NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

Priority: IST-2004-2.4.7 — "Semantic-based knowledge and content systems"

D4.2.2 Ontology customization and module creation: query-based customization operators and model

Deliverable Co-ordinator: **Noam Bercovici (UKO-LD)**

Deliverable Co-ordinating Institution: **University of Koblenz-Landau**

Other Authors: **Gerd Gröner, Simon Schenk, Alexander Kubias (all UKO-LD) and Martin Dzbor (OU)**

This deliverable presents a prototype of a technique for realizing ontology customization operators, which can be used to propose to the user a personalized view on an ontology for his/her current task or his/her job. This prototype is based on a query language specifically designed to cope with more expressive ontologies. Thus, this deliverable also introduces this new query language for OWL-DL, called SAIQL, that is able to query the t-box and a-box at the same time. The deliverable concludes with presenting this query language in a role of an operator realizing ontology pruning based on query templates and informal input from the user in the form of text corpus.

Document Identifier:	NEON/2008/D4.2.2/v1.0	Date due:	February 29, 2008
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	February 29, 2008
Project start date	March 1, 2006	Version:	v1.0
Project duration:	4 years	State:	Final
		Distribution:	Public

NeOn Consortium

This document is part of the NeOn research project funded by the IST Programme of the Commission of the European Communities by the grant number IST-2005-027595. The following partners are involved in the project:

Open University (OU) – Coordinator Knowledge Media Institute – KMi Berrill Building, Walton Hall Milton Keynes, MK7 6AA United Kingdom Contact person: Martin Dzbor, Enrico Motta E-mail address: {m.dzbor, e.motta}@open.ac.uk	**Universität Karlsruhe – TH (UKARL)** Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB Englerstrasse 11 D-76128 Karlsruhe, Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de
Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.ump.es	**Software AG (SAG)** Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com
Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Jesús Contreras E-mail address: jcontreras@isoco.com	**Institut 'Jožef Stefan' (JSI)** Jamova 39 SL–1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si
Institut National de Recherche en Informatique et en Automatique (INRIA) ZIRST – 665 avenue de l'Europe Montbonnot Saint Martin 38334 Saint-Ismier, France Contact person: Jérôme Euzenat E-mail address: jerome.euzenat@inrialpes.fr	**University of Sheffield (USFD)** Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield, United Kingdom Contact person: Hamish Cunningham E-mail address: hamish@dcs.shef.ac.uk
Universität Kolenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de	**Consiglio Nazionale delle Ricerche (CNR)** Institute of cognitive sciences and technologies Via S. Marino della Battaglia 44 – 00185 Roma-Lazio Italy Contact person: Aldo Gangemi E-mail address: aldo.gangemi@istc.cnr.it
Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de	**Food and Agriculture Organization of the United Nations (FAO)** Viale delle Terme di Caracalla 00100 Rome Italy Contact person: Marta Iglesias E-mail address: marta.iglesias@fao.org
Atos Origin S.A. (ATOS) Calle de Albarracín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.parientelobo@atosorigin.com	**Laboratorios KIN, S.A. (KIN)** C/Ciudad de Granada, 123 08018 Barcelona Spain Contact person: Antonio López E-mail address: alopez@kin.es

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to the writing of this document or its parts:

- University of Koblenz-Landau (UKO-LD)
- The Open University (OU)

Change Log

Version	Date	Amended by	Changes
v0.1	21.01.2008	Noam Bercovici	Creation of document
v0.2	24.01.2008	Noam Bercovici	Use Cases added
v0.3	05.02.2008	Noam Bercovici & Gerd Gröner	SAIQL details added
v0.4	11.02.2008	Noam Bercovici	Matching operator added
v0.5	22.02.2008	Noam Bercovici	Matching operator finalized
v0.6	14.03.2008	Noam Bercovici	Introduction and executive summary.
v0.7	04.04.2008	Martin Dzbor	Positioning of work, overall revision
v1.0	07.04.2008	Martin Dzbor & Noam Bercovici	QA, conclusions enriched

Executive Summary

The ontology becomes lager and more complex, therefore the user has to deal with irrelevant parts of the ontology. The customization can be one way to solve this problem by reducing the amount of information presented to the user and give to him/her a smaller and relevant ontology for his/her current task.

An important aspect of this deliverable is the presentation of a new query language for OWL-DL. Actually, 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. This deliverable presents SAIQL, the novel Schema And Instance Query Language for OWL DL, that is well suited for ontology extraction. This extraction makes the result usable by the user because the result of SAQIL is an OWL ontology. In this deliverable, we describe its syntax and explain a basic evaluation strategy.

The goal of this deliverable is to present several operators for customizing an ontology. In this context we present a matching operator which is able to extract a domain-oriented sub-ontology from a larger ontology covering different topics. The relevant concepts forming a domain module are identified by their frequencies in a sample group of domain-specific documents. Further, we describe a method, which can provide an input to SAIQL queries in the form of a set of relevant concepts based on the user's texts. This set of concepts is build from the specific domain texts. From there, SAIQL brings along an ontology formed with this query from the network of ontologies.

Note on Sources and Original Contributions

The NeOn consortium is an inter-disciplinary team, and in order to make deliverables self-contained and comprehensible to all partners, some deliverables thus necessarily include state-of-the-art surveys and associated critical assessment. Where there is no advantage to recreating such materials from first principles, partners follow standard scientific practice and occasionally make use of their own pre-existing intellectual property in such sections. In the interests of transparency, we here identify the main sources of such pre-existing materials in this deliverable:

- Chapter 2 a summary of chapter 6 of [DDM+07].
- Chapter 3 is partially based on [KSSP07].

Contents

List of Figures

Chapter 1

Introduction

This deliverable will be organized in three parts. First we will position this work in the NeOn project by reminding the reader of several test cases in which customization issues arise. Thereafter, we will present a new approach for querying OWL DL. Actually, the existing approaches for querying OWL DL either operate on syntactic constructs without taking into account the semantics of OWL or do have only a restricted access to the T-Box. We the present OWL-SAIQL, the novel Schema And Instance Query Language for OWL DL that is well suited for ontology extraction. We describe its syntax and provide a model-theoretic semantics for OWL-SAIQL. Next, we explain a basic evaluation strategy for OWL-SAIQL queries that is a suitable basis for optimization techniques known from database management systems. We illustrate the use of OWL-SAIQL with an example of ontology extraction and re-use. Finally, we will propose an operator for customizing an ontology using an input from domain-related texts and a generic query pattern realizing the operation of *ontology pruning*.

As ontologies become more and more complex and as they are integrated into networks of ontologies, it is reasonable to investigate the means, which would be capable of making a large network of complex ontologies more manageable. The customization and personalization of ontologies includes, in principle, two areas relevant to the NeOn project. First, there is a possibility to customize ontology view, e.g., during exploring a network of ontologies. This customization is more or less ad-hoc and the results of the customization may be discarded once the user proceeds with exploring the ontology. This customization during exploring an ontology tries to reduce the complexity of an ontology and only show parts which are relevant for the current user.

Second, one can customize ontology schema itself, for the purposes of reusing ontologies and integrating them into a network with other ontologies according to specific needs (e.g. during the ontology deployment, reasoning or design phases). Here the results of the customization will often be integrated into the edited ontology.

Obviously, since customization depends on such aspects as context or modularization, this work is related to other work packages in the NeOn project. In particular, we highlight a few points of overlap and where potential interactions can be found with the other work packages.

First, one particular form of creating networked ontologies investigate in WP1 is their modularization. An ontology module is seen as a tuple of imported ontologies, certain import and export interfaces, and a set of mappings. Formal models of modularization techniques have been recently presented in [dHR+07], and in line with this report, we use terms “operator” and “algebraic operator” to describe useful ways of amending the ontologies and creating the modules. What this deliverable contributes to the state-of-the-art on modularization is technique is:

- realization of a prototype of a modularization technique for pruning ontologies, and
- use of the prototyped technique to customize a view of a particular ontology based on a text corpus as an informal input from the user.

Relationship to WP2 is less obvious, however, in chapter 4 we use SAIQL queries as generic templates, which can be filled in with a concrete input from the user or which can be bootstrapped and executed on a concrete corpus of text data. In other words, these defined queries may be seen as a specific type of "ontology pattern"; albeit we use SAIQL query templates less as "ontology design pattern" and more in a role of a driver for simplifying views on an ontology.

In WP2 the research is focusing on a more generic set of patterns; mainly on those used during ontology design phase [Gan05]. Our use of SAIQL templates is complementary and more akin to navigational patterns that were introduced in the previous deliverable [DDM+07]. Navigational patterns may be used, e.g., to provide an initial view of a complex, multi-disciplinary ontology, so that the user has a more informed choice for selecting appropriate visualization metaphors, facets, etc.

With respect to research done in WP3, the relationship is somewhat clearer. Customizing an ontology by adjusting what is shown to a particular user (e.g., by pruning unnecessary nodes and branches) can be pragmatically, seen as bringing an ontology in the context of a particular user. Even more importantly, the selection and use of particular SAIQL query templates by a given group of users may be seen as a valid contextual modifier for segmenting users into groups and for automatically constructing user profiles. The added value of our work is that SAIQL queries represent user preferences in terms of procedural knowledge, i.e., *how* they prefer to interact with a given ontology; whereas the majority of other user profiling techniques is focusing on *what* the user interacts with.

