop someValuesFrom(owl:Thing)))
Class(Convertible partial Car
   restriction(hasConvertibleTop someValuesFrom(owl:Thing)))

Class(Van partial Car)
Class(Van partial restriction(hasWheel cardinality(4)))
Class(Van partial Van)
Class(Van partial restriction(hasSlidingDoor someValuesFrom(owl:Thing)))
Class(Van partial restriction(hasWheel cardinality(4))
   restriction(hasSlidingDoor someValuesFrom(owl:Thing)))

Individual(c type(Car))
Individual(v type(Car))
Individual(c type(Convertible))
Individual(v type(Van))
```

Fig. 3. Resulting OWL Ontology for the Given Query Example

In the third and last step, the `CONSTRUCT` clause is evaluated. Each single axiom pattern of the `CONSTRUCT` clause is instantiated with each valid solution $s \in S_v$. Thus, the classes Car, Van and Convertible, their descriptions and the individuals c and v are inserted as axioms into a new OWL ontology that is shown in Figure 3.

6 Related Work

The most common way for querying OWL DL for schema and instance information is by using languages like SPARQL [2] or RQL [10]. They can retrieve RDF triples that match a given pattern. Compared to SAIQL, the RDF query languages are not able to retrieve schema information from the T-Box, which is not explicitly stated. They can only query the T-Box by inspecting the underlying RDF triples on a syntactic level. By using SAIQL, explicit and inferred knowledge can be found.

For instance, in our running example in section 2 it would not be possible to achieve the same answer that was retrieved by the SAIQL query by performing OWL-QL or SPARQL queries. OWL-QL has only a restricted access to the T-Box so that only named classes and individuals can be retrieved. Thus, it is not possible to retrieve the class descriptions that define the named classes. SPARQL is not aware of OWL semantics. Thus, the class name `Convertible` could not be retrieved because it is not explicitly stated as a subclass of the cardinality restriction for the property `hasWheel`. Even if we use an OWL reasoner such as Pellet [11] to infer such a relation, there is no standard way to explicitly store the results of the inferencing in RDF. Additionally, there are ambiguous serializations e.g. for qualified number restrictions. In our use case restriction(hasWheel cardinality(4)) could also be expressed as intersectionOf(restriction(hasWheel minCardinality(4)) restriction(hasWheel maxCardinality(4))).

Although it is possible to extract the desired part of an ontology manually, this is naturally a harder task than specifying a SAIQL query and, of course, it is error-prone. Thus, it is certainly easier and more reasonable to use a SAIQL query instead. The need for such an automatic extraction is even increased if the former large ontology is changed and, thus, an update of the extracted ontology is required. Because changes of company-wide ontologies can happen very often, manual approaches for ontology extraction are almost infeasible. Further information about ontology extraction or ontology segmentation can be found in [12].

7 Conclusion and Future Work

In this paper we have presented a novel query language for OWL DL, called SAIQL (Schema And Instance Query Language), that is suited for querying the T-Box and the A-Box in a uniform way.

As one main contribution, our query language cannot only handle class names and individuals, but also class descriptions. The extraction of these class descriptions is of major importance as they provide the definition for a certain class name. By constructing OWL DL axioms that contain the extracted class names, individuals and class descriptions, the query answer will be a fully working OWL DL ontology that can be directly re-used.

We have described the syntax of our query language and we have provided a basic evaluation strategy for it. As we demonstrated in our running example, SAIQL is appropriate for extracting parts of an ontology (schema and instances) that shall be re-used in other applications.

In our future work, we will implement the evaluation of SAIQL queries. We will also extend the expressivity of SAIQL, e.g. by allowing more than one ontology, from which axioms can be retrieved, or by introducing variable bindings for properties. As another important extension, the retrieval of anonymous individuals and classes will be handled.

Acknowledgments

This work has been partially supported by the EU in the projects NeOn (IST-2006-027595) and K-Space (FP6-027026).

References

1. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: The Making of a Web Ontology Language . Journal of Web Semantics **1**(1) (2003)
2. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. Technical report, W3C http://www.w3.org/TR/rdf-sparql-query/, October 2006.
3. Fikes, R., Hayes, P., Horrocks, I.: OWL-QL - A Language for Deductive Query Answering on the Semantic Web. Web Semantics: Science, Services and Agents on the World Wide Web **2**(1) (2004) 19–29
4. Horrocks, I., Tessaris, S.: A Conjunctive Query Language for Description Logic Aboxes. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, AAAI Press / The MIT Press (2000) 399–404
5. Cali, A., Kifer, M.: Containment of Conjunctive Object Meta-Queries. In: VLDB'2006: Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB Endowment (2006) 942–952
6. Sleeman, D.H., Potter, S., Robertson, D., Schorlemmer, W.M.: Ontology extraction for distributed environments. In: Knowledge Transformation for the Semantic Web. IOS Press, Amsterdam (2003) 80–91
7. Elena Paslaru Bontas, Malgorzata Mochol, R.T.: Case Studies on Ontology Reuse. In: Proceedings of I-KNOW 05. (2005)
8. Patel-Schneider, P., Hayes, P., Horrocks, I.: Web Ontlogy Language (OWL) Abstract Syntax and Semantics. http://www.w3.org/TR/owl-semantics, February 2003.
9. Pan, J.Z., Horrocks, I.: Owl-eu: Adding customised datatypes into owl. J. Web Sem. **4**(1) (2006) 29–39
10. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: A Declarative Query Language for RDF. In: WWW '02: Proceedings of the 11th International Conference on World Wide Web, ACM Press (2002) 592–603
11. Sirin, E., Parsia, B.: Pellet: An OWL DL Reasoner. In Haarslev, V., Möller, R., eds.: Description Logics. (2004)
12. Seidenberg, J., Rector, A.: Web Ontology Segmentation: Analysis, Classification and Use. In: WWW '06: Proceedings of the 15th International Conference on World Wide Web, New York, NY, USA, ACM Press (2006) 13–22