We also mentioned in the previous works that a good visualization of ontologies is an important aspect of ontology construction tools. As found in several user studies (including our own reported in D4.1.1 [DMA+06]), visualization supported by the existing tools is rather poor, being too general and providing limited support for problem-specific needs. Concatenating visualization techniques with ontology customization operators (e.g., the one for pruning, described later in the report) may help to show only the relevant, more focused parts of ontologies, rather than showing the entire graphs with potential thousands of nodes. Thus, a good level of detail in visualizing can be offset by showing only a pruned part of an ontology – an approach that clearly complements the techniques for context-sensitive visualization that strive to show large and networked ontologies by abstracting the level of visualized details.

Chapter 2

Ontology Customization: Test Cases

Ontology customization operators can be applied in two main areas: viewing and exploring ontologies on one hand side, and designing and modifying ontologies on the other. The user may need to seamlessly switch between designing and viewing an ontology as it is explained in [DDM+07]. This chapter is a summary of chapter 6 of [DDM+07] and serves to refresh the motivation for the work carried out. For more details please refer to [DDM+07].

2.1 Annotation of a Report by an FAO Librarian

In this scenario we assume an FAO librarian who wants to annotate a new scientific report coming from a regional centre. The report draws on the existing knowledge, which already has been formally captured in various departments of FAO. Nevertheless, the report also introduces new topics, issues, and perhaps terminology. One of the librarian's tasks is to select appropriate ontologies from the FAO's ontology repository or from the public repositories on the Web that would sufficiently cover the submitted report. His or her challenge is to identify the most useful subset of all possibly relevant ontologies with no knowledge of formal methods and/or measures for "minimal coverage".

The librarian's task comprises several sub-tasks that need to be supported by the librarian's ontology engineering toolkit. The first sub-task, for instance, sees the librarian noting a set of terms or tags around what s/he perceives as a central theme of the report. The problem our librarian may face is to formulate and run many separate queries on the NeOn infrastructure in order to gather relevant existing schemas, data sources, ontologies, etc. Even if the librarian is willing to query the infrastructure many times, this is likely to lead to the problem of processing large numbers of potentially relevant ontologies and semantic data. In order to provide help, we propose giving the librarian an operator that would match the content of the report with a preliminary set of applicable ontologies (see section 4) to reduce this amount of results.

After selecting the ontologies that sufficiently cover the neighborhood of the new report the librarian proceeds with defining recommendations on typical navigational patterns. These may correspond to different combinations of views or facets (see in [DDM+07]), and reflect different relationships – within one ontology or between several ontologies. The purpose of this step is to enable data manager to pre-compute those facets that are most likely to be visited by the visitors to the FAO Portal. It is not necessary to define all such patterns, as there already are various patterns discovered for various user groups – thanks to NeOn Profiler. The Profiler's capability to mine large collections of seemingly ad-hoc data leads to several proposals of profile segments. For example, s/he can preview profiling the end user groups according to domain type and also role, like policymaker, fisheries manager, student, assessment expert.

In terms of contributing to the profiling patterns, we propose treating templates in using specific customization operators as one form of the "pattern" that characterizes a particular user, user group, or a specific area of interest.

2.2 Personalized View on a Network of Ontologies

User profiling for the purpose of Human Ontology Interaction can be seen as adjusting the user's view to the information contained in a profile. It can be understood as showing the data and the ontology through the semantic lens of a particular user. One possibility is to offer a personalized view on ontology by reducing the amount of information presented to the user. Namely, ontologies often contain more (structural) detail than is required for a specific task of the user, and that the user is willing to parse. Having the possibility to customize views on an ontology is an important feature for reducing their complexity to a manageable size, e.g. by abstracting or collapsing irrelevant elements using a selection of algebraic operators based on the preferred content or role of the user.

There are basically two ways how to create such personalized views: On the one hand, a view may only show content which is relevant for the domain of interest of an individual user (in the abstract sense, as defined in [DDM+07]). On the other hand, one may restrict the view to content which is relevant for the current task currently performed by a user. Typically, the user profile may contain information about the domains of interest of a user while the task or role based filtering will usually be predefined and be influenced by e.g. certain navigational patterns that are "imposed by" (or associated with) a specific role of a user.

It is also possible to combine these two kinds of views – here again, an algebraic operator supporting a guided merger of modules/views may be applicable. For example, the content-based filtering might be used to reduce an ontology for a domain like "fishing techniques" and the role-based filtering would further adapt the "fishing techniques" to suit the (typical) needs of a report annotator. The intersection of both views can then be used by an annotator specializing in reports about the fishing techniques etc. The creation of views on an ontology will often be ad-hoc and customized to a particular problem, task or situation. This implies, that the view creation has to be sufficiently easy and lightweight in order to be feasible for non-power users. It shall be reasonably simple as not to distract users from their primary goals in the given situation.

We may consider constructing the user profile in a semi-automatic manner that would enable the user to explicitly select a sub-set of ontologies to be used for constructing the profile. This strategy may be particularly useful if the user is active in several roles while constructing different ontologies and wants to have several independent user profiles – each connected to some but not all of the ontologies that s/he constructed in the past. Advantage of the semi-automatic approach to user profiling can also come from handling new users with no ontologies constructed to date; in such a case we may use active learning to build an initial user profile for the novices.

2.3 Ontology Integration

This scenario deals with the customization of ontologies during the design-time. In this scenario, customization means integration of already existing ontologies into a network of ontologies and adapting them according to the needs of a user. Usually, this task is manually performed, like in the user study described in [DMG+06] and [DMA+06]. This study and other similar experiments (e.g. [MFRW00]) show that the integration task is rather vague and is not well supported by standard ontology editors like Protege [NSD+01]. In the NeOn user study [DMA+06] the authors suggested it would be ideal, if this important task was supported by some kind of "ontology re-use wizard", which might be based e.g. on group- or role-profiles or on navigational patterns. This wizard or a pattern of user interaction with complex ontologies would guide the user through the process and offer her/him a pre-selected set of techniques that proved to be useful in a given role or for a specific task. In this report, we present an initial prototype using SAIQL query templates in a role of ontology customization templates. Obviously, our prototypes are not yet fully wrapped up into usable graphical user interfaces, but this would be the focus for the next period.

As we have seen in those different test cases, the most important issues related to ontology customization are:

- reducing ontology complexity;

- making sense of links and relations within/between ontologies;
- modularization and view customization based on different user-related criteria which can form a group or an user-profile;
- hiding the low-level aspects of several ontology engineering tasks.

It is the purpose of this deliverable to address these issues and provide an initial prototype operator for ontology pruning using a variety of parameters.

Chapter 3

SAIQL: A Query Language for OWL DL Ontologies

With the standardization of the Web Ontology Language OWL [HPSvH03], the use and re-use of ontological knowledge has gained significant momentum. For using web ontologies it is crucial to be able to access them in an intuitive and versatile manner. In contrast to RDF, where SPARQL [PS06] is providing a generic means to access RDF data *and* RDF Schema information, a corresponding query language is missing for OWL.

Using SPARQL [PS06] for querying OWL ontologies only allows the user to query the OWL A-Box and T-Box, but it is not aware of OWL-specific semantics. Furthermore, using SPARQL to query OWL data may get cumbersome very quickly – mainly, because of the triple semantics underlying SPARQL. For T-Box retrieval SPARQL uses syntax based on RDF and RDFS without using any "own" semantics.

Existing OWL querying approaches, e.g. OWL-QL [FHH04], have only restricted access to the T-Box, so that only named classes and individuals can be retrieved. Recently, SPARQL-DL [SP07] has been presented, which can also be used to query OWL DL ontologies. This language variant is aware of the OWL DL semantics. Unfortunately, SPARQL-DL cannot extract class descriptions and does not return complete OWL DL axioms. These properties are highly desirable as we will show in the following.

The requirements for querying OWL naturally include conjunctive queries of the OWL A-Box [HT00]. However, as has been recently argued for other ontology languages with explicitly queryable schema representations (cf. [CK06]), 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 using the task of ontology extraction (cf. [SPRS03]) for re-using parts of an ontology. While the existing 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 all classes, which are defined using a particular restriction on a certain property or having a specific subclass in their definition.

As one of its main contributions, our proposed query language can handle class descriptions and property names, in addition to class and individual names. 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, property names and individual names are returned contained in OWL DL axioms. Thus, the result is a fully working OWL DL ontology.

In the following, we present a small illustrative use case grounded in ontology extraction. We will use it to derive some requirements from it (section 3.1). We then discuss some of the foundations on which we build our approach in section 3.3. In section 3.4, we present the original querying language OWL-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. Although we use our query language to query OWL-DL ontologies in this chapter, we do not see any problems to use OWL-SAIQL for OWL Lite or OWL 1.1 ontologies.

```
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))
```

Figure 3.1: Example `MotorOntology` given in OWL DL (in OWL abstract syntax)

3.1 Use Case and Requirements

In our running example, we assume a large ontology, which we refer to as MotorOntology. We want to extract and re-use parts of this ontology for a new information system about cars in order to save costs and ensure higher quality (cf., [EPB05] on the benefits of ontology re-use). Naturally, we do not want to adopt the entire source ontology, as we are only interested in the schema-level and instance information related to cars in particular. Thus we want to "prune" such data as engine specifications, components of engines, road characteristics, etc.

For extracting our target ontology from `MotorOntology` (see an excerpt in Figure 3.1), we are interested in all axioms that contain class names that satisfy the condition of being subclasses of the cardinality restriction with the value 4 applied to the property `hasWheel`. We are also interested in retrieving and including their full descriptions. In addition, we are interested in all axioms about individuals of these classes. Given the running example in Figure 3.1, 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 too.

From the presented use case we derive the following requirements:

First, the query language should take into account 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 individual names, but also class descriptions and property names. 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, property names and individual names contained in OWL DL axioms so that the result is a fully working OWL DL ontology. Thus, it should be possible to use the result of an OWL-SAIQL query as an input for another OWL-SAIQL query.

As models of OWL ontologies in general are infinite, 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, property names and individual names 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. For instance, 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 $?Y$.

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

Figure 3.2: OWL-SAIQL query for our example

3.2 SAIQL for OWL DL in a Nutshell

In our running example, we wanted to extract the axioms for our target ontology from the `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 OWL-SAIQL query in figure 3.2 is formulated. Within this query, three variables `?i`, `?X` and `?Z` are used. The variable `?i` is treated as a placeholder for an individual name, whereas the variable `?X` is a placeholder for a class name and `?Z` for a class description.

Furthermore, the OWL-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.

3.3 Abstract Syntax and Semantics for OWL DL

OWL-SAIQL is a query language for OWL DL.

In this section, we present a brief overview of some foundational aspects characterizing OWL DL that we need for defining SAIQL for OWL later on. As is well known in the research community, OWL is a W3C recommendation for expressing ontologies on the Semantic Web [PH07]. The current OWL standard comprises three flavours (or species) of the language: OWL Lite, OWL DL and OWL Full. From these, we will refer to OWL DL, which is based on the description logic SHOIQ.

3.3.1 Abstract Syntax for OWL DL

In the following, we briefly summarize the abstract syntax for OWL DL by means of an extended BNF. The syntax is adopted from [PSHH]. 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 chapter. Furthermore, we omit datatypes and datatype properties. Based on this syntax, we will define the syntax of OWL-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 } ')'
   | 'EquivalentClasses(' description { description } ')'
   | 'SubClassOf(' description description ')'

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

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

3.3.2 Excerpts of OWL DL semantics

In the following, we describe parts of the model-theoretic semantics for OWL DL. The presented semantics is taken from [PH06]. We have slightly modified its presentation given here in order to use it more easily for the definition of OWL-SAIQL in the remainder of the chapter.

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. We denote their union as $N = N_C \cup N_{IP} \cup N_{DP} \cup N_I$. 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 an 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 3.3.1, datatypes and datatype properties are not considered in detail in this chapter. More details of the semantics of OWL DL can be found in [PSHH].

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, N_{CD} is a finite set of* class descriptions *(constructed from N) that consists of all class descriptions appearing in O.*

Users can add more class descriptions to the finite set N_{CD} that are necessary for their applications if need arises.

Example. Given the knowledge base in Figure 3.1, we have:

- $N_C = \{$Car, Convertible, MotorBike, Van$\}$
- $N_{IP} = \{$hasWheel, hasConvertibleTop, hasSlidingDoor$\}$

- $N_{DP} = \emptyset$
- $N_I = \{$c,v,m$\}$
- $N_{CD} = N_C \cup \{$ 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 OWL-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 [PH06], we distinguish class axioms, individual axioms and property axioms.

3.4 SAIQL for OWL DL

In this section the syntax and the model-theoretic semantics of OWL-SAIQL is described. Additionally, a basic evaluation strategy for OWL-SAIQL queries is proposed.

3.4.1 SAIQL Syntax

As mentioned before, axioms are the central elements of OWL DL ontologies and they are also important for OWL-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 names, class descriptions, individual-valued property names and individual names. The range of these variables can be either N_C, N_I, N_{IP} or N_{CD}. The name of a variable must start with a "?".*

As proposed for axioms without any variables, an axiom pattern p can be either a class axiom pattern pc or an individual axiom pattern pi. For the sake of simplicity, property axiom patterns are not considered within this chapter, but can be considered likewise.

Example. The OWL-SAIQL query in Figure 3.2 contains two individual axiom patterns and three class axiom patterns in the `CONSTRUCT` clause and in the `WHERE` clause. For instance, pc = `SubClassOf(?X ?Z)` is a class axiom pattern and pi = `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 3.2, 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

- *the* `constructClause` CC *contains a set of axiom patterns p_j, i.e. $CC = \bigcup_0^{\mathrm{m}} p_j$,*

- *the* `fromClause` FC *contains an URI reference of an ontology* O*,*
- *the* `letClause` LC *specifies the range of the variables and*
- *the* `whereClause` WC *contains a conjunction of axiom patterns* p_i*, i.e.* $WC = \bigwedge_0^{\mathrm{n}} p_i$*.*

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

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

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

axiomPattern ::= classAxiomPattern | individualAxiomPattern

className ::= URIreference
individualName ::= URIreference
ontologyID ::= URIreference
indProperty ::= URIreference

variableBinding ::= classNameBinding
   | individualNameBinding
   | classDescriptionBinding
   | indPropertyBinding

classNameBinding ::= 'ClassName' classNameVar {',' classNameVar}
individualNameBinding ::= 'IndividualName' individualNameVar
   {',' individualNameVar}
classDescriptionBinding ::= 'ClassDescription' classDescriptionVar
   {',' classDescriptionVar}
individualPropertyBinding ::= 'IndividualProperty' indPropertyVar
   {',' indPropertyVar}

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

classNameOrVar ::= classNameVar | className
indNameOrVar ::= individualNameVar | individualName
classDescOrVar ::= classDescVar | classDesc

classAxiomPattern ::=
   'SubClassOf(' classDescOrVar classDescOrVar ')'
   | 'DisjointClasses(' classDescOrVar classDescOrVar ')'
   | 'EquivalentClasses(' classDescOrVar classDescOrVar ')'

classDesc ::= classNameOrVar
   | restriction
   | 'UnionOf(' {classDescOrVar } ')'
   | 'IntersectionOf(' { classDescOrVar } ')'
   | 'ComplementOf(' classDescOrVar ')'

restriction ::= 'All(' indProperty classDescOrVar ')'
   | 'Some(' indProperty classDescOrVar ')'
   | 'Value(' indProperty indNameOrVar ')'
   | 'Min(' indProperty non-negative-integer ')'
   | 'Max(' indProperty non-negative-integer ')'
   | 'Exact(' indProperty non-negative-integer ')'

individualAxiomPattern ::=
   'Individual(' indNameOrVar 'type(' classDescOrVar ')' ')'
   | 'SameIndividual(' indNameOrVar indNameOrVar ')'
   | 'DifferentIndividuals(' indNameOrVar indNameOrVar ')'
```

3.4.2 Model-Theoretic Semantics for OWL-SAIQL

In this section, we extend OWL DL interpretations to give semantics for OWL-SAIQL.

Definition 8 *Given an ontology O and a corresponding N_{CD}, the OWL DL interpretation function $.^I$ can be extended in such a way that it also maps each class axiom pattern pc and each individual axiom pattern pi to a member of the boolean set $\{1, 0\}$.*
For each class axiom pattern $pc =$ SubClassOf(C, D), where $C, D \in N_{CD}$ are class descriptions from O: $pc^I = \begin{cases} 1 & \text{if } C^I \subseteq D^I \\ 0 & \text{otherwise} \end{cases}$.

For each class axiom pattern $pc =$ DisjointClasses(C, D): where $C, D \in N_{CD}$ are class descriptions from O: $pc^I = \begin{cases} 1 & \text{if } C^I \cap D^I = \emptyset \\ 0 & \text{otherwise} \end{cases}$.

For each class axiom pattern $pc =$ EquivalentClasses(C, D): where $C, D \in N_{CD}$ are class descriptions from O: $pc^I = \begin{cases} 1 & \text{if } C^I = D^I \\ 0 & \text{otherwise} \end{cases}$.

For each individual axiom pattern $pi =$ Individual$(a$ type$(C))$, where $a \in N_I$ is an individual name and $C \in N_{CD}$ is a class description from O: $pi^I = \begin{cases} 1 & \text{if } a^I \in C^I \\ 0 & \text{otherwise} \end{cases}$.

For each individual axiom pattern $pi =$ SameIndividual$(a\ b)$ where $a, b \in N_I$ are individual names: $pi^I = \begin{cases} 1 & \text{if } a^I = b^I \\ 0 & \text{otherwise} \end{cases}$.

For each individual axiom pattern $pi =$ DifferentIndividuals$(a\ b)$, where $a, b \in N_I$ are individual names: $pi^I = \begin{cases} 1 & \text{if } a^I \neq b^I \\ 0 & \text{otherwise} \end{cases}$.

In order to obtain ground instantiated axiom patterns, the variables of the axiom patterns have to be replaced by concrete values. Thereby, the solution space is restricted and the differentiation between class names and class descriptions is performed.

Definition 9 *A substitution $[?x_1/a_1]$ replaces a variable $?x_1$ by a value a_1, which is from the range as defined in the* `LET` *clause, that is from N_C, N_{CD}, N_{IP} or N_I. A* solution $s = [?\vec{x}/\vec{a}] = [?x_1/a_1][?x_2/a_2] \ldots [?x_n/a_n]$ *of a* `WHERE` *clause WC is a composition of substitutions*[1]*, one for every variable declared in the* `LET` *clause. The set of all syntactically possible solutions is called S_{all}.*

In order to determine if a solution is valid, the `WHERE` clause WC is instantiated with each solution.

Definition 10 *The* `WHERE` *clause $WC = \bigwedge_0^n p_i$ is instantiated with the solution $s = [?\vec{x}/\vec{a}]$ by instantiating its axiom patterns p_i with the solution s. The instantiated* `WHERE` *clause is written as $WC_s = WC_{[?\vec{x}/\vec{a}]}$. An axiom pattern p* instantiated *with a solution $s = [?\vec{x}/\vec{a}]$ is an axiom pattern $p_s = p_{[?\vec{x}/\vec{a}]}$ where every occurrence of a variable is replaced by the associated value in the solution.*

Definition 11 *A solution $s = [?\vec{x}/\vec{a}]$ is* valid *w.r.t. the* `WHERE` *clause WC if for each OWL interpretation I $(WC_{[?\vec{x}/\vec{a}]})^I = 1$. The set of valid solutions $S_v \subseteq S_{all}$ of a* `WHERE` *clause is the set of solutions, which are valid w.r.t. the* `WHERE` *clause.*

[1] 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.

The valid solutions, which fulfill the conditions in the WHERE clause can then be used in order to create new axioms according to the axiom patterns that are contained in the CONSTRUCT clause. Therefore, the axiom patterns in the CONSTRUCT clause have to be instantiated.

Definition 12 *The CONSTRUCT clause* $CC = \bigcup_0^n p_j$ *is instantiated with the solution* $s = [?\vec{x}/\vec{a}]$ *by instantiating its axiom patterns* p_j *with the solution* s*. The instantiated CONSTRUCT clause is written as* $CC_s = CC_{[?\vec{x}/\vec{a}]}$.

Definition 13 *The set of* resulting axioms *of a SAIQL query* N_{AX} *is the set of axiom patterns in the CONSTRUCT clause* CC *instantiated with every valid solution* $s = [?\vec{x}/\vec{a}]$*, i.e. every* $s \in S_v$*:* $N_{AX} = \bigcup_{s \in S_v} CC_s$

Note that OWL-SAIQL does not enforce the result in the CONSTRUCT clause to be consistent. This is up to the user, in order not to limit potential uses of OWL-SAIQL.

3.4.3 Basic Process for the Query Evaluation

In the following we describe the first and fairly primitive evaluation strategy for OWL-SAIQL queries. This strategy is not meant to be scalable, but to serve as a baseline for future work on efficient query engines. 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 by syntactically parsing the ontology, namely the finite set of class names N_C, the finite set of class descriptions N_{CD}, the finite set of individual-valued property names N_{IP} 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, as declared in the LET clause. Afterwards, the set of all syntactically possible solutions S_{all} is created by building the Cartesian product of the sets N_C, N_{CD}, N_{IP} and N_I according to the used variables.

In the second step the WHERE clause is evaluated. The conjunction of the axiom patterns in the WHERE clause is instantiated with each solution $s \in S_{all}$ and, then, it is decided for each solution s if it is valid or not. Each valid solution s is added to the set of valid solutions S_v. This step is suitable for further optimizations like join order processing or tree index access [d'A07].

In the third and last step, the CONSTRUCT clause is evaluated and the result of the query is generated. Therefore, the axiom patterns in the CONSTRUCT clause are instantiated with each valid solution $s \in S_v$ and, thus, new axioms are created. The result of the query evaluation is a new set of axioms, i.e. a new ontology.

3.4.4 Example of a Query Evaluation in SAIQL

As mentioned in our use case in section 3.1, an information system about company cars has to be developed. Because of the benefits of re-using ontologies, an appropriate part of the ontology underlying the domain for the new information system, 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 3.1, the OWL-SAIQL query in Figure 3.2 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$\}$
- $N_{CD} = N_C \cup \{$ 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))$\}$
- $N_{IP} = \{$hasWheel, hasConvertibleTop, hasSlidingDoor$\}$

Afterwards, the set of all syntactically 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 evaluated. 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 axiom patterns, $S_v \subseteq S_{all}$ is retrieved.

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 3.3 and Figure 3.4.

```
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))
```

Figure 3.3: Resulting OWL Ontology for the Given Query Example

3.5 Related Work

The most common way for querying OWL DL for schema and instance information is by using RDF query languages like SPARQL [PS06] or RQL [KAC+02]. They can retrieve RDF triples that match a given pattern. Unfortunately, these query languages are not aware of OWL semantics.

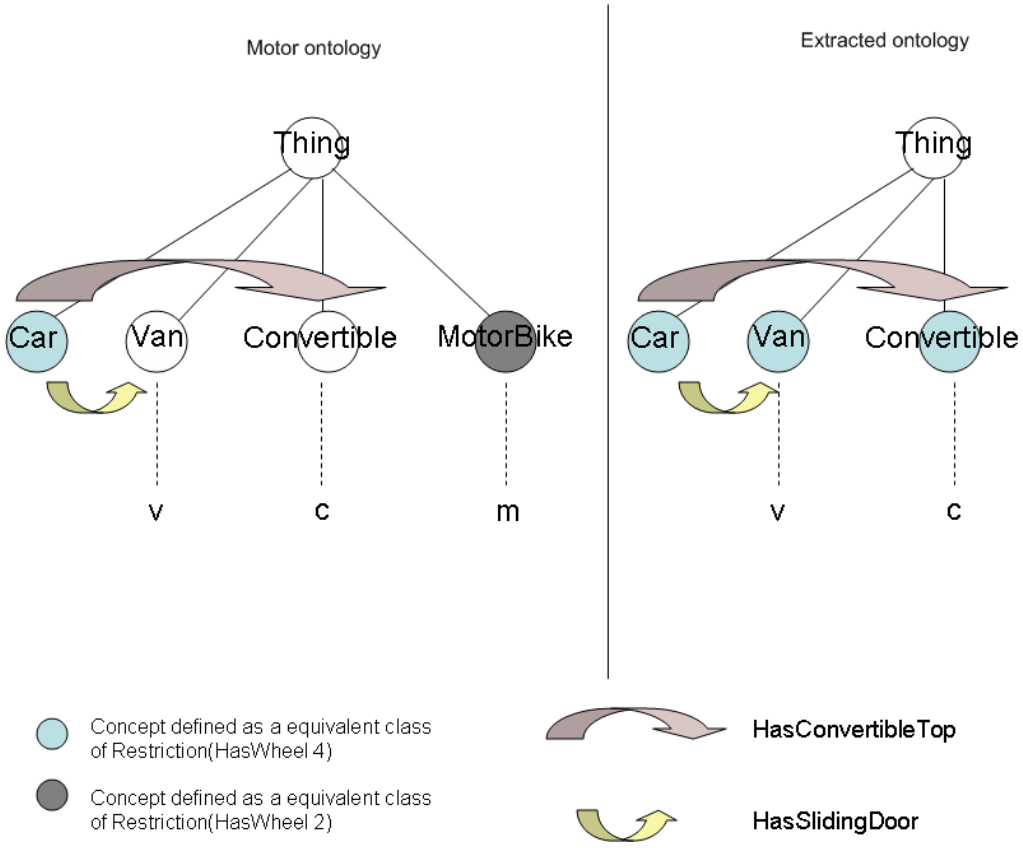

Figure 3.4: Graphical view of MotorOntology and the ontology result of the query example

For instance, using SPARQL [PS06] the class name `Convertible` in our example use case 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 [SP04] 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))).

Compared to OWL-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 OWL-SAIQL, explicit and inferred knowledge can be found. For instance, in our running example in section 3.1 it would not be possible to achieve the same answer that was retrieved by the OWL-SAIQL query by performing SPARQL queries.

There exist also some OWL query languages like OWL-QL [FHH04] or SPARQL-DL [SP07] that can be used to query OWL DL ontologies. However, they have also some shortcomings regarding the requirements of our example use case.

For instance, OWL-QL [FHH04] 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.

Very recently, a very interesting approach for querying OWL DL ontologies, namely SPARQL-DL [SP07], has been presented. SPARQL-DL is able to mix T-Box and A-Box queries and, thus, it is similar to our query language. Certainly, this query language is aware of the OWL DL semantics. Unfortunately, SPARQL-DL cannot extract class descriptions and it cannot extract whole OWL DL axioms. However, in the future we envision a synthesis of their approach and ours.

Chapter 4

Customizing Ontologies Using Relevant Texts

The general objective of a matching operator is to offer a personalized view of an ontology by reducing the amount of information presented to the user. As a first step, we look at an operator able to extract a domain-oriented sub-ontology from a larger ontology covering different topics. The relevant concepts forming a domain sub-ontology ("module") are identified by their frequencies in a sample group of domain-specific documents. Subsequently, we present an operator, which can provide an input to SAIQL queries in the form of a set of relevant concepts based on the user's texts. This set of concepts is build from the specific domain texts. From there, SAIQL brings along an ontology formed with this set of concepts from the network of ontologies. Those operators are based on the work of Kietz et al. in [KMV00] and Volz et al. in [VSML03].

4.1 Concept Acquisition from Texts

A lexicalisation is required in order to recognize a concept in the corpus. The field of the NLP [1] provide different techniques for the extraction of concepts from a text. Traditionally, this conversion of words to concepts has been performed using a thesaurus. The same technique can be used to compute the frequency of concepts. The Thesauri used are either specially created for this task, or are already pre existing language models, such as WordNet or equivalent.
The term frequencies [2] ($tf_{i,j} = \frac{n_{i,j}}{\sum n_{k,j}}$ [3]) is a measure well known in the information retrieval community. In [VSML03], they use a more elaborate measurement, *TF/IDF* [4] introduce by [SB87]. This weight is a statistical measure used to evaluate how important a word is to a document in a corpus. The importance increases proportionally to the number of times a word appears in the document but is balanced by the frequency of the word in the corpus.

$$TF/IDF_{i,j} = \underbrace{\frac{n_{i,j}}{\sum n_{k,j}}}_{Term \quad Frequency} * \underbrace{log(\frac{|number\ of\ document\ in\ the\ corpus|}{|d_i : t_i \in D_j|})}_{Inverted \quad Document \quad Frequency}$$

The mappings of words to concepts can often give ambiguous results. Typically each word in a given language will relate to several possible concepts. However, these ambiguities tend to be less important for the purposes of concept acquisition than they are with other close fields (e.g. machine translation). There are many techniques of disambiguation that may be used. Examples are linguistic analysis of the text and the frequency of the use of word and concept association that may be inferred from large text corpora.

[1] natural language processing (NLP)
[2] the frequency of a word in a text
[3] $n_{i,j}$ is the number of occurrences of the considered term in document d_j
[4] Term frequency/Inverted Document Frequency

4.2 Domain Focusing

After the extraction of the domain specific concepts from the texts (see 4.1) many generic concepts are mixed with the domain specific concepts, therefore we have to remove those concepts to focus on the user's domain of interest. The removal concept process follows the heuristic motivated by [KMV00], who brought up the idea that the domain specific concept must be more frequent in a domain specific corpus than in a general corpus. To carry out the method the user has to select carefully two corpora: one containing specific domain documents and one containing generic documents. This last corpus can be extracted from a news paper archive as they used in [KMV00].

At this point, the system identifies the occurrences of the concepts of the lingual ontology in the documents and then computes their frequency they are used in the documents relevant to the domain. The same computation is used in the generic documents. Then the system compares the concept frequencies of the two corpora (cf. figure 4.1). During this process the user can interact with the system in order to give a tolerance threshold for the comparison or in [VSML03] Volz et al. introduce different level of granularity: "ALL" which compare frequencies using all the documents of the corpora and "ONE"which compare frequencies using one document of the specific domain against the generic corpus. Kietz et al. give the following idea: "if a concept appears in the specific domain corpus but not in the ontology, the system should propose this concept to the user for adding it in the ontology".

When concepts are deleted the frequencies of this concept are then propagated to its supertypes. For example, if the system deletes "chair" the occurrence of "chair" will be reported on "furniture".

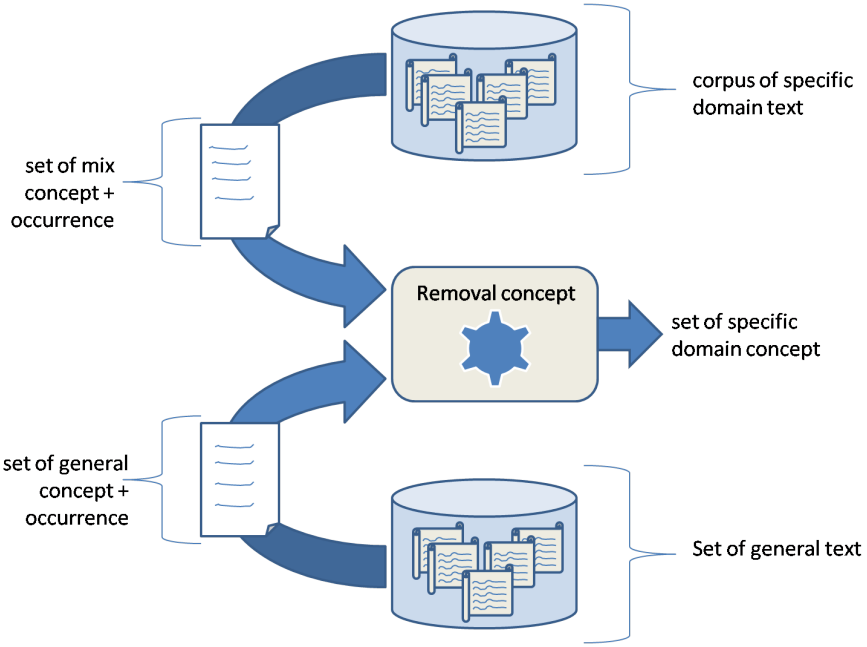

Figure 4.1: How does the matching operator work?

4.3 Text-Based Matching: a method

First, the user has to choose the specific domain text. The step seems trivial but this choice is crucial for the quality of the set of domain specific concepts as it is shown in [KMV00]. One way for the customization of an ontology is to personalize it for the user or a "kind" of user (cf. 2.2). This approach is perfect to create many different profiles. Indeed the texts given by the user strongly influence the result. So we know e.g. that the *Fisheries Managers* have as domains of interest the exclusive economic zones, major fishing areas and political land areas . Therefore the user profile *Fisheries Manager* can be built from corpora of this different

fields.

The next step is to acquire a set of specific-domain concepts from the texts chosen by the user. There are several techniques in the natural language processing to realize this task such as the term frequency (cf. 4.1). After this step the set of concepts contains a mix of domain-specific concepts and generic concepts. So we need to focus on the relevant concepts of the target domain. For this extraction, two sets of documents are essential. One of which includes domain specific documents and the other one sample group of generic documents (cf. 4.2). The result of the process is a set of specific domain concepts.

4.4 Customizing ontologies using the matching operator

This set of specific domain concepts can be used in different ways to customize ontologies, such as those underpinning our running example of a MotorOntology. We will present two selected methods. The first method is called pruning and thereafter, we will present a technique for extracting a relevant part of an ontology with SAIQL.

The objective of pruning is to reduce a huge ontology by cutting parts of this ontology which are not relevant for the currant task of the user. For that the pruning method requires as an input a set of relevant concepts for the user. There are different methods more or less automatable and more or less efficient. One of them keeps all the concepts linked to the relevant concept. Indeed this method is very generous in pruning elements because it does not delete any concept as long as the concept is related through subtype relationships or non-taxonomic relationships. By removing only obvious irrelevant elements, the pruning method will include elements having “isA” relations and association relations with originally chosen concepts.

4.4.1 Method for pruning of Swartout et al.

Swartout et al. developed this pruning method in [SPKR96] and applied it to a linguistic ontology called SENSUS. The SENSUS ontology contains over 50,000 concepts. Swartout et al developed a domain ontology for military air campaign planning from SENSUS. A domain expert selects “seed” terms that correspond to the relevant terms, manually he inserts those “seed” terms in SENSUS. In the pruning process, all the parent concepts are considered relevant. As a result of this process, we can identify a concept (e.g.frequent parent) having a lot of relevant subtype concepts. Then, all of its branch concepts including the concepts (e.g., military terms) considered irrelevant before will be added as relevant concepts.

This method involves a great deal of manual intervention. An expert must identify the “seed” terms, then the ontology designer have to find and select the concepts that represent those seed terms or add them if they are not present in the ontology, and finally accept or discard the relevant subtrees inferred automatically by the method. On the other hand, as shown by Swartout et al., this pruning method works well with linguistic ontology. Therefore, the ontology pruned by this pruning method can be very generic and large in size. Swartout et al. reported that 60 seed terms with their pruning method generate a pruned ontology with around than 1,600 concepts.

4.4.2 Method for pruning of Volz and Maedche

This method is described in [KMV00] and [VSML03]. Its goal is to create a domain ontology as automatically as possible. The method uses a linguistic base ontology and a set of documents (cf.4.3) in order to automate the pruning activity. The phase of removal concept described in 4.2 deletes the concepts that were not relevant. In this step the user may be active in the pruning phase to validate the pruning process and delete or select concepts manually. This method is highly automatic because the user’s task is limited to select two sets of documents, which can easily be obtained from an intranet or the Internet. The time of selecting these documents is insignificant in comparison to create a set of specific domain concepts manually. In addition those documents are also used to detect potential refinements in the ontology. This method was designed

for pruning linguistic ontologies, but it uses an ontology language that is an extension of the RDF Schema and supports the definition of several integrity constraints.

The test of this method proposed by [VSML03] includes two kinds of generic corpora, one being really generic e.g., a newspaper archive (such as Reuters) and the other being for the purposes of extraction concept a corpus close to the domain of interest of the user. The results of this method show clearly that the choice of the generic corpus has an influence on the result. Indeed "the generic corpus closely related to the target domain leads a bigger upper-level of the ontology, i.e. allow to generalize the resulting ontology".

4.4.3 Pruning with SAIQL

SAIQL (cf. chapter 3) needs a query for bootstrapping the process of customizing an ontology. It is this query that indicates how the ontology will be customized. In this section we will present a few SAIQL query templates, or *customization patterns*, which can be use for pruning this ontology.

There is different ways for using this query pattern, the first solution is to use as an input, the result of the matching operator as shown is the figure 4.2. The second solution would be to let the user give his relevant concept or descriptions of concept to the system for his current task.

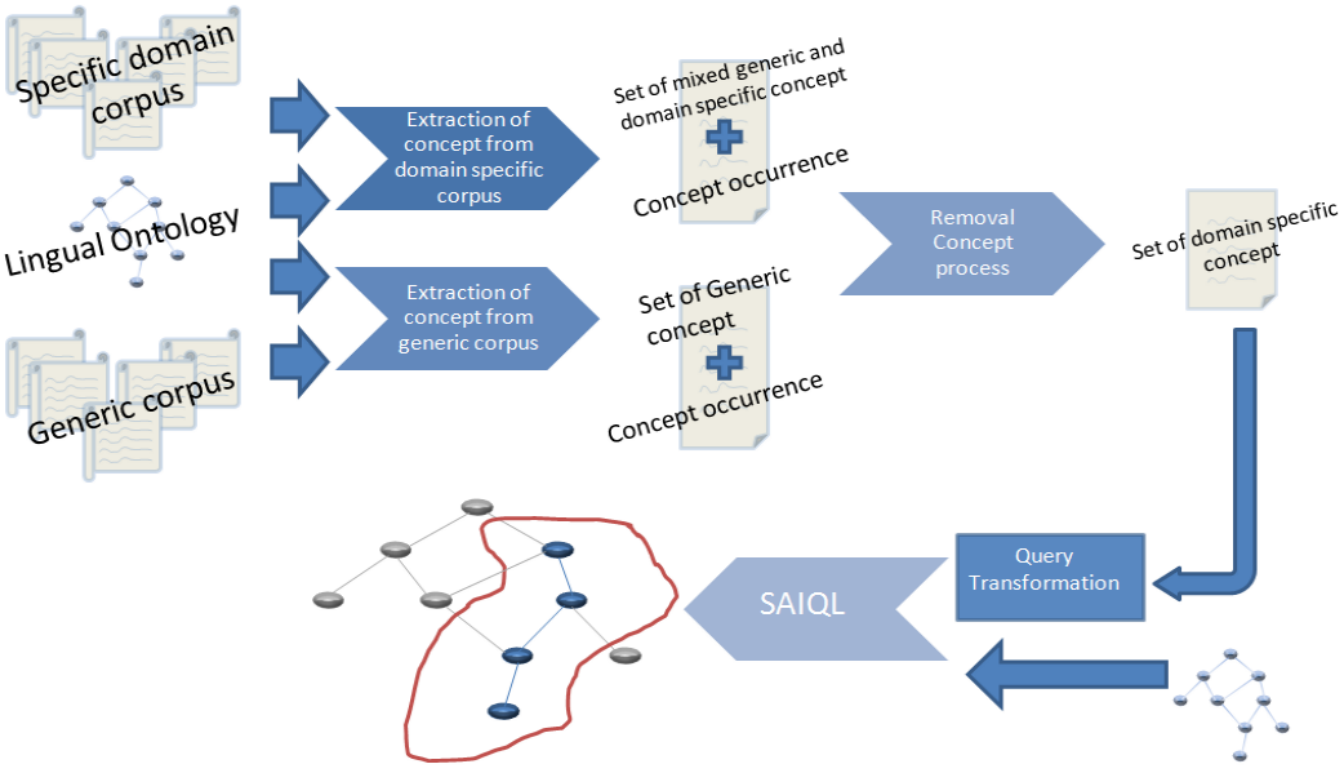

Figure 4.2: The methodology to combine the matching operator and SAIQL

Figure 4.3 presents a query pattern, which can cover all the necessary parts of the OWL-SAIQL syntax for the user. This template or pattern can be, in principle, hidden behind a graphical user interface of an ontology customization wizard or a questionnaire. Hence, the user will only provide inputs in the form of key concepts, and the (to be developed) NeOn Toolkit plugin will take care of realizing the appropriate query. This query tempalte/pattern is an example of what can be the key benefit of using a query language like SAIQL for customizing ontologies compared to the different pruning methods presented earlier (cf. the two previous sub-sections).

```
CONSTRUCT SubClassOf(?X ?Z); Individual(?i type(?X))
FROM OntologyURI
LET IndividualName ?i; ClassName ?X; ClassDescription ?Z
WHERE SubClassOf( ?X UnionOf( list of relevant concepts
or/and relevant descriptions of concept ) )
AND Individual( ?i Type( ?X )) AND SubClassOf( ?X ?Z )
```

Figure 4.3: SAIQL query pattern for pruning from concepts and descriptions of concept

Actually, the above query template/pattern extracts all the concepts, their individuals and the description concepts related to them. The benefit provided by SAIQL in this specific case is twofold: First we are not only pruning an ontology but at the same time we extract a part of this ontology. This means the part comprising the relation, the concepts, their description and the associated individuals. Whereas the previous pruning methods only provide the concepts and their subsumed concepts. The second benefit of the query patterns in SAIQL is that they allow a very fine-grained statements to be crafted and organized into "families" or groups that can be eventually invoked by the user through a graphical user interface. Hence, query-based customization operators can be numerous and can include both generic and highly domain-specialized templates, which leads to substantially more numerous ways for customizing ontology too.

We proposed to test the above query with an example presented in the WP8 (cf. [GDM+06]) concerning the pharmaInnova invoicing schema ontology developed to support NeOn use case on invoicing integration. The objective of the pharmaInnova project is to improve commercial relations between laboratories and their suppliers and customers, by introducing electronic interchanges of invoices to decrease time and cost instead of using traditional documents in paper. The result is a large ontology covering financial, organizational, legal, and many other aspects. Figure 4.4 shows a query for pruning this ontology, a list of relevant concepts to seed the pruning process has been provided by ISOCO. The result of this query is expected to be a module of the pharmaInnova ontology that contains all the concepts and their description related either to *CurrencyMeasure* or *PICompany* or *QuantityType* or *PharmaInnovaInvoice* or *PIAmount* or the concept that has got at least one *body* and exactly one *header* and one *summary*, thus realizing every *invoice concept*.

At those concepts we add their subsumptions and all the relations attached to those previous concepts. This query also adds the individuals of all the found concepts.

```
CONSTRUCT SubClassOf(?X ?Z); Individual(?i type(?X))
FROM OntologyURI
LET IndividualName ?i; ClassName ?X; ClassDescription ?Z
WHERE SubClassOf( ?X UnionOf(CurrencyMeasure PI-
Company QuantityType PharmaInnovaInvoice PIAmount
IntersectionOf(min(hasBody 1) exact(hasHeader 1) ex-
act(hasSummary 1))) )
AND Individual( ?i Type( ?X )) AND SubClassOf( ?X ?Z )
```

Figure 4.4: SAIQL query pattern for pruning from concepts

We have run the above query template filled with the data provided by ISOCO on the current implementation of SAIQL. While SAIQL was able to process the query and start the evaluation of the resulting sub-queries, it was observed the entire process of query decomposition was not sufficiently efficient to conclude query evaluation. Thus, this application of a generic pruning operator based on a SAIQL query did not yield acceptable results within the time allocated to query execution. Due to the current implementation of OWL-SAIQL the performance of this prototype operator suffered when the degree of complexity of the processed ontology increased – the exploration of the space of *all* possible sub-queries is an activity with exponential complexity and the query engine can rapidly run out of memory. Nevertheless, this prototyping and the formative evaluation of the prototypes on a real-world ontology provided a valuable test, as it pointed to concrete features of the SAIQL implementation that can be targeted in the subsequent period.

We learned from this formative test phase one main point where it failed. This point is about the first implementation of SAIQL as it has been implemented as a proof of concept and not as a technique fully deployable via a NeOn Toolkit plugin. This "naive" implementation of query evaluation strategy could not cope with large

ontologies; mainly because of a cartesian product is done to compute the tuples of all the possible solution for the query. Actually, the current query evaluation strategy did compute all tuples of the possible solution; only that in the case of pharmaInnova [5] the engine has computed more than 55 million [6] tuples, and thus the host computer quickly ran into a problem of memory shortage. We have continued with testing to obtain a result on a part of pharmaInnova ontology and with strict execution time constraints, but only a very small part of the total space of solutions managed to give a result for this pruning query (cf.4.3). Those parts of the pharmaInnove ontology that got through the engine within given time constraints were too small and not sufficiently representative to make a conclusion.

Clearly, query-based operators exhibit potential, but more work needs to done on the level of optimizing the query execution strategies and possibly adapting the query execution strategies to fit the pattern of complexity embedded in a particular "WHERE" clause of the query. In our formative tests we opted for a rather ambitious disjunctive "WHERE" clause, which leads us to including more optimization of the query execution being a key work in the next period. This optimization may be achieved, e.g., by scheduling sub-query execution so that sub-queries get processed *progressively*. In other words, simpler sub-queries can be processed before the engine reaches more complex combinations of tuples – we expect this would enable us to cover a much larger portion of the space of potential solutions in a systemic manner. Consequently, this will enable us to obtain partial results even for more complex ontologies, where the naive query execution models would rapidly degrade due to getting "stuck" with exploring complex tuples before simpler variants.

Currently we are working on an optimized version of SAIQL, which will implement several query optimization techniques as suggested above, and we expect this re-design will lead to the improvement of the engine performance and the rectification of the memory shortage and time performance shortcomings the current implementation is prone to.

[5] PharmaInnova ontology is composed of 700 class named, 64 individuals and 1232 description concepts
[6] 700*64*1232=55193600

Chapter 5

Conclusion and future work

In this deliverable we have presented a novel query language for OWL DL, called OWL-SAIQL (Schema And Instance Query Language), that is suitable for querying the T-Box and the A-Box in a uniform way. We have proceeded to use this language as a basis for prototyping a few ontology customization operators, with a focus being given to ontology pruning.

As one main contribution, our query language does not only handle class names and individuals, but also supports class descriptions and property names. The extraction of these class descriptions is of major importance in the task of ontology customization, as these additional expressive features provide the definition for every class name. By constructing OWL DL axioms that contain extracted class names, individuals, property names and class descriptions, the query answer will be a fully working OWL DL ontology that can be directly re-used. Hence, by applying this enriched querying paradigm we have truly achieved the key goal of producing a new ontology by customizing an existing one. In the deliverable, we have described the syntax and the model-theoretic semantics of the proposed query language, OWL-SAIQL. In addition, we have provided a basic evaluation strategy for OWL-SAIQL queries. As we demonstrated in our running example, OWL-SAIQL is appropriate for extracting parts of an ontology (schema and instances) that shall be re-used in other applications.

Next, we presented details of a specific operator for customizing an ontology taking input from an informal corpus of related texts provided by the user. In addition, we have shown that this operator can be easily automated by treating the SAIQL queries as templates that can be filled in by concrete class names, possibly arising from the associated text corpus. Also, a query pattern for pruning ontology has been provided. An example of such query templates is proposed for reducing the “pharmaInnova invoicing schema” ontology that was originally presented in WP8. We have also implemented the basic strategy for evaluating OWL-SAIQL queries using the Pellet reasoner [SP04].

This initial prototyping has enabled us to gain valuable lessons to shape further research into using query-based ontology customization techniques as viable “operators”; i.e., something that can be used directly by a non-expert user from within ontology engineering environments such as NeOn Toolkit. With respect to pruning the “pharmaInnova” ontology we found that the performance of the current OWL-SAIQL implementation was not satisfactory to provide meaningful results for this ontology. In particular, the size of the ontology is problem for the actual query evaluation mechanism.

In this deliverable we pointed out some general benefits of using the combination of SAIQL and the matching operator, for example, the ability of SAIQL to extract a complete ontology from a few concepts and their descriptions. In this deliverable we have used queries directly, rather than being accessed through a NeOn Toolkit plugin, because this work is still at a prototyping stage and subject to substantial improvements over the next period. Nonetheless, the user interaction component to wrap the low-level query formulation is, obviously, an important part of our future agenda in this task. As observed in several user studies, including those arising from NeOn, query formulation in languages like SPARQL or SAIQL is not something an ordinary user can do and will do. Hence, one short-term opportunity for further work is to bring this querying support into the toolkit, hiding query templates behind some more accessible “ontology customization wizards”.

In our future work, we will also need to extend the expressivity of OWL-SAIQL, e.g., by allowing more than one ontology, from which axioms can be retrieved. This capability will push the approach beyond retrieval from a single ontology – the current point of departure for all declarative query mechanisms – towards the retrieval on the Semantic Web at large and in the networks of ontologies in particular.

We are currently re-implementing parts of the first implementation of SAIQL based on the lessons learnt from this early prototyping activity. In fact, this first implementation prototype was an extremely useful proof of concept. One of its key shortcomings relates to dealing with large ontologies, where the following failures have been observed: misuse of memory. The problem come from the basic evaluating query strategy used by this first version. Some answers at this problem can be solve because of the query optimization field.

This prototypical approach to realizing the OWL-SAIQL query engine and using it as a basis for constructing ontology customization operators can be improved by using query optimization techniques, e.g., estimating the complexity of different “WHERE” clause patterns prior to evaluating them. This will allow to address the issue of the capabilities of SAIQL to address with ontologies of a larger size and the improvement of the answering time.

As a second step, we intend to build in a progressive query evaluation mechanism; i.e., we first need to evaluate those queries that exhibit less complex “WHERE” clause patterns. Since this might be an iterative activity, we will need to re-estimate the complexity of the remaining, i.e., not yet evaluated, “WHERE” clause patterns. Step by step, we progressively get to evaluating more complex clauses, thus achieving the objective of real-time responsiveness of an ontology customization operator without sacrificing its ontological and semantic validity. With this amended query evaluation and execution strategy we hope to improve the capabilities of SAIQL to cope with ontologies of a larger size and higher degree of complexity.

Further optimization is likely to address the need for an improved reasoning support, in order to improve the time it takes SAIQL to derive an answer to the query. We expect to achieve a test of different inference engines to evaluate which one is the most performance combined with SAIQL.

With the different optimization that we are implementing, SAIQL is appearing as polyvalent and great potential operator for the customization using querying. This novel technic of customization is a good thing to aim for giving to the user expert or not a easy way to personalized an ontology. But it is a work in its infancy, so our prototyping is good opportunity to obtain feedback and shape the query language early during its design, before any standardizations are attempted.

Appendix A

Manual for SAIQL Query Engine

A.1 Start of the Application

After starting, the application should look like shown in the following figure A.1.

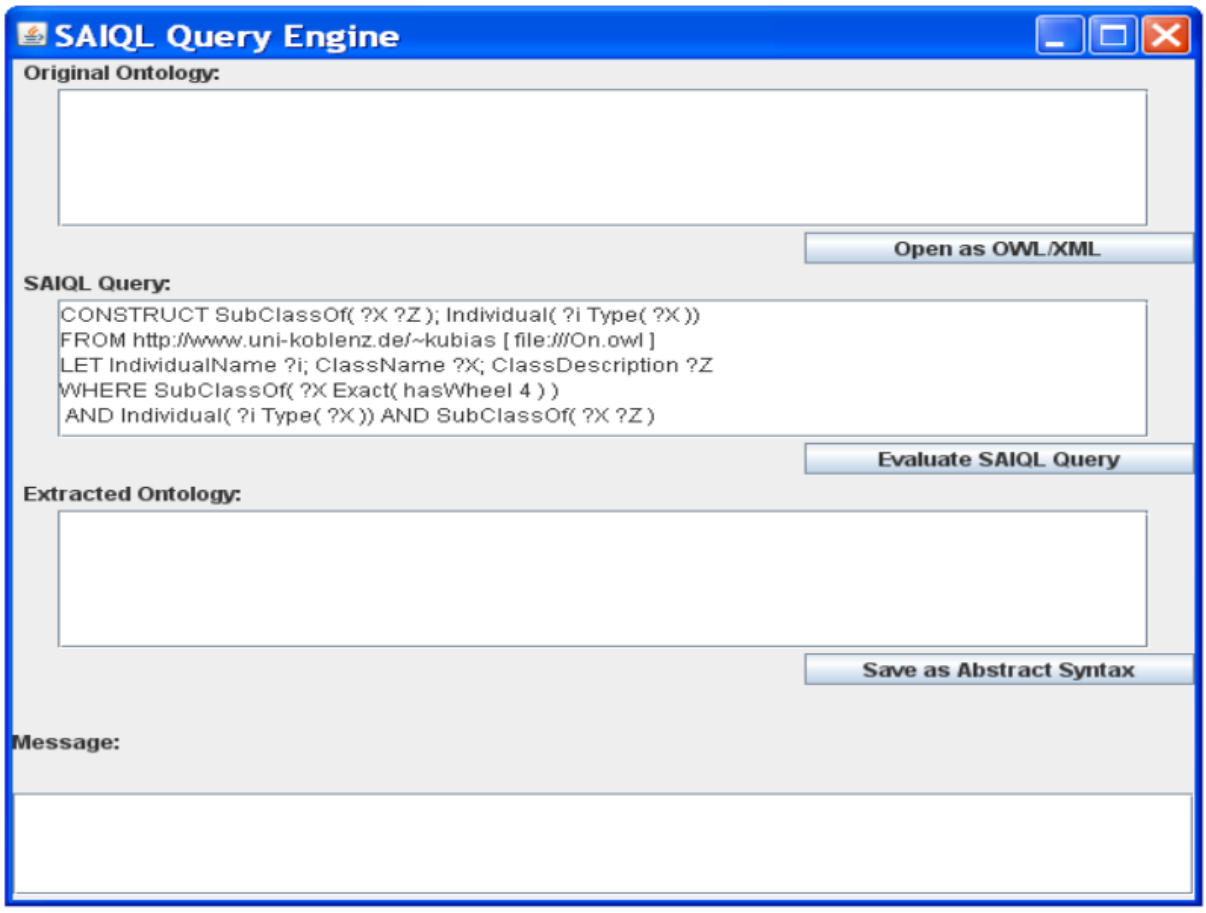

Figure A.1: The main window of SAIQL

A.2 Loading of the Original Ontology

The first step is to open or to load the Original Ontology. SAIQL extract elements from this ontology. Use the button „Open as OWL/XML“ in the section „Original Ontology“. Afterwards the ontology is loaded and shown as OWL Abstract Syntax in the memo-textfield, below the label „Original Ontology“. This situation is demonstrated in the next figure A.2.

Figure A.2: The window for loading an ontology in SAIQL

This action changed the SAIQL-expression in the next memo-textfield. The line: FROM http://www.uni-koblenz.de/~kubias // [file:/C:/ISWEB/Software/workspaceSAIQL/SAIQLQueryEngine/MotorOntology.owl].

A.3 Entering the SAIQL Query

The next step is the input of the SAIQL Query in the second Memo-Textfield in the section „SAIQL Query“.

Note: The SAIQL-Syntax was changed in the meantime and is now as described in the SAIQL-Techreport-2007 and shortened demonstrated below:

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

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

axiomPattern ::= classAxiomPattern | individualAxiomPattern

className ::= URIreference individualName ::= URIreference
ontologyID ::= URIreference indProperty ::= URIreference

variableBinding ::= classNameBinding
   | individualNameBinding
   | classDescriptionBinding
   | indPropertyBinding

classNameBinding ::= 'ClassName' classNameVar {',' classNameVar}
individualNameBinding ::= 'IndividualName' individualNameVar
   {',' individualNameVar}
```

```
classDescriptionBinding ::= 'ClassDescription' classDescriptionVar
   {',' classDescriptionVar}
individualPropertyBinding ::= 'IndividualProperty' indPropertyVar
   {',' indPropertyVar}

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

classNameOrVar ::= classNameVar | className indNameOrVar ::=
individualNameVar | individualName classDescOrVar ::= classDescVar |
classDesc

classAxiomPattern ::=
   'SubClassOf(' classDescOrVar classDescOrVar ')'
   | 'DisjointClasses(' classDescOrVar classDescOrVar ')'
   | 'EquivalentClasses(' classDescOrVar classDescOrVar ')'

classDesc ::= classNameOrVar
   | restriction
   | 'UnionOf(' {classDescOrVar } ')'
   | 'IntersectionOf(' { classDescOrVar } ')'
   | 'ComplementOf(' classDescOrVar ')'

restriction ::= 'All(' indProperty classDescOrVar ')'
   | 'Some(' indProperty classDescOrVar ')'
   | 'Value(' indProperty indNameOrVar ')'
   | 'Min(' indProperty non-negative-integer ')'
   | 'Max(' indProperty non-negative-integer ')'
   | 'Exact(' indProperty non-negative-integer ')'

individualAxiomPattern ::=
   'Individual(' indNameOrVar 'type(' classDescOrVar ')' ')'
   | 'SameIndividual(' indNameOrVar indNameOrVar ')'
   | 'DifferentIndividuals(' indNameOrVar indNameOrVar ')'
```

A.4 Evaluation of the SAIQL Query

The next step is the evaluation of the SAIQL Query. Use the button “Evaluate SAIQL Query”. This section of the application is shown in the figure belowA.3.

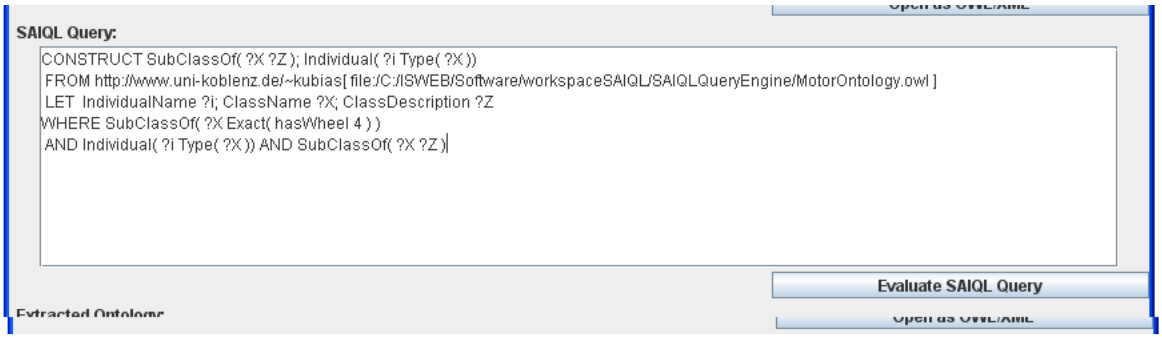

Figure A.3: The window for quering SAIQL

The evaluation process will last for some time and could even pause the GUI during this action.

A.5 Display of the extracted ontology

The extracted ontology“ is shown in the Memo-Textfield „Extracted Ontology“ in OWL Abstract Syntax.

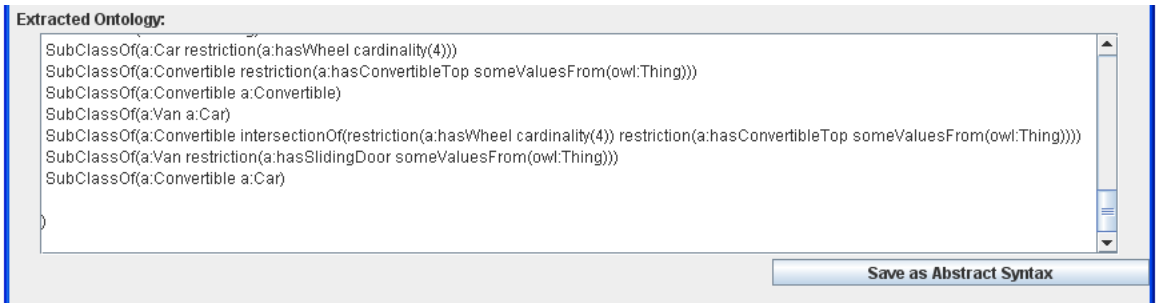

Figure A.4: The window with the result of SAIQL

A.6 Save the extracted Ontology

The extracted ontology could be saved as OWL Abstract Syntax, using the „Save as Abstract Syntax“ button.

Bibliography

[CK06] Andrea Cali and Michael Kifer. Containment of Conjunctive Object Meta-Queries. In *VLDB'2006: Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 942–952. VLDB Endowment, 2006.

[d'A07] C. d'Amato. *Similarity-based Learning Methods for the Semantic Web*. PhD thesis, University of Bari, 2007.

[DDM+07] Klaas Dellschaft, Martin Dzbor, Dunja Mladenic, Alexander Kubias, Carlos Buil Aranda, and Jose Manuel Gomez. Review of methodds and models for customizing/personalizing ontologies. Deliverable D4.2.1, NeOn Project, 2007.

[dHR+07] Mathieu dŠAquin, Peter Haase, Sebastian Rudolph, Jerome Euzenat, Antoine Zimmermann, Martin Dzbor, Marta Iglesias, Yves Jacques, Caterina Caracciolo, Carlos Buil Aranda, and Jose Manuel Gomez. Neon formalisms for modularization: Syntax, semantics, algebra. Deliverable D1.1.3, NeOn Project, 2007.

[DMA+06] M. Dzbor, E. Motta, C. Buil Aranda, J.M. Gomez, O. Goerlitz, and H. Lewen. Developing ontologies in OWL: An observational study. In *Online Proceedings of the Workshop on OWL: Experiences and Directions 2006*, 2006.

[DMG+06] M. Dzbor, E. Motta, J. M. Gomez, C. Buil Aranda, K. Dellschaft, O. Grlitz, and H. Lewen. Analysis of user needs, behaviours & requirements wrt. user interfaces for ontology engineering. Deliverable D4.1.1, NeOn Project, 2006.

[EPB05] Robert Tolksdorf Elena Paslaru Bontas, Malgorzata Mochol. Case Studies on Ontology Reuse. In *Proceedings of I-KNOW ÂŠ05*, 2005.

[FHH04] Richard Fikes, Patrick Hayes, and Ian Horrocks. 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):19–29, 2004.

[Gan05] Aldo Gangemi. Ontology design patterns for semantic web content. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *Proceedings of ISWC'05: the 4th International Semantic Web Conference, Galway, Ireland, November 6–10, 2005*, volume 3729 of *Lecture Notes in Computer Science*, pages 262–276. Springer, 2005.

[GDM+06] Jose Manuel Gomez, Claire Daviaud, Berta Morera, Richard Benjamins, Tomas Pariente, Lobo German, Herrero Carcel, and Gloria Tort. Analysis of the pharma domain and requirements. Deliverable D8.1.1, NeOn Project, 2006.

[HPSvH03] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics*, 1(1), 2003.

[HT00] Ian Horrocks and Sergio Tessaris. 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*, pages 399–404. AAAI Press / The MIT Press, 2000.

[KAC+02] Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. RQL: A Declarative Query Language for RDF. In *WWW '02: Proceedings of the 11th International Conference on World Wide Web*, pages 592–603. ACM Press, 2002.

[KMV00] Joerg-Uwe Kietz, Alexander Maedche, and Raphael Volz. A method for semi-automatic ontology acquisition from a corporate intranet. In *Proc. of Workshop Ontologies and Text, co-located with the 12th International Workshop on Knowledge Engineering and Knowledge Management (EKAW'2000)*, Juan-Les-Pins, France, 10 2000.

[KSSP07] Alexander Kubias, Simon Schenk, Steffen Staab, and Jeff Z. Pan. Owl saiql - an owl dl query language for ontology extraction. In *Proceedings of the 2007 International Workshop on OWL: Experiences and directions (OWLED-07)*, 2007.

[MFRW00] D. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning*, 2000.

[NSD+01] N. Noy, M. Sintek, S. Decker, M. Crubezy, R. Fergerson, and M. Musen. Creating semantic web contents with proteg 2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.

[PH06] Jeff Z. Pan and Ian Horrocks. Owl-eu: Adding customised datatypes into owl. *J. Web Sem.*, 4(1):29–39, 2006.

[PH07] Jeff Z. Pan and Ian Horrocks. RDFS(FA): Connecting RDF(S) and OWL DL. *IEEE Trans. Knowl. Data Eng.*, 19(2):192–206, 2007.

[PS06] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. Technical report, W3C, 2006. http://www.w3.org/TR/rdf-sparql-query/, October 2006.

[PSHH] P. Patel-Schneider, P. Hayes, and I. Horrocks. Web Ontology Language (OWL) Abstract Syntax and Semantics. http://www.w3.org/TR/owl-semantics, February 2003.

[SB87] Gerard Salton and Chris Buckley. Term weighting approaches in automatic text retrieval. Technical report, Ithaca, NY, USA, 1987.

[SP04] Evren Sirin and Bijan Parsia. Pellet: An OWL DL Reasoner. In Volker Haarslev and Ralf Möller, editors, *Description Logics*, 2004.

[SP07] Evren Sirin and Bijan Parsia. SPARQL-DL: SPARQL Query for OWL-DL. In *Proceedings of the 2007 International Workshop on OWL: Experiences and directions (OWLED-2007)*, 2007.

[SPKR96] B. Swartout, R. Patil, K. Knight, and T. Russ. Toward distributed use of large-scale ontologies. In *the 10th Workshop on Knowledge Acquisition*, Banff, Canada, 1996.

[SPRS03] Derek H. Sleeman, Stephen Potter, Dave Robertson, and W. Marco Schorlemmer. Ontology extraction for distributed environments. In *Knowledge Transformation for the Semantic Web*, pages 80–91. IOS Press, Amsterdam, 2003.

[VSML03] R. Volz, R. Studer, A. Maedche, and B. Lauser. Pruning-based identification of domain ontologies. *Journal of Universal Computer Science*, 9(6):520–529, 2003.